

# Introduction au Deep Learning

Basé sur Keras-Tensorflow

Vincent Boyer

vincent.boyer1@live.fr

Mars 2018

# Contents

<b>1</b>	<b>Préambule</b>	<b>14</b>
<b>2</b>	<b>Les Fondamentaux</b>	<b>14</b>
2.1	Un neurone: le perceptron . . . . .	15
2.2	Le biais . . . . .	16
2.3	Le Perceptron Multicouche . . . . .	17
2.4	L'apprentissage du neurone . . . . .	18
2.4.1	Forward . . . . .	19
2.4.1.1	Fonction de coût . . . . .	19
2.4.1.2	Fonction d'activation . . . . .	20
2.4.2	Backward . . . . .	22
2.4.2.1	Rétropropagation du gradient . . . . .	23
2.5	Descente du gradient et optimiseur . . . . .	25
2.5.1	Approche par Batch, Minibatch et Stochastique . . . . .	25
2.5.1.1	Approche par Batch . . . . .	25
2.5.1.2	Approche Stochastique . . . . .	25
2.5.1.3	Approche par MiniBatch . . . . .	26
2.5.2	Aperçu des optimiseurs . . . . .	26
2.5.3	Gradient noise . . . . .	28
2.5.4	Gradient Clipping . . . . .	28
2.6	ReLU et les dangers du gradient . . . . .	29
2.6.1	Vanishing Gradient . . . . .	29
2.6.2	Exploding Gradient . . . . .	29
2.6.3	Fonction ReLu et Dead ReLu . . . . .	29
2.7	Initialisation des poids . . . . .	33
2.8	Jeu d'apprentissage et spécificités . . . . .	34
2.9	Prédiction multi-label et multi-classe . . . . .	35
2.9.1	Généralités . . . . .	35
2.9.2	Prédiction et distribution de données . . . . .	37
2.9.3	Méthodes d'apprentissage . . . . .	38
2.9.3.1	Au niveau des données . . . . .	38
2.9.3.2	Au niveau du modèle . . . . .	39
2.10	Calcul matriciel et neurones . . . . .	40
<b>3</b>	<b>Normalisation des données</b>	<b>42</b>
3.1	Centrer les données . . . . .	43
3.2	Réduire les données . . . . .	43
3.3	Centrer-Réduire les données . . . . .	43
3.4	Analyse en Composante Principale . . . . .	44

<b>4 Régularisation et sur-apprentissage</b>	<b>45</b>
4.1 Le sur-apprentissage . . . . .	45
4.2 Limiter le sur-apprentissage . . . . .	46
4.3 Régularisation . . . . .	48
4.3.1 L1-Régulation . . . . .	48
4.3.2 L2-Régulation . . . . .	48
4.3.3 ElasticNet-Régulation . . . . .	49
4.4 Max norm constraints . . . . .	49
4.5 DropOut . . . . .	49
4.6 DropConnect . . . . .	50
4.7 Dense-Sparse-Dense training . . . . .	51
4.8 Batch Normalization . . . . .	52
4.9 Critère d'arrêt de l'apprentissage . . . . .	53
4.9.1 Early Stopping . . . . .	54
4.9.1.1 Generalization Loss ( $GL_\alpha$ ) . . . . .	54
4.9.1.2 Quotient of Generalization Loss and Progress ( $PQ_{alpha}$ ) . . . . .	55
4.9.1.3 Successive Generalization Error ( $UP_s$ ) . . . . .	55
4.9.2 Arrêt supervisé . . . . .	56
4.10 Data Augmentation . . . . .	56
4.11 Complexité de l'architecture et mémoire . . . . .	58
4.12 Transfer Learning . . . . .	59
<b>5 Réseaux convolutifs</b>	<b>62</b>
5.1 Généralités . . . . .	62
5.2 Couche d'Entrée . . . . .	64
5.3 Couche de Convolution . . . . .	64
5.3.1 Nature d'une couche de convolution . . . . .	64
5.3.2 Interprétation graphique . . . . .	65
5.3.3 Couche de Pooling . . . . .	67
5.3.3.1 Global Pooling . . . . .	71
5.3.3.2 k-Max Pooling . . . . .	72
5.3.4 Couche de Unpooling . . . . .	73
5.3.5 Couche de convolution 1*1 . . . . .	74
5.3.6 Couche de convolution dilatée . . . . .	74
5.3.7 Couche de convolution transposée . . . . .	75
5.3.8 Couche de Depthwise/Pointwise . . . . .	76
5.4 Conversion d'une couche Full-Connected en couche de convolution	78
5.5 Convolutional Neural Network (CNN) et Fully Convolutional Network (FCN) . . . . .	79
5.6 Les modèles convolutifs de référence . . . . .	79
5.6.1 AlexNet . . . . .	79
5.6.2 ZF Net . . . . .	80
5.6.3 VGG Net . . . . .	80
5.6.4 GoogleNet/Inception . . . . .	80
5.6.5 Xception . . . . .	82

5.6.6	Microsoft Resnet . . . . .	83
5.6.7	ResNet et améliorations . . . . .	84
5.6.7.1	Wide Residual Network . . . . .	85
5.6.7.2	Aggregated Residual Transformations for Deep Neural Networks (ResNext) . . . . .	85
5.6.7.3	Densely Connected Convolutionnal Networks (DenseNet) . . . . .	86
5.6.7.4	Stochastic Depth . . . . .	89
5.6.7.5	Residual Networks of Residual Networks: Multilevel Residual Networks . . . . .	89
5.6.7.6	Sparcely Connected Convolutional Networks (SparseNet) . . . . .	90
5.6.7.7	Résumé des différentes approches appliquées aux ResNet . . . . .	91
5.6.8	FractalNet: Ultra-Deep Neural Networks without Residuals . . . . .	91
5.6.9	Classement des architectures standards . . . . .	92
5.6.10	Réseaux de neurones compressés . . . . .	93
5.6.10.1	SqueezeNet . . . . .	93
5.6.10.2	MobileNet . . . . .	96
5.6.10.3	Modèles expérimentaux récents . . . . .	96
5.6.11	Les jeux de données de référence . . . . .	97
<b>6</b>	<b>Encoder-Decoder</b>	<b>98</b>
6.1	Généralités . . . . .	98
6.2	Autoencoder . . . . .	99
6.2.1	Autoencoder Undercomplete . . . . .	100
6.2.2	Autoencoder Régularisé ou Overcomplete . . . . .	100
6.2.2.1	Denoising Autoencoder . . . . .	101
6.2.2.2	Sparse Autoencoder . . . . .	102
6.2.2.3	Contractive Autoencoders . . . . .	103
6.3	Variational Autoencoders . . . . .	106
<b>7</b>	<b>Réseaux antagonistes génératifs</b>	<b>108</b>
7.1	Généralités . . . . .	108
7.1.0.1	Fonction de perte . . . . .	109
7.2	Difficultés d'apprentissage . . . . .	111
7.3	Deep Convolutional Generative Adversarial Networks (DCGAN) . . . . .	112
<b>8</b>	<b>Réseaux siamois</b>	<b>112</b>
8.1	Généralités . . . . .	112
8.2	Triplet Loss . . . . .	114
<b>9</b>	<b>Réseaux récurrents</b>	<b>115</b>
9.1	Généralité . . . . .	115
9.2	Long Short-Term Memory (LSTM) . . . . .	115
9.3	Gated Recurrent Unit (GRU) . . . . .	116
9.4	Architecture-type . . . . .	116

<b>10 Deep Learning et Attention</b>	<b>117</b>
10.1 Généralités . . . . .	117
10.2 Soft Attention . . . . .	118
10.2.1 Fonction de proximité . . . . .	119
10.3 Hard Attention . . . . .	120
10.4 Réseaux récurrents . . . . .	120
10.4.1 Approche Bahdanau . . . . .	121
10.4.2 Approche Luong . . . . .	122
10.4.2.1 Attention globale et locale . . . . .	123
10.4.3 Généralisation pour l'exploitation d'image: Image Captionning . . . . .	124
10.5 Réseaux convolutifs . . . . .	125
10.5.1 Spatial Transformer . . . . .	126
10.5.1.1 Transformation d'image . . . . .	127
10.5.1.2 Interpolation bilinéaire . . . . .	128
10.5.1.3 Le module SNT . . . . .	129
10.5.2 Squeeze-and-Excitation Networks (SENet) . . . . .	132
10.5.2.1 Concurrent Spatial and Channel Squeeze and excitation . . . . .	132
<b>11 L'analyse d'image et ses formes</b>	<b>133</b>
11.1 Les différentes thématiques de l'analyse d'image . . . . .	133
11.2 Généralités théoriques . . . . .	138
11.2.1 Théorie - Object Detection, un problème de Régression . . . . .	138
11.2.2 Théorie - Landmark detection, un problème de Régression . . . . .	139
11.2.3 Sliding Windows . . . . .	140
11.2.4 Sliding Windows et réseau convolutif . . . . .	142
11.2.5 Region Proposal . . . . .	142
11.2.5.1 Selective Search . . . . .	143
11.2.6 Anchor Boxes . . . . .	144
11.2.7 RoI Pooling . . . . .	145
11.2.8 Intersection over Union (IoU) . . . . .	147
11.2.9 Non-Max Suppression . . . . .	148
11.2.10 Feature Pyramid . . . . .	150
11.2.10.1 Amélioration de l'architecture du réseau central . . . . .	152
11.2.10.2 A faire . . . . .	152
11.3 Object recognition: méthodes State-of-the-art . . . . .	152
11.3.1 R-CNN - Algorithme précurseur . . . . .	153
11.3.2 Spatial Pyramid Pooling - Algorithme précurseur . . . . .	153
11.3.3 Fast R-CNN - Algorithme précurseur . . . . .	156
11.3.4 Faster R-CNN . . . . .	156
11.3.5 Différence principale entre Single shot et Region based . . . . .	157
11.3.6 You Only Look Once (YOLO) . . . . .	160
11.3.6.1 YOLOv1 . . . . .	160
11.3.6.2 YOLOv2 - YOLO9000 . . . . .	164
11.3.6.3 YOLOv3 . . . . .	168

11.3.7	Single Shot MultiBox Detector (SSD) . . . . .	171
11.3.7.1	SSD . . . . .	171
11.3.7.2	Tiny SSD . . . . .	173
11.3.7.3	Deconvolutional Single Shot Detector (DSSD) .	174
11.3.8	RetinaNet . . . . .	176
11.3.8.1	Focal Loss . . . . .	176
11.3.8.2	Le modèle RetinaNet . . . . .	177
11.3.9	Comparatif des modèles récents . . . . .	178
<b>12 Application au traitement du langage écrit</b>		<b>178</b>
12.1	Introduction . . . . .	178
12.1.1	La loi de Zipf et Mandelbrot . . . . .	178
12.1.2	Pré-traitement d'un texte . . . . .	180
12.1.2.1	Tokenization . . . . .	181
12.1.2.2	Lemmatisation et racinisation . . . . .	181
12.1.2.3	Stopwords . . . . .	182
12.1.2.4	Problématique et pré-traitement . . . . .	182
12.2	Représentation vectorielle . . . . .	183
12.2.1	Projection Word-Based . . . . .	183
12.2.2	Projection Character-Based . . . . .	183
12.3	Classification . . . . .	183
12.3.1	Classification par CNN . . . . .	184
12.3.1.1	CNN for Sentence Classification . . . . .	184
12.3.1.2	Améliorations notables . . . . .	184
12.3.1.3	Very Deep CNN . . . . .	187
12.3.1.4	Comparatif expérimental sur les architectures CNN	187
12.3.2	Classification par RNN . . . . .	189
12.3.2.1	LSTM Deep Sentence Embedding . . . . .	189
12.3.2.2	Discriminative RNN . . . . .	189
12.3.2.3	Hierarchical Attention Networks . . . . .	189
12.3.3	Classification par CNN-RNN . . . . .	190
12.3.3.1	CNN-RNN . . . . .	190
12.3.3.2	C-LSTM . . . . .	192
12.3.3.3	AC-BLSTM . . . . .	194
<b>13 Apprentissage par Renforcement et Deep Learning</b>		<b>196</b>
13.1	Approche théorique . . . . .	196
13.1.1	Généralités . . . . .	196
13.1.1.1	L'environnement . . . . .	197
13.1.1.2	Formalisation du problème . . . . .	199
13.1.1.3	Définitions fondamentales . . . . .	200
13.1.1.4	L'équation fondatrice: l'équation de Bellman .	201
13.1.1.5	Model-Free et Model-Based . . . . .	202
13.1.1.6	Différence temporelle . . . . .	203
13.1.1.7	Un standard: l'algorithme Q-Learning . . . . .	204
13.1.1.8	Approximation des valeurs d'états . . . . .	207

13.2 Deep Q-Learning . . . . .	207
13.3 Prioritized Experience Replay . . . . .	209
13.4 Double Deep Q-Learning . . . . .	211
13.5 Dueling Deep Q-Learning . . . . .	212
13.6 Double Dueling Q-Learning . . . . .	213
<b>14 Lecture et approfondissement</b>	<b>213</b>
14.1 Fondamentaux . . . . .	213
14.2 Réseaux convolutifs . . . . .	214
14.3 Jeu de données d'apprentissage . . . . .	214
14.4 Recueil thématique d'articles de recherche . . . . .	214
14.4.1 Multi-thématique . . . . .	214
14.4.2 Analyse d'image - Toute thématique . . . . .	214
14.5 Image Segmentation . . . . .	214
<b>15 Installation de Tensorflow et Keras</b>	<b>215</b>
15.1 Installation de Tensorflow pour utilisation GPU sur Ubuntu . . .	215
15.2 Installation de Keras . . . . .	217
<b>16 Tensorflow, théorie et exemple</b>	<b>218</b>
16.1 Introduction . . . . .	218
16.2 Architecture d'exécution . . . . .	218
16.2.1 Généralités . . . . .	219
16.2.2 Tensor . . . . .	219
16.2.3 Instanciation et Session . . . . .	219
16.3 Fondamentaux du framework . . . . .	220
16.3.1 Déclaration des variables du graphe . . . . .	220
16.3.2 Nomination des variables . . . . .	222
16.3.3 Création d'une Session . . . . .	224
16.3.4 Application aux réseaux de neurones: l'exemple du Perceptron multicoche . . . . .	226
16.3.5 Exemple de Perceptron Multicoche avec une API haut niveau de Tensorflow: Layers . . . . .	231
16.3.6 Réaliser des graphiques de suivi avec Tensorboard . . . .	232
16.3.7 Interaction entre couches et isolation de variable . . . .	236
16.3.8 Sauvegarder et restaurer un modèle . . . . .	240
16.3.8.1 Structure de données . . . . .	240
16.3.8.2 Sauvegarder un modèle . . . . .	240
16.3.8.3 Importation d'un modèle . . . . .	242
<b>17 Ajouts à venir</b>	<b>243</b>
17.1 Application à la reconnaissance vocale . . . . .	243
17.2 Application au traitement du langage écrit . . . . .	243
17.2.1 Traduction - Neural Machine Translation . . . . .	243
17.2.2 Traduction - Attention is all your need . . . . .	243
17.2.3 Recherche d'informations . . . . .	243

17.2.4	Désambiguïsation d'entités . . . . .	243
17.3	L'analyse d'image et ses formes . . . . .	243
17.3.1	Segmentation: méthodes State-of-the-art . . . . .	243
17.3.2	Object Tracking: méthodes State-of-the-art . . . . .	243
17.4	Machine Learning et Éthique . . . . .	243
17.5	Deep Learning Bayésien . . . . .	243
17.6	Tutoriel applicatif de Keras . . . . .	243
<b>18</b>	<b>Appendix</b>	<b>243</b>
18.1	Object Detection API de Google/Tensorflow . . . . .	243
18.1.1	Installation . . . . .	244
18.1.2	Difficultés possibles: Version de Protobuf incorrecte . . . . .	244
18.1.3	Labellisation d'image . . . . .	244
18.2	Présentation de l'API . . . . .	246

## List of Figures

1	La structure d'un neurone (perceptron) . . . . .	16
2	Modèle FeedForward: Le Perceptron multicouche . . . . .	18
3	Fonctions d'activation: a) Sigmoïde, b) tanh, c) ReLu et leurs dérivées associées . . . . .	21
4	Rétropropagation du gradient - Cette image provient de [67] . .	23
5	Fonction ReLu et ses Variantes . . . . .	31
6	Exemple de deux filtres de phase opposée . . . . .	32
7	Exemple d'un jeu de données (toute ressemblance avec des données réelles est fortuite) . . . . .	42
8	Exemple de sur-apprentissage (bleu) et d'apprentissage optimal (vert) . . . . .	45
9	Compromis biais-variance et impact sur le modèle . . . . .	47
10	Exemple de sous-apprentissage . . . . .	47
11	Comparaison d'un réseau sous DropOut et DropConnect . . . . .	50
12	Relation des poids après DropOut . . . . .	50
13	Dense-Sparse-Dense Routine . . . . .	52
14	Non-alignement des distributions . . . . .	53
15	Normalisation par Batch Normalization . . . . .	53
16	a) Early Stop théorique b) Comportement-type de l'erreur de validation en situation réelle . . . . .	54
17	Exemple de courbe d'erreur d'apprentissage et de validation . .	57
18	Optimisation réalisée sur les matrices de poids par Deep Compression . . . . .	59
19	Architecture d'une couche de convolution . . . . .	65
20	Impact du stride sur la sortie de la convolution . . . . .	67
21	Exemple d'une convolution par filtre . . . . .	67
22	Problématique de la dimension du filtre: gauche (valide), droite (invalidé) . . . . .	68
23	Application du 0-Padding . . . . .	68
24	Application du Pooling: Réduction et exemple avec la fonction max selon un filtre 2*2 . . . . .	70
25	Exemple de Max-Unpooling statique . . . . .	73
26	Exemple de Max-Unpooling dynamique . . . . .	74
27	Exemple d'une convolution dilatée . . . . .	75
28	Exemple de convolution transposée: Gauche: entrée 2*2 sans stride, Droite: Entrée 3*3 avec stride de 1 . . . . .	76
29	Exemple d'une couche Depthwise/Pointwise avec 2 filtres . . .	77
30	Exemple d'une conversion de couche Full-Connected vers une structure convolutive . . . . .	79
31	Architecture de l'Inception module . . . . .	81
32	Architecture de l'architecture Xception: à gauche, un block standard et à droite un block simplifié (topologie uniforme) . . . . .	83

33	Architecture de l'architecture Xception: à gauche, un block Inception reformulé et à droite un block Inception dans sa forme extrême . . . . .	83
34	Architecture du Residual Block . . . . .	84
35	Intuition sur la notion de Résidus . . . . .	84
36	Architecture d'un Wide Residual block (les étapes de Batch normalisation) et de ReLu ne sont pas affichées) - la largeur des rectangles représentant les couches déterminent graphiquement le nombre de filtres employés pour la couche associée . . . . .	85
37	Différentes représentations de l'architecture d'un block du réseau ResNext: 1) Approche ResNet, 2) Approche Inception, 3) Approche par convolution groupée . . . . .	86
38	Architecture d'un réseau Densely Connected . . . . .	88
39	Réseau DenseNet . . . . .	88
40	Stochastic Depth routine . . . . .	89
41	Exemple d'un réseau RoR de niveau 3 . . . . .	90
42	Différence entre un block issu de ResNet, DenseNet et SparseNet	91
43	Construction d'un bloc du réseau FractalNet . . . . .	93
44	Analyse des réseaux de Deep Learning (2017) - Ce graphique a été créé par Alfredo Canziani, Adam Paszke et Eugenio Culurciello	94
45	Architecture du Fire Module . . . . .	95
46	Architecture d'un Autoencoder . . . . .	100
47	Architecture d'un Denoising Autoencoder . . . . .	101
48	Filtre d'un k-Sparse Autoencoder selon la valeur de k sur le jeu de données MNIST . . . . .	104
49	Architecture d'un Variationnal Autoencoder . . . . .	107
50	Détermination du vecteur de contexte par un Variationnal Autoencoder . . . . .	108
51	Generation d'image par un Variationnal Autoencoder entraîné sur MNIST . . . . .	108
52	Architecture d'un réseau antagoniste génératif . . . . .	109
53	Comparaison des gradients de la fonction de perte du Générateur	111
54	Exemple simple d'un réseau siamois . . . . .	114
55	Architecture-type d'un réseau récurrent . . . . .	117
56	Performance de la traduction selon la dimension du texte d'entrée	118
57	Schématisation de l'approche Soft Attention . . . . .	119
58	Schématisation de l'approche Hard Attention . . . . .	120
59	Visualisation du comportement de Soft Attention et Hard Attention	120
60	Attention et réseaux récurrents . . . . .	121
61	Attention selon Bahdanau . . . . .	122
62	Attention selon Luong . . . . .	123
63	Extraction des vecteurs de contexte à partir d'une feature map .	126
64	Illustration de l'interpolation bilinéaire . . . . .	129
65	Illustration du module SNT . . . . .	130
66	Illustration d'une transformation par SNT avec le dataset MNIST	131

67	Comparaison entre un module ResNet standard et un module SE-Resnet . . . . .	134
68	Illustration des blocs sSE et scSE . . . . .	134
69	Exemple d'analyse d'image: 1) Classification, 2) Object detection, 3) Object recognition, 4) Segmantic segmentation, 5) Instance segmentation, 6) Object proposal, 7) Text Detection, 8) landmark Detection (Pose estimation), 9) Image Captionning . . . . .	137
70	Exemple - Object Detection . . . . .	139
71	Exemple - Landmark Detection . . . . .	140
72	Exemple - Sliding Windows . . . . .	141
73	Limitation du Sliding Windows . . . . .	142
74	Exemple - Convolute Sliding Windows . . . . .	143
75	Exemple - Graph-based Segmentation . . . . .	144
76	Exemple - Selective Search . . . . .	145
77	Exemple - Anchor Boxes . . . . .	146
78	Exemple - RoI Pooling . . . . .	147
79	Illustration de la métrique Intersection over Union (IoU) . . . . .	149
80	Illustration de la problématique de la superposition de bounding boxes . . . . .	150
81	Illustration de Non-Max Detection . . . . .	150
82	Illustration de l'architecture Feature Pyramid: a) Architecture générale b) Architecture des connexions latérales . . . . .	151
83	Illustration d'un réseau de prédiction de bounding boxes selon R-CNN . . . . .	154
84	Illustration d'un réseau R-CNN . . . . .	154
85	Illustration du Spatial Pyramid Pooling . . . . .	155
86	Illustration du Spatial Pyramid Pooling Network . . . . .	155
87	Analogie de RoI Pooling avec Spatial Pyramid Pooling . . . . .	156
88	Illustration du Fast R-CNN . . . . .	157
89	Illustration du Faster R-CNN . . . . .	158
90	Comparatif de performance entre R-CNN, Fast R-CNN et Faster R-CNN . . . . .	158
91	Différence de la sortie des couches convolutives entre les approches Single shot et Region based . . . . .	159
92	Fonctionnement général de l'algorithme YOLOv1 . . . . .	164
93	Architecture des couches Full-Connected de l'algorithme YOLOv1	165
94	Architecture de l'algorithme YOLOv1 . . . . .	165
95	Différence de centre entre un quadrillage pair et impair . . . . .	166
96	Conversion des prédictions du réseau YOLOv2 selon l'anchor associée . . . . .	167
97	Architecture du réseau convolutif YOLOv2 . . . . .	168
98	Architecture du réseau convolutif YOLOv3 . . . . .	171
99	Architecture du réseau SSD . . . . .	173
100	Architecture de Multibox . . . . .	173
101	Architecture de Deconvolutional Single Shot Detector (DSSD) . . . . .	175
102	Architecture du Deconvolution module de DSSD . . . . .	175

103	Architecture du module prédictif de DSSD . . . . .	176
104	Erreur obtenue par Focal Loss selon $\gamma$ . . . . .	177
105	Architecture du réseau RetinaNet . . . . .	178
106	Comparatif des modèles d'Object Detection . . . . .	179
107	Comparatif des modèles d'Object Detection pour un mAP à 0.5 IoU . . . . .	179
108	Loi de Zipf pour K=1 . . . . .	180
109	Architecture de CNN for Sentence Classification . . . . .	184
110	Architecture de Dynamic Convolutional Neural Network . . . . .	185
111	Architecture de Multi Channel Variable size CNN . . . . .	186
112	Architecture de Very Deep CNN . . . . .	187
113	Architecture de LSTM Deep Sentence Embedding . . . . .	189
114	Architecture de Discriminative RNN . . . . .	190
115	Architecture de Hierarchical Attention Networks . . . . .	191
116	Architecture de CNN-RNN . . . . .	192
117	Architecture de C-LSTM . . . . .	193
118	Architecture de C-LSTM avec Pooling . . . . .	193
119	Architecture de AC-BLSTM . . . . .	195
120	Harmonisation des dimensions des feature map selon l'approche de AC-BLSTM . . . . .	195
121	Graphique séquentiel de l'approche par renforcement . . . . .	198
122	Algorithme du TD(0) . . . . .	204
123	Algorithme du Q-Learning(0) . . . . .	206
124	Algorithme du Deep Q-Learning . . . . .	209
125	Algorithme du Dueling Deep Q-Learning . . . . .	213
126	Logo des framework Tensorflow et Caffe . . . . .	218
127	Représentation d'un Tensor de dimension 1, 2 et 3 . . . . .	219
128	Exemple d'interface basique de Tensorboard . . . . .	237
129	Exemple d'interface avancée de Tensorboard . . . . .	238
130	Block Resnet d'illustration . . . . .	238
131	Utilisation de LabelImg . . . . .	247

## Listings

1	Visualisation sous Tensorflow des GPU disponibles . . . . .	217
2	Visualisation sous Keras des GPU disponibles . . . . .	218
3	Déclaration d'une constante avec Tensorflow . . . . .	220
4	Déclaration bas niveau d'un Tensor variable avec Tensorflow . . .	221
5	Déclaration haut niveau d'un Tensor variable avec Tensorflow . .	221
6	Déclaration d'un Placeholder . . . . .	222
7	Gestion du namespace d'une variable par variable_scope() . . . .	222
8	Gestion du namespace d'une variable par name_scope() . . . .	223
9	Sélection par condition d'un sous-ensemble de variable . . . . .	224
10	Création d'une Session . . . . .	225
11	Création d'une Session avec un bloc "with" . . . . .	226
12	Perceptron multicouche: Initialisation des variables . . . . .	227
13	Perceptron multicouche: Architecture du réseau . . . . .	228
14	Perceptron multicouche: Fonction de coût et optimisation . . . .	229
15	Perceptron multicouche: Création de la Session . . . . .	230
16	Perceptron multicouche: Création de l'architecture du réseau avec Layers . . . . .	231
17	Tensorboard: Récupération des variables à observer . . . . .	233
18	Tensorboard: Création du fichier de log . . . . .	233
19	Tensorboard: Lancement du serveur local . . . . .	234
20	Tensorboard: Création du fichier de log . . . . .	235
21	Implémentation basique d'un block resnet . . . . .	236
22	Choix des poids à mettre à jour lors d'une rétropropagation du gradient . . . . .	239
23	Sauvegarder un modèle . . . . .	240
24	Sauvegarder un modèle sous condition . . . . .	241
25	Importer un modèle . . . . .	242
26	Importer un modèle . . . . .	242
27	Installation de protobuf3 . . . . .	244
28	Exemple de labellisation au format PASCAL VOC d'une image contenant uniquement un alpaga . . . . .	245

## 1 Préambule

Cette introduction n'a pas vocation à introduire les concepts mathématiques associés aux réseaux de neurones de manière approfondie. Elle se limitera à survoler les bases d'un réseau de neurones afin de fournir une vision d'ensemble et une sensibilité permettant au lecteur, par la suite, de se former de manière autonome.

On supposera les fondamentaux de l'apprentissage automatique comme acquis, i.e les différents types d'apprentissage (supervisé, non supervisé, ...), les méthodes de cross-validation rudimentaires (séparation des jeux de données) et les bases mathématiques telles que les fondamentaux en Analyse (fonctions utilitaires, dérivation et convergence) et en Statistiques (estimateurs (moyenne, variance,...), corrélation, distribution).

## 2 Les Fondamentaux

Le deep Learning est une catégorie d'algorithmes basée sur un assemblage spécifique d'entités nommées **neurones**<sup>1</sup>. Ces entités sont assemblées sous forme de couches plus ou moins nombreuses et profondes, à l'architecture variable selon la complexité du modèle recherché et du problème à résoudre.

L'objectif du machine learning est d'approximer une fonction hypothèse définie par un jeu de données. Cette fonction hypothèse doit être généralisable à des données non représentées dans le jeu de données. L'approche du Deep Learning est de modéliser les données avec une haute capacité d'abstraction où chaque couche extrait et transforme les données issues de la couche précédente<sup>2</sup> avec une approche majoritairement **non linéaire**. Un réseau possède donc un niveau d'abstraction dépendant de son architecture. Il n'y a pas de méthodologie reconnue pour déterminer l'architecture *idéale* pour une problématique donnée. Seule l'approche empirique est employée à ce jour.

Aujourd'hui, le Deep Learning est très populaire pour ses résultats qui sont, dans de nombreux domaines, les meilleurs de l'état de l'art. Néanmoins, sa grande qualité repose sur sa capacité à s'émanciper du travail de pré-traitement des données nécessaires aux méthodes plus classiques. Par exemple, une image, pour être exploitée par des modèles standards, doit être pré-traitée pour extraire son contenu utile. Ces méthodes d'extraction sont généralistes, lourdes à employer et pas toujours efficaces. Les modèles de Deep Learning (notamment convolutifs<sup>3</sup>) ont la capacité d'apprendre dynamiquement l'extraction du contenu utile d'une image. Ainsi, le réseau se spécialise - sans intervention humaine - sur le type d'image qu'il traite et bien souvent, dépasse les performances

---

<sup>1</sup>Une analogie peut être réalisée avec le cerveau humain

<sup>2</sup>Selon le type d'architecture, il peut y avoir des phénomènes de rétro-action

<sup>3</sup>Nous l'étudierons dans cette introduction

de méthodes de pré-traitement standards. Cette capacité d'auto-apprentissage rend le Deep Learning modulable, simple<sup>4</sup> et facilement déployable. En effet, en simplifiant, il suffit de donner la donnée brute et le réseau s'adaptera seul. Cette capacité se généralise à différents types de données: signal 1D (texte, signal sonore), signal 2D (image), séries temporelles... Cependant, il est nécessaire d'adapter l'architecture du modèle au types de données et du problème ciblé.

Cette souplesse d'utilisation a un coût non négligeable: **le volume de données**. Pour être fonctionnel, un réseau de Deep Learning doit exploiter un volume très important de données durant son apprentissage<sup>5</sup>. Aujourd'hui, peu d'acteurs du milieu public ou privé ont un volume suffisant à disposition. Le Deep Learning ne constitue donc pas un remplaçant absolu aux approches de Machine Learning traditionnelles mais une alternative lorsque l'apprentissage peut être réalisé dans de bonnes conditions. De même, les architectures de Deep Learning sont exigeantes en temps machine et en ressources. Il est parfois pertinent d'exploiter des modèles plus légers et accessibles lorsque la tâche à accomplir ne nécessite pas un modèle à forte complexité.

De ce fait, aujourd'hui, pour les tâches d'*optimisation*, le Machine Learning traditionnel garde une place dominante alors que sur des thématiques complexes ou créatives (génération automatique, traduction...), le Deep Learning devient nécessaire.

**Important:** Le Deep Learning profite d'une médiatisation importante qui tend à extrapoler ses qualités. Il est crucial de comprendre en profondeur les qualités et les défauts de ce type de modèle afin de ne pas alimenter les risques d'un développement non maîtrisé et biaisé.

## 2.1 Un neurone: le perceptron

Le modèle le plus simple correspond au modèle unitaire: un seul neurone. Ce modèle est appelé **perceptron** et a été inventé en 1957 par Frank Rosenblatt. La Figure 1 représente un neurone.

Le perceptron (neurone) **selon Rosenblatt** est défini par différents éléments. Ainsi, un neurone  $j$  est décrit par:

- Des données d'entrée:  $x_1, x_2, \dots, x_n$
- Des poids (qui pondèrent les données d'entrée):  $\omega_1, \dots, \omega_n$
- Un biais (qui régule l'activation du neurone):  $b_j$

---

<sup>4</sup>La simplicité est relative...

<sup>5</sup>Le volume dépend de la tâche à accomplir et de la nature des données

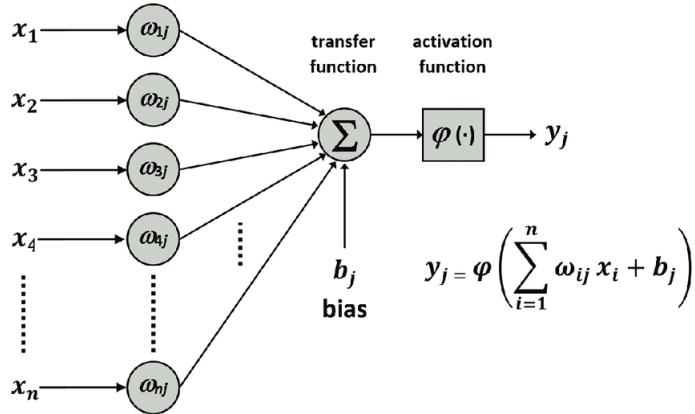


Figure 1: La structure d'un neurone (perceptron)

- Une fonction d'activation (qui modélise la réponse du neurone en fonction des entrées pondérées):  $\varphi()$  avec  $\varphi = \begin{cases} 0 & \text{si } x < 0 \\ 1 & \text{si } x > 0 \end{cases}$  (fonction heavyside<sup>6</sup>)
- Une réponse (ou sortie):  $y_j$

Le fonctionnement d'un neurone est simple.

Supposons une entrée  $[x_1, \dots, x_n]$  (de dimension n). Chaque entrée i est pondérée par un poids  $\omega_i$  associé<sup>7</sup>. Ainsi:  $[x_1, \dots, x_n] \xrightarrow{\omega} [x_1 * w_1, \dots, x_n * w_n]$

Une somme est réalisée de l'ensemble des données d'entrée pondérées et le biais  $b_j$ . On obtient donc:  $\text{transfer\_function} = \sum_{i=1}^n (x_i * w_i) + b_j$

Le biais est souvent représenté comme une valeur d'entrée constante (supposons 1 par exemple) et pondérée par un poids comme le serait une autre entrée. Ainsi, nous obtenons:  $\text{transfer\_function} = \sum_{i=1}^n (x_i * w_i) + w_0 = \sum_{i=0}^n (x_i * w_i)$  avec  $x_0 = 1$

La somme obtenue est transformée par une fonction d'activation  $\varphi()$  et définira la sortie  $y$  du neurone:  $y = \varphi(\text{transfer\_function})$

## 2.2 Le biais

Le biais est utilisé pour éviter que l'hyperplan réalisé par le neurone ait l'obligation de passer par l'origine. L'impact, la détermination et la représentation du biais sont encore un sujet d'étude et de recherche. Afin de ne pas complexifier cette introduction avec des approches mathématiques secondaires, nous n'approfondirons pas cette problématique. Cependant, il est important d'avoir

<sup>6</sup>Nous verrons par la suite que c'est un très mauvais choix de fonction !

<sup>7</sup>il y a donc n poids

une sensibilité sur son utilité. Un exemple simple permet de la comprendre.

Supposons une image complètement noire (pixels à 0) et une sortie souhaitée égale à  $\lambda$ . Cette problématique est insoluble car, avec les fonctions d'activation standards, la valeur de sortie par rapport à une stimulation nulle est invariable et correspond à la valeur de la fonction d'activation en 0. Supposons la fonction heavyside, la sortie serait donc de 0.5 et constante.

Néanmoins, dans le cadre d'un réseau profond, le biais tend à voir son impact diminuer avec la complexité du modèle où le comportement d'activation est partiellement réalisé par des neurones de la couche précédente. Ainsi, dans le contexte du deep learning où les modèles se complexifient énormément, la gestion du biais peut être simplifiée voir "délaisse".

Dans les réseaux profonds, le biais est aussi utilisé pour se protéger des défauts liées à l'initialisation des neurones, notamment le phénomène *Dead ReLu*<sup>8</sup>.

### 2.3 Le Perceptron Multicouche

Un neurone unique possède des qualités limitées<sup>9</sup> et ne peut résoudre que des problèmes simples<sup>10</sup>. Afin de résoudre des problèmes plus complexes, il est nécessaire d'en assembler plusieurs afin de créer une architecture plus performante. La première architecture que nous verrons est le modèle *FeedForward*.

Le modèle FeedForward est caractérisé par des couches de neurones où chaque neurone est connecté à l'intégralité des neurones de la couche suivante, permettant une *propagation en avant* de l'information. Le réseau le plus populaire se reposant sur ce modèle est le Perceptron Multicouche. La Figure 2 en présente un exemple.

Ainsi, les données d'entrées sont réparties sur la couche d'entrée composée de 1 ou plusieurs neurones où chaque donnée est envoyée à l'intégralité des neurones. Les sorties de ces neurones sont connectées à l'intégralité des neurones de la couche suivante, appelée couche cachée. Il peut y avoir plusieurs couches cachées. Les données d'entrées des couches cachées sont donc les sorties des neurones de la couche précédente. La couche de sortie correspond à la dernière couche du réseau et la sortie des neurones de cette couche correspond à la prédiction réalisée par le réseau. Ainsi, par exemple, dans un problème de classification à N classes<sup>11</sup>, la couche de sortie sera composée de N neurones où les N sorties correspondent à la prédiction associée à chacune des classes (souvent sous forme probabiliste).

---

<sup>8</sup>Voir Section 2.4.1, partie fonction d'activation, ReLu

<sup>9</sup>Par exemple, il ne peut traiter que des problèmes linéairement séparables

<sup>10</sup>Ils sont souvent employés pour simuler des portes logiques par exemple

<sup>11</sup>Voir Section 2.9 pour plus de détails

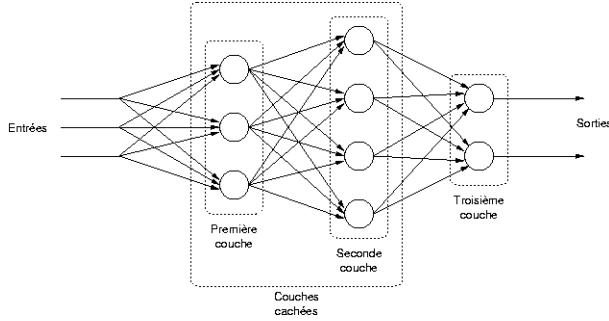


Figure 2: Modèle FeedForward: Le Perceptron multicouche

Le nombre de neurones par couche et le nombre de couche nécessaire pour résoudre un problème ne peuvent être déterminés à l'avance. Il n'existe pas de méthode reconnue pour obtenir le réseau "idéal" pour résoudre un problème. Néanmoins, plus le nombre de neurones et/ou de couches est élevé, plus le modèle possède un haut pouvoir d'abstraction. Avoir une haute capacité d'abstraction est nécessaire sur des problématiques complexes mais augmenter la taille du réseau ne garantit pas un meilleur résultat sur l'ensemble des problèmes et présente des risques non négligeables que nous approfondirons dans la suite de cette introduction.

En théorie, un perceptron multicouche à une couche cachée est capable d'approximer toute fonction d'un sous-ensemble compact de  $R^n$  avec un nombre fini de neurones.

## 2.4 L'apprentissage du neurone

Un neurone est défini par ses poids (et son biais). Lors de la création du neurone, les poids sont générés arbitrairement. Il est donc nécessaire de lui permettre *d'apprendre*, i.e mettre à jour ses poids, afin qu'il puisse limiter ses erreurs de prédiction, i.e minimiser la valeur de la fonction de coût. Cet objectif est réalisé en deux étapes: l'étape *Forward* et l'étape *Backward*

L'étape *Forward* consiste à déterminer une prédiction réalisée par le neurone et à calculer l'erreur (potentiellement cumulée) réalisée par rapport à une valeur de référence indiquée par les données d'apprentissage. L'étape *Backward* réalise la mise à jour des poids du neurone en fonction de l'erreur réalisée grâce à l'algorithme de **Rétropropagation du gradient**.

## 2.4.1 Forward

### 2.4.1.1 Fonction de coût

Afin de pouvoir mettre à jour les poids, il est nécessaire de savoir lorsque le neurone s'est trompé et quelle est l'ampleur de son erreur. La fonction de coût est utilisée pour quantifier cette erreur. Il existe différentes fonctions de coût selon la problématique à traiter. Les principales sont définies par:

- Problème de **régession**:

- Mean Squared Error:  $\mathcal{L} = \frac{1}{n} \sum_{i=1}^n (y^{(i)} - \hat{y}^{(i)})^2$

Cette fonction est la fonction la plus répandue et standard. Sa forme quadratique justifie qu'elle est convexe et ainsi, favorise l'apprentissage du réseau de neurone qui s'apparente à un problème d'optimisation. Cette fonction, du fait de la mise au carré, maximise l'importance des *grandes erreurs*. C'est une spécificité importante à considérer car elle rendra l'apprentissage sensible aux valeurs extrêmes. De plus, cette fonction favorise une vitesse de convergence relativement lente du réseau de neurone.

- Mean Squared Logarithmic Error:  $\mathcal{L} = \frac{1}{n} \sum_{i=1}^n (\log(y^{(i)}) + 1) - \log(\hat{y}^{(i)} + 1))^2$

Cette fonction est une variante de Mean Squared Error à la différence qu'elle est moins sensible aux *grandes erreurs* de prédiction. Dans le cas de données non normalisées aux échelles différentes, cette fonction limite la pénalisation des grands écarts de prédiction qui peuvent être liés, dans ce contexte, à l'échelle des données et non une véritable erreur de prédiction.

- Problème de **classification**:

- Cross Entropy:  $\mathcal{L} = -\frac{1}{n} \sum_{i=1}^n [y^{(i)} \log(\hat{y}^{(i)}) + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})]$

Cette fonction est la fonction standard dans le cadre de la classification binaire. Contrairement aux fonctions quadratiques, elle possède une vitesse de convergence plus rapide et tend à favoriser la convergence vers le minimum global (mais rien ne garantit cela !)

- Negative Logarithmic Likelihood:  $\mathcal{L} = -\frac{1}{n} \sum_{i=1}^n \log(\hat{y}^{(i)})$

Cette fonction de coût est employée dans le cadre de problème de classification multi-classe. Elle peut être considérée comme une généralisation de la Cross Entropy.

Il existe de nombreuses autres fonctions de coût aux spécificités particulières. Le choix de la fonction de coût demande de l'intuition et une certaine sensibilité mathématique. Cependant, son choix est crucial pour un fonctionnement optimal du réseau de neurones. Le travail d'analyse des différentes fonctions est une étape indispensable pour l'étude de problèmes complexes.

#### 2.4.1.2 Fonction d'activation

**Important:** Cette partie nécessite d'avoir une compréhension générale de la Partie 2.4.2 pour en comprendre pleinement les problématiques.

Bien que le perceptron de Rosenblatt utilise la fonction Heavyside comme fonction d'activation, il est possible d'en utiliser d'autres. Comme pour le choix de la fonction de coût, le choix de la fonction d'activation est capitale dans le développement d'un réseau de neurones. Il n'y a pas de méthodes clairement définies dans le choix de cette fonction. Les méthodes sont exclusivement empiriques bien que certaines fonctions semblent "sortir du lot" [46].

Pour être pleinement fonctionnelle, les fonctions d'activations doivent présenter des caractéristiques particulières:

- **Être dérivable en tout point:** Le gradient étant dépendant de la dérivée de la fonction d'activation, une fonction non dérivable est difficilement utilisable.
- **La dérivé doit être non nulle:** Si la dérivée de la fonction d'activation est nulle, le gradient sera nul. De ce fait, l'apprentissage est impossible. Ce problème justifie la non-efficacité de la fonction Heavyside du perceptron standard car il ne peut pas apprendre en faisant varier ses poids.
- **Être non linéaire:** Cette condition n'est pas une nécessité absolue mais permet de considérer les problématiques non linéairement séparables.

Les principales fonctions d'activation actuelles sont:

- Fonction Sigmoïde:  $f(x) = \frac{1}{1+e^{-x}}$

La fonction sigmoïde est une référence historique en tant que fonction d'activation et reste aujourd'hui, très utilisée. Elle est appréciée pour sa dérivée non constante, ce qui permet d'avoir un gradient variable selon la valeur et donc, une convergence plus performante. Néanmoins, le fait qu'elle soit positive la rend peu performante sur certains problèmes. De plus, cette fonction favorise le phénomène de *vanishing gradient*<sup>12</sup>. Un effet de bord est aussi présent. En effet, au-delà de (-3,3), la valeur de la dérivée est très faible ce qui impliquera un gradient faible. Un gradient

---

<sup>12</sup>Cette notion sera étudiée par la suite

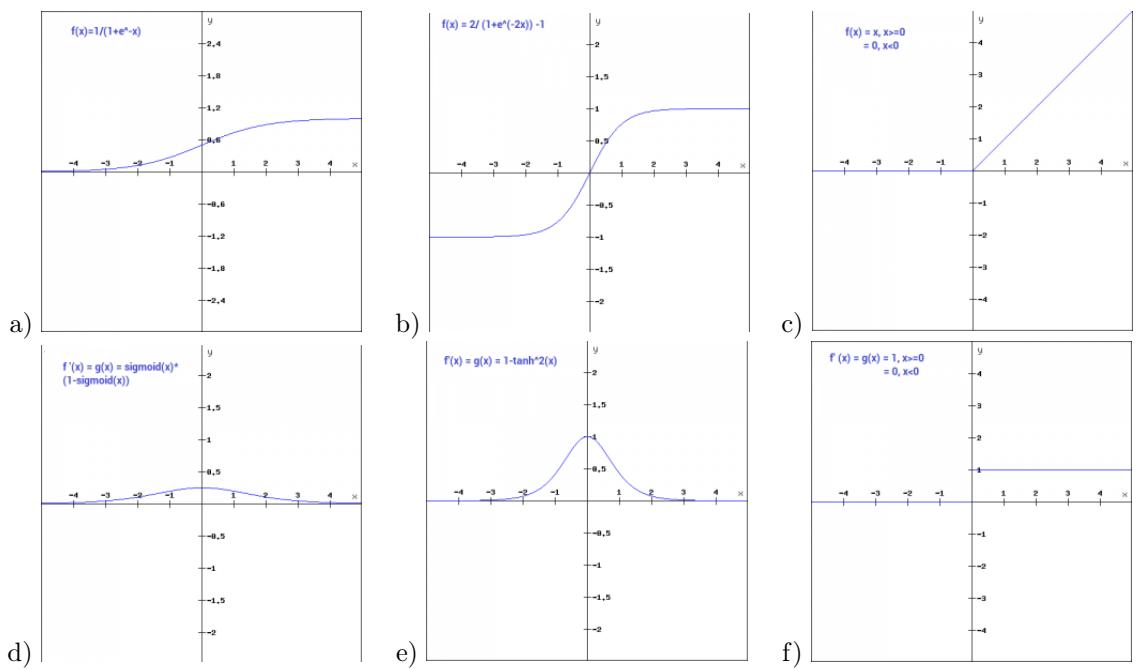


Figure 3: Fonctions d'activation: a) Sigmoid, b) tanh, c) ReLu et leurs dérivées associées

très faible limitera grandement les variations des poids et augmentera le temps d'apprentissage du neurone (voire le "freezera").

Cette fonction est souvent utilisée sur la couche de sortie pour représenter une prédiction sous forme de probabilité (car résultat entre 0 et 1). C'est la fonction de référence dans le cadre de la classification binaire probabiliste.

**Remarque:** Un perceptron avec la sigmoïde comme fonction d'activation est identique à une régression logistique<sup>13</sup>.

- Fonction tanh:  $f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$

La fonction tanh présente les mêmes caractéristiques que la fonction sigmoïde. La différence se situe sur le fait qu'elle est symétrique en 0 et prend valeur dans (-1,1), ce qui lui permet de ne pas être limitée par une condition de valeur positive. De plus, on observe un pic plus prononcé sur sa dérivée, ce qui implique un gradient plus important. Ceci est une qualité et un défaut car selon le problème, ça peut accélérer la convergence ou, au contraire, l'empêcher de pleinement converger.

- Fonction ReLu:  $f(x) = \begin{cases} 0 & \text{si } x < 0 \\ \max(0, x) & \text{si } x \geq 0 \end{cases}$

La fonction ReLu est, aujourd'hui, la fonction d'activation la plus employée dans les réseaux de neurones au niveau des couches cachées<sup>14</sup>. Cette fonction présente des particularités très importantes dans le cadre des réseaux profonds. Nous l'étudierons en profondeur dans la suite de cette introduction.

- Fonction Softmax:  $\delta(z)_j = \frac{e^{z_j}}{\sum_{i=1}^N e^{z_i}}$  avec  $z = (z_1, \dots, z_N)$  et  $j \in [1, \dots, N]$

La fonction Softmax est une version généralisée de la fonction Sigmoïde. Elle est ainsi utilisée dans la couche de sortie pour des problèmes de classifications multi-classes (voir Section 2.9) avec représentation probabiliste. Elle est employée lorsque la couche de sortie possède différents neurones associés aux classes prédictes afin d'extraire la classe dont la probabilité est la plus élevée.

### 2.4.2 Backward

La fonction de coût détermine l'erreur du neurone durant son apprentissage. L'étape suivante est de permettre au neurone d'apprendre, d'évoluer en fonction

---

<sup>13</sup>Soyez fier de vous. Nous venons de réinventer la roue !

<sup>14</sup>Cette fonction présente peu d'intérêt en couche de sortie du fait de son faible pouvoir explicatif

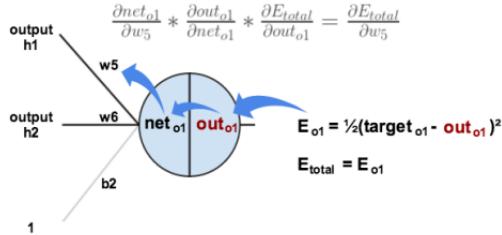


Figure 4: Rétropropagation du gradient - Cette image provient de [67]

de l'erreur réalisée et de faire varier chaque poids selon son impact dans l'erreur produite. Pour cela, la *descente de gradient*<sup>15</sup> est utilisée *localement* et son application récursif sur les différents neurones du réseau forme l'algorithme de **Rétropropagation du gradient**.

#### 2.4.2.1 Rétropropagation du gradient

Cette introduction n'approfondira pas le détail mathématique de son fonctionnement dans le cadre des réseaux de neurones. Néanmoins, il est indispensable de comprendre la démarche suivie car une grande partie des paramétrages d'un réseau repose sur une optimisation de son fonctionnement.

Supposons le neurone présenté sur la Figure 4. Le neurone a deux entrées  $h_1$  et  $h_2$  pondérées respectivement par  $w_5$  et  $w_6$ . La somme des entrées pondérées et du biais est représentée par  $net_{o1}$  et la sortie du neurone est  $out_{o1}$ . La fonction d'activation est la fonction sigmoïde. On considère la fonction de perte définie par:  $\mathcal{L} = \frac{1}{2} * (y - y')^2$ . L'étape Forward produit une erreur  $E_{total} = \frac{1}{2} * (target_{o1} - out_{o1})^2$  avec target, la valeur à atteindre et out, la sortie du neurone.

L'objectif de l'étape Backward est de calculer une correction d'erreur pour chaque poids indépendamment. Pour cela, il est nécessaire de définir la contribution d'un poids sur l'erreur produite. Pour cela, nous utiliserons la notion de *dérivé partielle*. Ainsi, pour un poids  $i$  d'un perceptron, il faut déterminer  $\frac{\partial E_{total}}{\partial \omega_i}$ .

Considérons le poids  $\omega_5$ . Nous voulons donc déterminer  $\frac{\partial E_{total}}{\partial \omega_5}$ . Par un jeu d'écriture, nous avons donc:  $\frac{\partial E_{total}}{\partial \omega_5} = \frac{\partial E_{total}}{\partial out_{o1}} * \frac{\partial out_{o1}}{\partial net_{o1}} * \frac{\partial net_{o1}}{\partial \omega_5}$

Avec ce jeu d'écriture, nous pouvons facilement déterminer les différentes composantes de la relation. Ainsi:

---

<sup>15</sup>Il s'agit d'une méthode d'optimisation simple mais efficace. Il est possible d'en exploiter d'autres mais à ce jour, cette approche fait consensus dans la communauté scientifique.

- $\frac{\partial E_{total}}{\partial out_{o1}} = -(target_{o1} - out_{o1})$

Cette valeur représente l'impact de la sortie du neurone sur l'erreur produite par ce dernier.

- $\frac{\partial out_{o1}}{\partial net_{o1}} = out_{o1} * (1 - out_{o1})^{16}$

Cette valeur représente l'impact de la somme pondérée sur la sortie du neurone

- $\frac{\partial net_{o1}}{\partial \omega_5} = h_1$

Cette valeur représente l'impact du poids  $\omega_5$  sur la somme pondérée du neurone

Pour des raisons de simplifications d'écriture, nous dirons que  $\frac{\partial E_{total}}{\partial out_{o1}} * \frac{\partial out_{o1}}{\partial net_{o1}} = \delta_{o1}$ . Ainsi, nous avons  $\frac{\partial E_{total}}{\partial \omega_5} = \delta_{o1} * h_1$ .

La mise à jour du poids est donc définie par la relation:  $w_i^+ = w_i - \eta * \delta_{o1} * h_1$  avec  $\eta$ , le pas d'apprentissage. On observe ainsi que la correction de poids d'un neurone dépend de son entrée.

Le pas d'apprentissage détermine l'importance de l'évolution d'un poids en faisant varier la valeur du correctif imposé au poids. Un pas faible ralentira la vitesse de convergence du neurone mais tendra, à terme, à proposer une meilleure convergence. Au contraire, un pas trop grand favorisera une converge rapide mais cette dernière sera *grossière*. Un compromis entre vitesse et performance doit être trouvée. Un pas mal choisi peut provoquer des phénomènes d'oscillation d'un poids qui ne peut atteindre l'optimum local à cause d'une variation trop importante de sa valeur.

L'exemple ci-dessus ne traite que de l'apprentissage d'un neurone mais se généralise à un réseau de neurone (perceptron multicouche) avec le même raisonnement. Nous n'approfondirons pas ce calcul car les notations s'alourdissent grandement et ne présentent pas de nouveauté notable.

**Important:** Un réseau de neurone converge vers un minimum local ! Différents réseaux (au niveau des poids) peuvent être obtenus à partir des mêmes données. **Rien ne garantit** que la solution proposée par le réseau soit la **meilleure** !

---

<sup>16</sup>La dérivé de la fonction sigmoïde  $\sigma$  en  $x$  est  $\sigma(x) \cdot (1 - \sigma(x))$

## 2.5 Descente du gradient et optimiseur

Dans la section 2.4.2, nous avons observé que la correction appliquée sur les poids est dépendante de l'erreur totale du réseau. Il est important de définir comment cette erreur est calculée. Pour cela, il y a trois approches complémentaires: la détermination du nombres de données d'apprentissage analysées pour une même itération, la détermination du pas d'apprentissage optimal et la méthode de calcul du gradient.

### 2.5.1 Approche par Batch, Minibatch et Stochastique

Le calcul du gradient se présente sous trois formes distinctes dépendant du nombre de données d'apprentissage utilisées pour déterminer une valeur de gradient à rétro-propager.

#### 2.5.1.1 Approche par Batch

L'approche par batch correspond à la version originale du calcul du gradient. Cette approche utilise l'intégralité des données d'apprentissage avant de faire une rétro-propagation (une mise à jour des poids). Ainsi, l'erreur du réseau correspond à la moyenne de l'erreur réalisée par les prédictions de l'intégralité des données d'apprentissage.

Cette approche tend à définir un gradient stable permettant de limiter les "convergences oscillantes" du réseau. Il permet donc, en théorie, de favoriser une convergence lissée. Néanmoins, elle tend à converger vers un minimum local moins performant du fait de la trop grande généralité du gradient<sup>17</sup>. De plus, il est nécessaire de calculer l'intégralité des erreurs avant de réaliser une mise à jour. Bien que ces calculs puissent être parallélisés, les mises à jour des poids sont lentes et de ce fait, l'apprentissage du réseau aussi. Dans le cas de jeu de données massifs (plusieurs millions voir milliards de données), cette approche est inutilisable en temps humain.

#### 2.5.1.2 Approche Stochastique

L'approche stochastique calcule l'erreur avec une donnée d'apprentissage uniquement. Ainsi, il y a un apprentissage par rétropropagation à chaque prédiction réalisée durant l'apprentissage.

Cette approche permet une évolution rapide du réseau du fait de la mise à jour récurrente des poids à chaque prédiction d'apprentissage. L'aspect stochastique favorise un gradient *bruité* qui limite le risque de convergence précipitée vers un

---

<sup>17</sup>Rappelez-vous le compromis biais-variance. Trop de généralité nuit à la spécialisation du modèle !

minimum local mais favorise aussi la convergence vers un minimum potentiellement moins performant. De plus, l'aspect bruité du gradient entraîne une forte variance dans la mise à jour des poids et de ce fait, tend à rendre l'évolution du réseau moins *lissée*. Cette approche est très sensible aux données aberrantes et extrêmes.

### 2.5.1.3 Approche par MiniBatch

L'approche par MiniBatch est un compromis entre l'approche par Batch et Stochastique. Avec cette méthode, l'erreur est calculé à partir d'un sous-ensemble du jeu d'apprentissage. Cette méthode est la plus répandue aujourd'hui du fait de ses performances.

Cette méthode permet une plus grande robustesse et un meilleur contrôle de l'évolution du réseau que l'approche stochastique tout en conservant une mise à jour du réseau "régulière", ce qui permet une convergence du modèle plus rapide. Le MiniBatch étant de taille restreinte, la problématique de convergence précipitée de l'approche par Batch est évitée. Cette méthode offre ainsi un bon compromis qui favorise les performances, de même que pour les facilités d'implémentation en limitant les problématiques de stockage des données d'apprentissage en mémoire. Néanmoins, le nombre d'entités dans le MiniBatch est un hyperparamètre important qui ne peut être défini qu'empiriquement.

### 2.5.2 Aperçu des optimiseurs

Lors de la mise à jour des poids, le pas (facteur multiplicatif du gradient) a une grande importance. Une des grandes difficultés est de déterminer qu'elle est sa valeur optimale et ce, de manière statique ou dynamique. Dans la version initiale de l'algorithme du gradient, le pas est un hyperparamètre constant et fixé par l'utilisateur. Cette méthode s'avère difficile à optimiser et peu efficace expérimentalement.

Pour corriger ce problème, des améliorations (appelées *optimizer*) ont été proposées. Nous ne rentrerons pas dans le détail mathématique de leur fonctionnement mais nous observerons l'idée générale derrière les différentes solutions proposées.

Deux facteurs d'optimisation existent: l'importance à associer à la mise à jour d'un poids et l'étude du *moment* des gradients. Le moment représente "l'engouement" que l'on a dans la mise à jour du gradient proposé. Supposons une succession de gradients dans la même direction<sup>18</sup> et de grande intensité, nous pouvons donc supposer être sur une voie de convergence favorable. De ce

---

<sup>18</sup>Imaginez une balle qui glisse le long d'une pente. L'énergie emmagasinée par la balle augmente avec la pente et diminue avec une montée ou un sol plat. Le moment peut être associé à l'énergie cinétique de cette balle

fait, il faut favoriser la mise à jour associée. Au contraire, si la valeur s'amoindrit ou que la direction varie, il faut limiter l'engouement de la mise à jour afin de ne pas "se perdre". Le moment permet ainsi de limiter les oscillations durant la convergence, d'avoir une mémoire sur son évolution et d'accélérer la convergence.

Les deux approches principales basées sur le moment sont:

- **Vanilla Momentum:** Cette méthode est simple. Elle est calculée en ajoutant à la valeur du gradient (pondéré par le pas), la valeur du gradient précédent pondéré par une constante qui représente l'importance associé au moment. Cette méthode nécessite la mise en place d'un hyperparamètre choisi par l'utilisateur.
- **Nesterov Momentum:** Cette méthode est une variante du Vanilla Momentum. Alors que l'approche Vanilla Momentum n'anticipe pas ses positions futures, l'approche par Nesterov cherche à approximer la valeur future du gradient afin d'orienter son moment. Cette approche possède une preuve théorique plus forte, est plus efficace dans le cadre d'optimisation convexe et semble donner de meilleurs résultats expérimentaux.

Les approches principales basées sur le pas sont:

- **Adagrad:** Cet optimizer modifie le pas afin qu'il devienne dynamique et dépendant du paramètre optimisé. Ainsi, un paramètre rarement optimisé/observé aura un pas important alors qu'un paramètre régulièrement mis à jour aura un pas minoré. Cette méthode nous émancipe de l'étude d'une valeur pour le pas, si ce n'est la détermination d'une valeur initiale (souvent placé à 0.01).

La faiblesse de cette méthode est associée au facteur uniquement dégressif de la valeur du pas. En effet, selon cet optimizer, il ne peut "que" diminuer. Ainsi, au fil des apprentissages, le pas diminuera jusqu'à devenir infinitésimal et figera le réseau. Dans le cas d'apprentissage très profond, cette limitation est très problématique car la convergence n'est pas atteinte avant le blocage du pas.

- **AdaDelta:** Cet optimizer est une variante de Adagrad qui corrige le problème de la minoration agressive du pas. Ainsi, au lieu de considérer l'intégralité des pas, cet optimizer se concentre sur une fenêtre des derniers gradients calculés. Il propose aussi une méthode pour supprimer l'initialisation du pas, émancipant l'utilisateur d'un hyperparamètre.
- **RMSprop:** Cet optimizer est équivalent à AdalDelta si ce n'est qu'il ne propose pas d'approche pour s'émanciper de l'initialisation du pas. L'étude de ces deux optimizers sort du cadre de cette introduction. Veuillez vous référer à la liste des lectures proposées si vous souhaitez approfondir cette problématique.

Des approches unissent l'étude du pas et du moment telles que:

- **Adam:** Cet optimizer peut être caractérisé par l'union de l'optimizer RMSprop et de la considération du moment du gradient. Nous n'approfondirons pas l'étude de cet optimizer. Il est, aujourd'hui, l'approche la plus répandue et la plus utilisée en initialisation "par défaut".
- **Nadam:** Cet optimizer est comparable à Adam mais il emploie Nesterov Momentum pour le calcul du moment.

La liste présentée est non-exhaustive et l'étude des optimizers es encore un sujet de recherche actuel. Il n'existe pas de hiérarchie de performance clairement établie. Seule l'approche empirique est exploitée aujourd'hui.

### 2.5.3 Gradient noise

Afin de favoriser la robustesse du réseau face à des stimulations ambiguës ou inhabituelles, il est intéressant de *bruiter*[4] la valeur du gradient durant l'apprentissage afin de renforcer le réseau. Cette approche consiste à ajouter un bruit gaussien (distribution gaussienne centrée en 0 et écart-type variable) à la valeur du gradient calculé à chaque rétropropagation. Ainsi, supposons  $g_{i,t}$ , le gradient obtenu pour le poids i à l'instant t, alors  $g_{i,t,gauss} = g_{i,t} + \mathcal{N}(0, \sigma^2)$  avec  $\sigma_t^2 = \frac{\eta}{(1+t)^\gamma}$  et  $\eta \in [0.01, 0.3, 1.0]$ .

Ainsi, le bruit sera plus important en début d'apprentissage pour forcer le réseau à ne pas converger trop rapidement vers un minimum local<sup>19</sup>. Cette méthode serait performante pour les modèles très profonds, limiterait l'impact d'une mauvaise initialisation des poids du réseau et favoriserait "l'échappement" des minimum locaux (qui sont de plus en plus nombreux alors que le modèle s'approfondit). Cette méthode, bien qu'élégante, n'est que peu employée par la recherche et son efficacité incertaine.

### 2.5.4 Gradient Clipping

Afin de limiter le risque d'explosion du gradient (phénomène nommé *exploding gradient*<sup>20</sup>), il est pertinent de borner la valeur du gradient en normalisant sa valeur absolue lorsqu'elle dépasse un seuil défini. Cette méthode est appelée *Gradient Clipping*[73]. Il existe plusieurs manières de réaliser cette normalisation bien que la majorité des implémentations se limitent à l'utilisation d'une valeur seuil, i.e  $|g_{i,t}| > \lambda \Rightarrow |g_{i,t}| = \lambda$ . Cette approche est reconnue comme efficace au fil des publications de recherche.

---

<sup>19</sup>Favoriser l'exploration des solutions possibles

<sup>20</sup>Voir Section 2.6.2

## 2.6 ReLu et les dangers du gradient

Le gradient est l'élément central pour l'apprentissage d'un réseau de neurones. Cependant, trois dangers majeurs sont à considérer pour garantir l'intégrité des gradients: **Exploding Gradient**, **Vanishing Gradient** et **Dead ReLu**.

Un réseau de neurone peut avoir des centaines (ou plus) couches associées à des milliers (ou plus) neurones. De ce fait, le modèle peut être très profond. Nous avons vu que l'apprentissage se base sur la rétropropagation du gradient. Cette rétropropagation est un produit de facteur. Ainsi, plus le poids à mettre à jour est éloignée de la sortie du réseau (couches basses du réseau), plus le degrés du produit est élevé. Cette particularité peut poser problème selon l'architecture du réseau (notamment la profondeur) ou les fonctions de transfert dont la dérivée est comprise dans  $[0, 1[$  ou strictement supérieure à 1 sur un intervalle quelconque.

### 2.6.1 Vanishing Gradient

Dans le cas où la valeur des dérivées partielles des couches intermédiaires est comprise dans  $[0, 1[$ , il y a un risque de *vanishing gradient*. En effet, pour  $n$  dans  $[0, 1[$ ,  $\prod_{k=1}^{\infty} \alpha_n \xrightarrow{\infty} 0$ . Cette particularité peut figer le réseau et annuler la mise à jour de neurones éloignés car la mise à jour du poids est infinitésimale.

### 2.6.2 Exploding Gradient

Au contraire, pour des valeurs comprises dans  $]1, \infty[$ , il y a un risque d'*exploding gradient*. Pour  $n$  dans  $]1, \infty[$ ,  $\prod_{k=1}^{\infty} \alpha_n \xrightarrow{\infty}$ . Ainsi, la valeur du gradient "explose" et peut empêcher une bonne convergence du modèle (évolution du poids trop importante) voire annuler l'apprentissage du réseau avec des valeurs dites NaN (Not A Number) car dépassant les limites matérielles de la représentation des nombres par ordinateur.

### 2.6.3 Fonction ReLu et Dead ReLu

Pour corriger cette difficulté, une fonction d'activation est souvent utilisée pour les couches cachées des réseaux: la fonction ReLu.

La fonction ReLu est définie par:  $f(x) = \begin{cases} 0 & \text{si } x < 0 \\ \max(0, x) & \text{si } x \geq 0 \end{cases}$

Sa dérivée est nulle sur  $]\infty, 0]$  et égale à 1 sur  $]0, \infty[$ . De part les valeurs de sa dérivée, le risque d'*exploding gradient* est annulé. Néanmoins, la présence d'une dérivée nulle peut laisser penser que le risque de *vanishing gradient* est présent. Bien que vrai, ce défaut est compensé par la capacité de ReLu à rendre le réseau éparse.

Le risque de sur-apprentissage<sup>21</sup> est présent lorsque les neurones deviennent trop inter-dépendants, que ce soit durant la prédiction ou l'apprentissage. L'idée principale, soutenue par l'analogie biologique du cerveau, est que le réseau doit être localement stimulé pour répondre à une entrée. Ainsi, selon l'entrée, l'ensemble du réseau ne doit pas être activé mais seulement une sous-partie capable d'interpréter cette stimulation. La fonction ReLU, nulle pour tout  $x$  négatif, permet de simuler ce comportement. De plus, la dérivée de ReLu, semblable à une fonction *Porte*, permet de réaliser des mises à jour éparses du réseau. La présence d'une dérivé nulle (pour  $x$  négatif) permet de limiter localement la mise à jour des poids, rendant les évolutions du système localisées et moins inter-dépendantes. En cas de stimulation positive, la dérivée étant de 1, l'influence sur la valeur du gradient est nulle. Cette fonction d'activation permet ainsi d'éteindre des neurones durant la phase de prédiction et force le réseau à avoir une stimulation localisée pour réaliser la prédiction. Durant la rétro-propagation, son comportement orchestre les régions du réseau qui apprennent et celles qui dorment. Ainsi, cette fonction permet d'agir sur l'architecture du réseau pour limiter les problèmes de corrélation néfastes entre les neurones qui favorisent le sur-apprentissage. De plus, cette limitation des neurones employés permet de limiter le coût de calcul et ainsi, d'augmenter la vitesse du réseau (en prédiction et apprentissage).

Ces spécificités ont popularisé la fonction ReLu qui est, aujourd'hui, la fonction de référence pour les couches cachées des réseaux neuronaux. Elle est peu employée pour la couche de sortie car elle ne présente que peu de pouvoir explicatif. Les fonction sigmoïde/softmax sont préférées pour leur représentation probabiliste ou encore la fonction linéaire standard pour une représentation sans norme spécifique.

Néanmoins, cette fonction est sensible au phénomène appelé *Dead ReLu*. En effet, la dérivée (localement) nulle de la fonction ReLu présente des caractéristiques dangereuses. En effet, dans le cas d'initialisation du réseau avec des poids mal calibrés, il est possible que la stimulation soit fortement négative (peu importe la données d'apprentissage), provocant une sortie nulle permanente du neurone. La dérivée étant nulle, le neurone n'apprend pas (gradient nul) et de ce fait, la sortie ne pourra jamais être autre que nulle durant tout l'apprentissage. Ce comportement est associé à un "neurone mort": le neurone n'apprend pas et retourne une sortie nulle en tout temps. Cette particularité peut entraîner l'inactivité d'une partie plus ou moins importante du réseau et nuire à son efficacité voire le condamner. Afin de lutter contre ce problème, l'utilisation de *Régularisation*<sup>22</sup> en plus d'une bonne initialisation<sup>23</sup>) sont employées.

Afin de lutter contre ce phénomène, des améliorations ont été proposées pour

---

<sup>21</sup>Voir Section 4.1

<sup>22</sup>Voir Section 4.3

<sup>23</sup>Voir Section 2.7

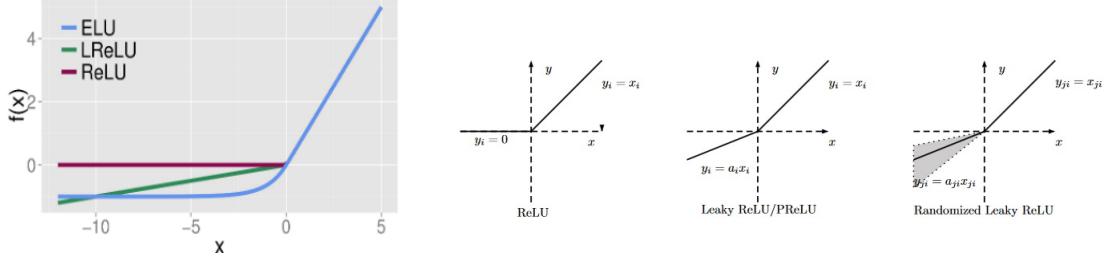


Figure 5: Fonction ReLu et ses Variantes

la fonction ReLu (un graphique récapitulatif est visible sur la Figure 5):

1. **Leaky ReLu(LReLU)[104]:**

La fonction est définie par:  $f(x) = \begin{cases} \alpha x & \text{si } x < 0 \\ \max(0, x) & \text{si } x \geq 0 \end{cases}$  avec  $\alpha$  petit (souvent initié à 0.01).

Cette variante permet de conserver la capacité d'apprentissage du neurone en supprimant la possibilité de gradient nul. Néanmoins, la valeur de la dérivée reste très faible (égale à  $\alpha$ ), ce qui peut demander un temps d'apprentissage important pour ré-activer le neurone. De plus, elle demande la détermination d'un nouveau hyperparamètre.

2. **Randomized ReLu(RReLU)[104]:**

Cette fonction est définie par:  $f(x) = \begin{cases} \alpha^{(\mathcal{U}_i)} x & \text{si } x < 0 \\ \max(0, x) & \text{si } x \geq 0 \end{cases}$  avec  $\alpha^{(\mathcal{U}_i)}$ , valeur aléatoire issue d'une distribution uniforme  $\mathcal{U}(m, n)$  avec  $m < n$  et  $[m, n] \in [0, 1[$ .

Lors de l'apprentissage,  $\alpha^{(\mathcal{U}_i)}$  varie à chaque itérations. Lors du l'évaluation (test) du modèle ou de prédiction, la valeur de  $\alpha^{(\mathcal{U}_i)}$  est fixé. L'objectif de l'aléatoire est de diminuer le risque de sur-apprentissage du réseau.

3. **Parameterized ReLu(PReLU)[104]:**

Cette fonction est semblable à Leaky ReLu mais le coefficient  $\alpha$  est calculé dynamiquement par backpropagation. Ainsi,  $\alpha$  n'est plus un hyperparamètre mais demande un coût de calcul supérieur (très négligeable).

4. **Exponential ReLU(ELU)[12]:**

La fonction est définie par:  $f(x) = \begin{cases} \alpha(\exp(x) - 1) & \text{si } x < 0 \\ \max(0, x) & \text{si } x \geq 0 \end{cases}$

Cette fonction permet de borner la valeur d'activation pour  $x < 0$ , augmentant ainsi sa résistance au bruit. Le facteur  $\alpha$  est un hyperparamètre fixé.

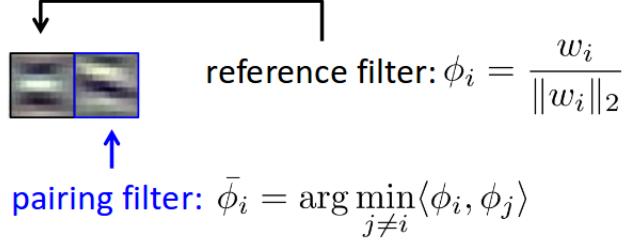


Figure 6: Exemple de deux filtres de phase opposée

##### 5. Concatenated ReLU(CReLU)[84]:

Expérimentalement, il a été montré que les filtres des premières couches d'un réseau tendent à être redondants afin d'extraire l'information issue des *phases* positives et négatives d'un signal donné. Cette faiblesse oblige le réseau à exploiter d'autres filtres pour exploiter l'information perdue par l'application d'une fonction d'activation ReLU (perte de l'information de la phase négative). Par exemple, sur la Figure 6, nous pouvons observer un couple de filtres. Le comportement des filtres est similaire mais opposé par la phase. Afin de limiter la redondance de filtre, Concatenated ReLU propose d'extraire l'information intégralement sans la perte imposée par ReLU en considérant aussi les valeurs négatives selon la même approche que ReLU possède avec les valeurs positives.

CRReLU est ainsi définie par:  $f(x) = (\max(0, x), \max(0, -x))$ .

Elle est très similaire à Absolute Value Rectification (AVR) mais au lieu d'additionner les deux sorties intermédiaires, CRReLU les concatène. De ce fait, la sortie de cette fonction d'activation est de profondeur 2. Cette fonction d'activation permet ainsi de limiter le nombre de filtres nécessaires en optimisant l'extraction d'informations d'un signal donné.

##### 6. Scaled Exponential Linear Units:

La fonction est définie par:  $f(x) = \lambda \begin{cases} \alpha(\exp(x) - 1) & \text{si } x < 0 \\ \max(0, x) & \text{si } x \geq 0 \end{cases}$

**Attention:** Cette partie nécessite la connaissance des fondamentaux d'architecture d'un réseau profond.

Cette fonction est une des composantes utilisée dans le cadre des Self-Normalizing Neural Networks[50] (SNN). Cette architecture propose une autre approche afin de résoudre la problématique de la normalisation des données dans les réseaux très profonds. Au lieu d'employer des méth-

odes de normalisation externes<sup>24</sup> appliquées à la sortie d'une fonction d'activation, la sortie de la fonction d'activation fournit des valeurs déjà normalisées. Pour que cette spécificité soit réalisée, il est nécessaire d'employer des fonctions SELU initialisées selon la distribution  $W \sim \mathcal{N}(0, \frac{1}{n_{in}})$ . Cette distribution est comparable aux distributions standards (MSRA/Xavier par exemple) mais la variance n'exploite pas le facteur 2 qui neutralise les effets de la fonction d'activation. Pour plus de détails, notamment mathématiques, veuillez vous référer à l'article associé [50]<sup>25</sup>.

## 2.7 Initialisation des poids

L'initialisation des poids d'un réseau est un critère important à considérer. Une mauvaise initialisation peut provoquer la divergence de réseau, notamment dans le cas de réseau profond. Il est donc important de réaliser une initialisation limitant ce danger. L'objectif de l'initialisation est de faire en sorte que les sorties des neurones aient approximativement la même variance, de même que pour les valeurs des gradients obtenus par rétropropagation. Plusieurs approches ont été popularisées ces dernières années:

**Basé sur la variance des sorties de neurones uniquement:**

- **Calibration de la variance:** L'initialisation avec calibration de la variance revient à choisir aléatoirement une valeur dans une distribution normale définie par:  $W \sim \mathcal{N}(0, \frac{1}{n_{in}})$  avec  $n_{in}$ , nombre d'entrées du neurone (ou nombre de neurones sur la couche précédente).
- **ReLU Calibration:** La fonction ReLu est la fonction d'activation la plus populaire et utilisée actuellement (pour les couches cachées). Son étude est encore un sujet de recherche important et dynamique. Une publication récente [42] préconise une initialisation telle que:  $W \sim \mathcal{N}(0, \frac{2}{n_{in}})$  ou via une distribution uniforme:  $W \sim U\left[-\sqrt{\frac{6}{n_{in}}}, \sqrt{\frac{6}{n_{in}}}\right]$ .

La distribution normale est la distribution standard pour initier les poids associées à la fonction ReLu.

**Basé sur un compromis entre la variance des sorties de neurones et des gradients:**

- **Xavier initialization:** D'autres approches préconisent un compromis entre la variance des sorties des neurones et des gradients calculés par rétropropagation. La méthode préconisée devient ainsi une dépendante du nombre d'entrées et de sorties d'un neurone<sup>26</sup>. La première approche

---

<sup>24</sup>Comme le Batch Normalisation par exemple

<sup>25</sup>C'est un article très mathématique et lourd à la lecture !

<sup>26</sup>Lors de la rétropropagation, ce sont les sorties des neurones qui sont exploitées

correspond à une distribution normale:  $W \sim \mathcal{N}\left(0, \frac{2}{n_{in}+n_{out}}\right)$  et la seconde, une distribution uniforme:  $\mathbf{W} \sim U\left[-\frac{\sqrt{6}}{\sqrt{n_{in}+n_{out}}}, \frac{\sqrt{6}}{\sqrt{n_{in}+n_{out}}}\right]$ .

**Important:** Par défaut, il est **important** d'éviter les initialisations aléatoires sur un intervalle élevé, de même qu'initialiser à 0 l'ensemble des poids (sauf spécificité d'une méthode).

## 2.8 Jeu d'apprentissage et spécificités

Une des plus grandes contraintes du Deep Learning est la création d'un jeu de données d'apprentissage<sup>27</sup> de *qualité*. La notion de *qualité* repose sur différents critères:

- **Spécialisation:** Un jeu de données doit se focaliser sur le phénomène qu'il représente. Par exemple, si l'on souhaite détecter des chats sur une image, il est évident que le jeu de données d'apprentissage devra contenir des chats...
- **Représentativité:** Un jeu de données doit être capable de représenter le phénomène sous toutes ses formes (tout du moins un maximum) afin de rendre la discrimination représentative de ses différentes hypothèses possibles. Supposons un jeu de données pour discriminer tout type de chats. Il est donc intéressant d'avoir des données variables selon:
  - **Spécificité du phénomène:** La première condition est de représenter un maximum de spécificité que peut posséder le phénomène. Par exemple, dans le cas d'un chat, il est intéressant de mêler des chats de différentes races, tailles, couleurs, posture et mouvement etc...
  - **Spécificité du contexte:** Le phénomène ne sera pas toujours représenté de manière centrée sur l'image tout en occupant la majorité de sa surface. C'est ainsi nécessaire de considérer des cas où le phénomène est localisé sur une sous-partie de l'image. Il est donc important d'avoir des images de l'environnement où se développe le contexte. Par exemple, en supposant le cas des chats, il est utile d'avoir des cas où la présence du chat est plus (ou moins) importante<sup>28</sup>, une diversité d'environnement (un chat dans un appartement, dans la rue,...)
  - **Spécificité du capteur:** Selon le capteur, la nature de l'image peut varier indépendamment du phénomène et de son environnement. Un appareil haute définition ne donnera pas la même qualité d'image qu'une caméra standard. Il est donc nécessaire de considérer cette spécificité en exploitant des images issues de sources différentes.
  - **Spécificité de détérioration:** Il est possible qu'une image soit détériorée (capteur défectueux, présence de bruit, etc...). Exploiter

---

<sup>27</sup>On utilise souvent l'anglicisme *dataset*

<sup>28</sup>La localisation des zones devrait être uniformément (idéalement) distribuée

des données bruitées permet donc de renforcer la robustesse du modèle.

- **Biais et Indépendance:** Cet aspect est un sujet de recherche intense et un des piliers de l'*Ethique* de l'Intelligence Artificielle. Afin d'apprendre, un modèle (tout comme un être humain) devra posséder des biais de décision. Comprendre parfaitement et prévenir des dérives prédictives (comme les *prédictions auto-génératrices*) nécessitera la maîtrise et la détection de ces biais. Il est ainsi pertinent d'étudier les biais que possède notre jeu d'apprentissage (en considérant les particularités des méthodes d'apprentissage de l'algorithme utilisé) en les détectant dans un premier cas et si possible, les supprimer (ou modifier) si ils représentent un danger. Réaliser cette analyse est encore un travail immature et non abouti mais **capital** à moyen-terme.

## 2.9 Prédiction multi-label et multi-classe

### 2.9.1 Généralités

La caractérisation *multi-classe / multi-label* est généralement réalisée dans le cadre d'une tâche de classification.

Une classification est caractérisée comme *mono-classe* lorsqu'elle ne discrimine qu'une classe. Il s'agit donc d'une classification binaire (Est / N'est pas). Au contraire, dans le cas d'une classification *multi-classe*, le réseau doit discriminer plusieurs classes afin de prédire les caractéristiques de l'entité inférée.

L'architecture du réseau est dépendante du type de classification souhaitée. En effet, dans le cadre d'une classification binaire, un seul neurone de sortie est nécessaire. Ce neurone aura pour fonction de calculer  $P(Y_{i,\text{classe}}|X_i, \theta)$ , i.e la probabilité que l'entité  $i$  soit de la classe  $Y_{\text{classe}}$  sachant ses caractéristiques  $X$  et l'architecture du réseau définie par  $\theta$ .

Au contraire, dans le cadre d'une tâche multi-classe, le réseau doit prédire  $n$  probabilités associées aux  $n$  classes définies. De ce fait, le réseau doit être capable de prédire  $P(Y_{i,\text{classe}_j}|X_i, \theta)$   
 $j \in [1, n]$

Un neurone est capable de prédire la probabilité associée à une classe. Pour prédire  $n$  probabilités associées à  $n$  classes, il est donc nécessaire d'avoir  $n$  neurones sur la couche de sortie.

On parle de classification *multi-classe* lorsque le modèle discrimine plusieurs classes mais qu'une entité ne peut être associée qu'à **une** classe uniquement, i.e les classes sont **mutuellement exclusives**. Par exemple, supposons un modèle de classification multi-classe capable de discriminer les chats et les chiens, les entités inférées par ce modèle seront classées comme **chat ou comme**

chien uniquement. Or, le modèle théorique présenté précédemment calcule  $P(Y_{i,\text{classe}_j}|X_i, \theta)$ . Pour respecter le cadre de la classification multi-tâche, le réseau doit donc être modifié pour calculer  $\underset{j \in [1,n]}{\operatorname{argmax}}(P(Y_{i,\text{classe}_j}|X_i, \theta))$ .

Au contraire, on parle de classification multi-label lorsqu'une entité peut être définie par **plusieurs classes**. Elles ne sont donc pas exclusives. Par exemple, un homme à vélo peut être catégorisé comme *homme* et *cycliste*. De ce fait, la classification multi-label, au contraire de la classification multi-classe, n'impose pas l'utilisation de *argmax*.

En résumé, la classification *multi-classe* prédit positivement une unique classe parmi un ensemble de classes alors que la classification *multi-label* en prédit positivement aucune, une (ou plusieurs) parmi un ensemble de classes.

La différence d'architecture entre ces deux approches repose essentiellement sur la fonction d'activation de la couche de sortie. Ainsi, dans le cadre d'une classification *multi-classe*, la couche de sortie sera associée à une activation par la fonction *softmax* qui permet d'extraire la probabilité la plus élevée en plus de respecter la condition d'exclusion. Néanmoins, cette fonction impose  $\sum_i P(Y_i|X) = 1$ . Cette hypothèse est plus *forte* que la condition d'exclusion. En effet, elle impose que l'*univers* corresponde à l'ensemble des classes du modèle, ce qui est une contrainte forte.

Au contraire, dans le cadre d'une classification multi-label, chaque neurone de la couche de sortie sera associé à la fonction *sigmoïde* qui permet de calculer la probabilité d'appartenance à une classe indépendamment pour chaque classe. Chaque classe étant indépendante des autres, il n'y a pas d'impératif de somme égale à 1.

**Remarque:** La fonction *softmax* extrait la probabilité la plus forte proportionnellement aux valeurs de l'ensemble de probabilités obtenues<sup>29</sup>. De ce fait, une classe est nécessairement prédite *positivement* bien qu'il soit possible que l'entité analysée ne corresponde pas à une des classes du réseau. Par exemple, supposons un réseau capable de discriminer les cyclistes et les chats. Si on présente un chien, le réseau déterminera des probabilités faibles pour les classes cyclistes et chats. Néanmoins, la fonction *softmax* prédira nécessairement des probabilités indépendamment de la valeur "absolue" de ces prédictions. La classe prédite sera probablement chat car un chat "ressemble plus" à un chien qu'à un cycliste. Comme un chien ressemble bien plus à un chat qu'à un cycliste, *softmax* prédira une probabilité élevée pour la classe *chat*. Cette particularité impose de créer une classe *neutre* qui correspond à une entité non caractérisée par les autres classes. Ainsi, si un modèle discrimine n classes, dans les faits, il

---

<sup>29</sup>L'utilisation de *softmax* impose l'hypothèse que chaque entité puisse être caractérisée par une classe du modèle.

devra en discriminer  $n+1$  pour considérer la possibilité d'une image non associée à une de ces classes.

Au contraire, la classification *multi-label* repose sur la valeur absolue des probabilités et non la valeur relative entre ces probabilités. Chaque probabilité étant indépendante, il est tout à fait possible qu'une prédiction de ce type de réseau soit négative pour chacune des classes. Ainsi, il n'y a pas d'impératif à la création d'une classe *neutre*.

Bien que l'approche *multi-label* soit plus souple et modulable, elle est plus dure à exploiter dans le cadre de l'apprentissage. L'approche *multi-classe* est souvent préférée lorsque le problème traité le permet.

### 2.9.2 Prédiction et distribution de données

La difficulté principale de la prédiction *multi-classe* (ou *multi-label*) est associée à la distribution des données d'apprentissage. En effet, il est possible qu'elle soit très déséquilibrée avec, par exemple, une classe fortement majoritaire et d'autres minoritaires. Cette particularité du jeu d'apprentissage induira un biais dans l'apprentissage qui peut être responsable d'un échec critique de l'apprentissage. Ce type de problématique est très répandu, notamment dans les tâches de détection d'entités rares qui sont très présentes dans le domaine médicale par exemple.

Supposons une tâche qui consiste à détecter les tweets *toxiques* et à les classifier selon différentes catégories (acharnement, contenu sexuel, discrimination raciale). Nous supposerons qu'un tweet peut appartenir à plusieurs catégories. Nous sommes donc dans le cadre d'une prédiction *multi-label*.

Notre jeu d'apprentissage correspond à un ensemble de tweets obtenus sur un intervalle continu et dont les tweets sont labellisés sans considération de leurs particularités (sain ou toxique). Il est évident que la majorité des tweets sont "sains" et de ce fait, non classifiés dans les sous catégories de toxicité. Ainsi, la majorité des tweets d'apprentissage sont négatifs pour chacune de ces classes, i.e présentent un vecteur de label égal à  $[0, 0, 0]$ . Nous supposerons les sous-catégories comme uniformément distribuées et la répartition sain/toxique équivalente à 95%-5%.

Le risque principal induit par ce type de données déséquilibrées est la (possible) convergence vers un minimum local sans pouvoir explicatif. En effet, nous avons 95% des données classées comme saine. Ainsi, si le modèle considère que toute donnée est saine, alors il aura 95% de bonne prédiction sur son jeu d'apprentissage. Les données d'apprentissage associées à un tweet toxique étant rares, leurs impacts sur la fonction de perte sont *noyés* dans l'ensemble des "bonnes prédictions", ce qui diminuera grandement leur pouvoir d'apprentissage. Ce phénomène est classique et particulièrement critique dans

le cadre de l'apprentissage machine en général<sup>30</sup>.

### 2.9.3 Méthodes d'apprentissage

Afin de permettre un meilleur apprentissage, différentes approches sont envisageables. Elles sont appliquées au niveau des données d'apprentissage ou au niveau du modèle entraîné. Ces méthodes favorisent le phénomène de sur-apprentissage (voir Section 4.1). Elles doivent donc être utilisées avec grande attention.

#### 2.9.3.1 Au niveau des données

Les principales approches au niveau des données reposent sur des méthodes d'échantillonnage. Elles sont divisées en deux groupes:

- **Undersampling:** Afin d'uniformiser les distributions, ce type d'approche propose de supprimer aléatoirement des données de la classe majoritaire. Bien que simple d'utilisation, cette méthode est *destructrice* et peut conduire à la perte de données au pouvoir explicatif important.

Afin de lutter contre ce risque, des améliorations ont été proposées afin de permettre la sélection de données au faible pouvoir explicatif. Elles analysent les données pour détecter les caractéristiques redondantes et se focaliser sur la suppression des *doublons*. Des approches de *Data Decontamination* sont envisageables aussi.

- **Oversampling:** Cette méthode *augmente* les données dont la classe est sous-représentées. L'approche standard consiste à répliquer aléatoirement des données dont la classe est minoritaire afin d'atteindre une distribution uniforme des classes. Néanmoins, un risque élevé de sur-apprentissage est à considérer.

Pour améliorer le pouvoir explicatif des données créées, des améliorations ont été faites notamment via l'utilisation de données artificiellement créées. Pour cela, l'interpolation d'entités *voisines* est exploitée. Les méthodes standards reposent sur l'utilisation du clustering qui permet de considérer l'équilibre intra/inter-classe. De même, exploiter le *Boosting* permet d'isoler les exemples *difficiles* et de cibler l'augmentation des données sur des exemples à problèmes<sup>31</sup>. Une autre approche (propre au Deep-Learning) consiste à créer des *minibatch* dont la distribution des classes est garantie uniforme par la sélection aléatoire d'exemples issus de chacune des classes.

---

<sup>30</sup>Ce problème illustre la nécessité d'une étude préalable des données afin de détecter ce phénomène

<sup>31</sup>Il y a un risque important de sur-apprentissage avec cette approche.

### 2.9.3.2 Au niveau du modèle

D'autres méthodes s'appliquent au niveau de l'architecture du modèle, de sa méthode d'apprentissage et de son rapport avec les données utilisées.

- **Calibration:** La *Calibration* est une méthode pour ajuster la prédiction réalisée par le modèle afin de limiter le biais induit par la distribution des données uniformisée par les méthodes ci-dessus. L'approche standard consiste à compenser le biais prédictif par la considération de la probabilité *a priori*<sup>32</sup> de la classe prédite.

Il a été montré qu'un réseau neuronal estime la probabilité à posteriori d'une entité. Par conséquent, un réseau neuronal estime:

$$y_{classe}(x) = p(\text{classe}|x) = \frac{p(\text{classe}) * p(x|\text{classe})}{p(x)}$$

$p(\text{classe})$  est biaisé par la méthode d'échantillonnage qui ajuste les données d'apprentissage. Afin de corriger ce biais, il est nécessaire de considérer la distribution initiale des données. Ainsi nous obtenons:

$$y_{classe}(x_i)^{\text{calibrated}} = y_{classe}(x_i) * \frac{\text{Card}(x_{\text{classe}})}{\text{Card}(x)}$$

- **Pondération des données d'apprentissage:** Afin de considérer le déséquilibre des distributions, il est possible de modifier l'importance associée à une donnée selon sa classe. L'approche traditionnelle repose sur une pondération en fonction de la fréquence de classe. Ainsi, une données appartenant à une classe fréquente aura une influence plus faible afin de compenser la présence élevée de ce type de données dans le minibatch. Au contraire, une donnée appartenant à une classe rare aura une erreur majorée afin de compenser la faible représentation de cette classe. Ainsi, l'erreur associée à une donnée est définie par:

$$Er_i^{\text{ponderated}} = Er_i * \frac{\text{Card}(x)}{\text{Card}(x_{\text{classe}})}$$

Au lieu de modifier la méthode de calcul de l'erreur, il est possible d'influencer la valeur du pas d'apprentissage en fonction de la données d'apprentissage. Ainsi, dans le cadre d'un apprentissage par gradient stochastique, la valeur du pas sera pondérée selon la classe de la donnée traitée. Le pas serait minoré si la donnée appartient à une classe très représentée et majoré dans le cas contraire. Il est possible de généraliser cette solution pour la rendre exploitable avec un minibatch de données.

Cette calibration est rudimentaire et simpliste. Une approche reposant sur une régression logistique[48] permet une prédiction des poids plus approfondie.

---

<sup>32</sup>Des bases sur l'inférence Bayésienne sont nécessaires.

Si vous souhaitez approfondir vos connaissances sur cette problématique, veuillez vous référer à l'article [7] qui résume les principales méthodes modernes tout en proposant un recueil bibliographique important pour la poursuite de vos recherches. De même, l'article [77] propose une approche de pondération des données basée sur le comportement du gradient durant l'apprentissage. Cette approche est généralisable à tout type d'architecture tout en présentant une efficacité notable. Il s'agit sans doute d'une des approches à l'état de l'art pour ce type de problème.

## 2.10 Calcul matriciel et neurones

Un réseau de neurones est une structure qui nécessite beaucoup de ressources. Optimiser le temps de calculs est une nécessité absolue. Pour cela, on exploite les capacités du calcul matriciel qui permet l'exploitation de données à grande échelle<sup>33</sup>. Nous supposerons comme acquis les fondamentaux du calcul matriciel, notamment la multiplication.

Nous avons vu qu'un neurone est défini par une somme de ses entrées pondérées par ses poids (additionnée par la suite avec le biais). Cet ensemble forme un *logit* et ce dernier va être utilisé par la fonction d'activation du neurone pour déterminer la sortie du neurone. La problématique du calcul se situe donc majoritairement dans le calcul du *logit*. Pour cela, une représentation matricielle est possible.

Supposons un vecteur de données dans  $R^{784}$ . Il peut être représenté par la matrice:

$$X = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{784} \end{bmatrix}$$

Supposons un vecteur de poids d'un neurone  $W = [w_1, w_1, w_2, \dots, w_n]$ . Nous souhaitons multiplier terme à terme l'entrée avec son poids associé. On observe donc que ce comportement est réalisable par une multiplication matricielle en considérant  $W^T$ .<sup>34</sup> De plus, dans un neurone, le nombre d'entrée coïncide avec le nombre de poids. La concordance des dimensions est donc respectée. On obtient donc:

$$W = \begin{bmatrix} w_{1,1} \\ w_{2,1} \\ \vdots \\ w_{784,1} \end{bmatrix} \longrightarrow W^T = [w_{1,1} \quad w_{1,2} \quad \dots \quad w_{1,784}]$$

---

<sup>33</sup>En plus d'être facilement réalisé par un GPU

<sup>34</sup>On exploite la transposé de W pour avoir un vecteur colonne et non ligne afin de permettre le produit matriciel.

Ainsi, le calcul du logit d'un neurone est égal à:

$$\text{logit} = [w_{1,1} \quad w_{1,2} \quad \dots \quad w_{1,784}] * \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{784} \end{bmatrix}$$

Nous avons vu comment représenter le fonctionnement d'un neurone mais un réseau de neurones possède (l'écrasante majorité du temps), plus d'un neurone par couche. Afin de représenter cette spécificité, l'astuce consiste à représenter l'ensemble des poids de l'ensemble des neurones de la couche à travers une même matrice. Ainsi, pour une couche de 10 neurones avec 784 entrées, nous considérerons une matrice de dimension  $784 * 10$ , i.e, une colonne représente les poids d'un même neurones. De même, supposons un minibatch de 5 données, nous aurons donc une matrice de dimension  $784*5$ . Soit  $X$ , la matrice représentant le minibatch et  $W$ , matrice de poids des différents neurones:

$$X = \begin{bmatrix} x_{1,1} & \cdots & x_{1,5} \\ x_{2,1} & \cdots & x_{2,5} \\ \vdots & \ddots & \vdots \\ x_{784,1} & \cdots & x_{784,5} \end{bmatrix}, \quad W = \begin{bmatrix} w_{1,1} & \cdots & w_{1,10} \\ w_{2,1} & \cdots & w_{2,10} \\ \vdots & \ddots & \vdots \\ w_{784,1} & \cdots & w_{784,10} \end{bmatrix}$$

Afin de calculer les logits, nous calculons  $W^T X$  soit le produit matriciel d'une matrice  $10*784$  et  $784*5$ . Nous obtenons donc une matrice de sortie de dimensions  $10*5$ . Une colonne de la matrice de sortie représente ainsi les sorties des différents neurones pour une même données et une ligne, les sorties d'un même neurone pour différentes données.

Cette méthode est appliquée pour tout réseau et tout type de donnée en entrée. C'est pourquoi il est nécessaire de considérer exclusivement des données numériques. L'exploitation de données non numériques telles que du texte ou des variables catégorielles demandent une étape de pré-traitement pour les rendre exploitables. C'est notamment l'objectif des *Words Embedding* qui propose une projection vectorielle d'un mot, transformant une donnée textuelle en une donnée vectorisée exploitable par un réseau. Dans le cas de données catégorielles, une méthode standard est le *One Hot Encoding* où une variable catégorielle à  $n$  catégories est remplacée par un vecteur binaire dans  $R^n$  où l'ensemble des dimensions est à 0 exceptée la dimension associée à la catégorie initiale de la donnée<sup>35</sup>.

L'écriture matricielle d'un neurone est souvent exploitée dans la littérature. Ainsi, par exemple, supposons une couche de neurones activée par la fonction *softmax*. Une écriture standard de ce neurone serait:

$$Y = \text{softmax}(W^T X + b)$$

---

<sup>35</sup>Le vecteur aurait la forme  $[0, 0, \dots, 1, 0]$

Mois	Bénéfices de l'entreprise	Bénéfices d'un employé X
Janvier	$2 * 10^7$	$4 * 10^2$
Février	$8 * 10^7$	$2 * 10^2$
Mars	$1 * 10^6$	$1 * 10^3$
Avril	$5 * 10^8$	$3 * 10^1$
...	...	...

Figure 7: Exemple d'un jeu de données (toute ressemblance avec des données réelles est fortuite)

Ou encore:

$$Y_j = \frac{\exp(W_j^T X + b_j)}{\sum_i \exp(W_i^T X + b_i)}$$

L'écriture mathématique a tendance à "alourdir" la lecture des papiers de recherche. Bien que les notations puissent impressionner, l'intuition et la compréhension des idées développées sont souvent accessibles à un lecteur sans formation mathématique avancé.

### 3 Normalisation des données

Les données utilisées par un réseau de neurones peuvent être très variées et de nature différente. Ainsi, par exemple, dans le cas d'image, les données sont très similaires: matrice Hauteur\*Largeur à valeurs dans  $[0, 255]$  (la résolution de l'image - nombre de pixels de l'image - est supposée constante). Il est souvent préférable de standardiser ses données afin de permettre un apprentissage de qualité.

Supposons maintenant des données numériques et financières: le bénéfice d'une entreprise et le bénéfice d'un employé lambda. Les données sont visibles sur la Figure 7. Les valeurs des données de l'entreprise sont importantes (de l'ordre de grandeur  $[10^5, 10^7]$ ) et les données de l'employé plus faible (de l'ordre de grandeur  $[10^1, 10^3]$ ). Cette différence provoquera donc une valeur moyenne (exprimée par la moyenne d'un point de vue statistique) significativement différente entre ces deux sous-ensembles de données.

De même, la variation entre les données (exprimée par la variance d'un point de vue statistique) sera d'un autre ordre de grandeur. Par exemple, entre Mars et Avril, la variation du bénéfice de l'entreprise se compte en millions alors que la variation du bénéfice d'un employé se compte en centaines.

Revenons au contexte du réseau de neurones et supposons une problématique qui reposera sur notre jeu de données. Nous avons vu qu'un réseau apprend en minimisant une fonction de coût et qu'un poids d'un neurone est corrigé en dépendance avec sa valeur d'entrée. Il est donc évident que des valeurs à des

échelles différentes vont sur(sous)-stimuler le neurone. Ceci est très problématique car cette différence va provoquer une pondération des données d'entrées. Dans notre exemple, les données de l'entreprise seront sur-pondérées par rapport aux données de l'employé. Il est donc nécessaire de **normaliser** les données afin d'éviter ce genre de problème. Pour cela, différentes (parfois complémentaires) existent.

### 3.1 Centrer les données

Afin de limiter le problème d'échelle des données, il est possible de les *centrer*. Centrer les données modifient les données afin que la moyenne de ces données soit 0. Les nouvelles données sont obtenues selon la relation suivante:  $data_{i,centre} = data_{i,raw} - \mu$  où  $\mu$  est la moyenne de cet ensemble de données.

**Important:** La moyenne n'est pas réalisée sur l'ensemble du jeu de données mais sur les données associées à un même attribut. Dans notre exemple, il y aurait une moyenne pour les données de l'entreprise et une moyenne pour les données de l'employé.

Cette modification est réalisée par la quasi-totalité des pré-traitements de données et limite grandement le risque de biais dans l'apprentissage.

### 3.2 Réduire les données

Les données d'un jeu d'apprentissage peuvent avoir une variance importante, i.e des valeurs éloignées de la moyenne associée. Les conséquences sont similaires à celles provoquées par une échelle différente de la moyenne. L'idée est donc d'utiliser une échelle commune à l'ensemble du jeu de données en imposant un écart-type de 1. Ainsi, les données seront représentées par une valeur contenue dans  $[-1, 1]$ . Les nouvelles données sont obtenues selon la relation suivante:  $data_{i,reduit} = \frac{data_{i,raw}}{\sigma}$  où  $\sigma$  est l'écart-type de cet ensemble de données.

**Important:** L'écart-type n'est pas réalisé sur l'ensemble du jeu de données mais sur les données associées à un même attribut. Dans notre exemple, il y aurait un écart-type pour les données de l'entreprise et un écart-type pour les données de l'employé.

Dans les faits, cette modification est peu employée de manière isolée.

### 3.3 Centrer-Réduire les données

Centrer-Réduire les données revient à centrer et réduire les données, i.e réaliser les deux modifications explicitées précédemment. Ainsi, les nouvelles données sont définies telles que:  $data_{i,reduit} = \frac{data_{i,raw} - \mu}{\sigma}$  avec  $\mu$ , la moyenne et  $\sigma$ , l'écart-type de ce jeu de données.

Cette normalisation est la normalisation la plus utilisée dans le pré-traitement des données et efficace dans le traitement de la majorité des jeux de données. Elle est souvent appliquée en **traitement par défaut**. Néanmoins, bien que centrer les réduire soit très recommandé dans la majorité des cas, réduire les données dépend du cas à traiter. Par exemple, dans le cas de différents jeux d'images, l'écart-type relatif tend à être similaire entre eux. *Réduire* l'image tend donc à être peu utile.

**Important:** Il est **capital** de définir la moyenne et l'écart-type sur le jeu d'apprentissage uniquement. Par exemple, supposons deux jeux de données *train* et *test*. Nous apprendrons notre modèle avec *train* et nous l'évaluerons avec *test*. Afin de normaliser les données, il est nécessaire de pouvoir déterminer la moyenne et l'écart-type. Pour cela, il est indispensable de ne considérer que le jeu de données d'apprentissage. Ainsi, les paramètres  $\mu$  et  $\sigma$  seront déterminés avec *train* et, lors de l'évaluation du modèle, la normalisation utilisera les paramètres  $\mu$  et  $\sigma$  calculés avec *train*. Une erreur classique est de déterminer ces paramètres sur l'intégralité des données (*train* et *test*). Ceci est une **erreur grave** qui peut grandement nuire à l'évaluation du modèle.

### 3.4 Analyse en Composante Principale

Il est possible que les données à étudier soient de très grande dimension (plusieurs milliers d'attribut<sup>36</sup> voir plus). Analyser des données volumineuses impose une contrainte de temps (pour l'apprentissage et la prédiction - ce qui est problématique dans le cas du temps-réel). Il est souvent préférable de diminuer le nombre de dimension des données afin de limiter ces problèmes. Pour cela, une approche simple est efficace: *l'Analyse en composante principale* (ACP).

Dans un jeu de données, chaque attribut représente une dimension. Ainsi, un jeu de données avec  $n$  attributs sera représenté par un vecteur à  $n$  dimensions. L'ACP permet de considérer les corrélations entre les attributs d'un jeu de données afin de créer de nouvelles dimensions. Une dimension créée par ACP permet donc d'expliquer l'information utile expliquée par plusieurs attributs et, de ce fait, de diminuer le nombre de dimensions<sup>37</sup>. Cette approche est mathématique. Nous ne détaillerons pas son fonctionnement dans cette introduction. Il faut juste noter l'importance de cette approche pour l'analyse de données très volumineuses et la capacité de projeter des données de dimension  $N$  dans une dimension  $M$  choisie par l'utilisateur (souvent 2 ou 3 pour de la visualisation à une centaine pour la conversion de données volumineuses).

**Important:** Cette approche est destructrice. En effet, la réduction de dimension impose une perte de données qui peut être importante ou faible selon la nature des données et l'importance de la réduction. Une analyse approfondie de

---

<sup>36</sup>Attribut correspond à une catégorie du jeu de données, pas à une donnée brute

<sup>37</sup>Plus les données sont corrélées, plus le nombre de dimension peut être faible sans perte d'information majeure

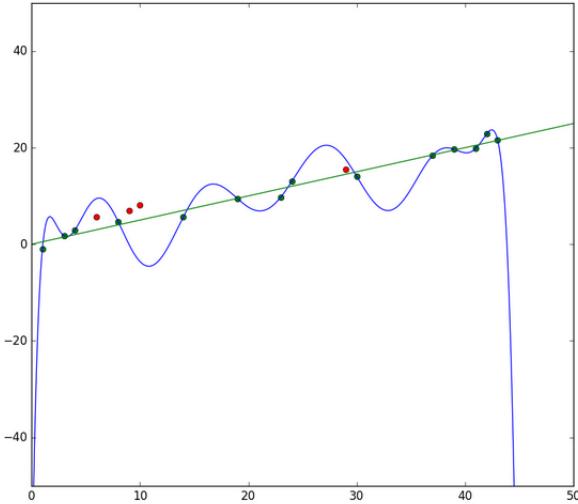


Figure 8: Exemple de sur-apprentissage (bleu) et d'apprentissage optimal (vert)

l'impact de la transformation est **capitale** pour limiter la perte d'information utile. Il est **intéressant** de noter que l'ACP est utilisable pour supprimer le bruit des données. En effet, un jeu de données peut présenter des valeurs aberrantes ou douteuses qui provoqueront un bruit dans les données et nueront à l'apprentissage du modèle. Diminuer le nombre de dimension permet donc de perdre l'information la moins représentative souvent caractéristique du bruit et de ce fait, de conserver des données généralisées<sup>38</sup>. Néanmoins, la détermination de présence de bruit ou non est souvent "tricky" et relève plus d'une sensibilité personnelle et d'un pari sur les données que d'une action mathématiquement démontrable. D'un point de vue métier, il est standard de conserver au moins 80% de l'information utile. Au-delà, le risque de destruction est trop important et souvent néfaste.

## 4 Régularisation et sur-apprentissage

### 4.1 Le sur-apprentissage

Le **danger principal** de tout algorithme d'apprentissage automatique est le *sur-apprentissage*. Comme vu dans l'introduction de cette partie, un algorithme d'apprentissage automatique cherche à approximer une fonction capable d'expliquer les données **et** capable de généralisation. Mais que signifie "généralisation" ?

---

<sup>38</sup>Et potentiellement de meilleure qualité selon le cas

Pour expliquer cette problématique, observons la Figure 8. Nous pouvons observer des points verts représentant les données du jeu d'apprentissage et des points rouges, les données inconnues qui doivent être prédites par le modèle. La courbe bleue et la courbe verte présentent deux fonctions apprises par deux modèles. On peut apercevoir que la courbe bleue passe par l'ensemble des points verts alors que la courbe verte possède une moins bonne performance. On peut donc naïvement dire que la courbe bleue a mieux appris. C'est vrai dans l'absolu: la courbe bleue a mieux appris le jeu de données d'apprentissage mais en plus d'apprendre le comportement général des données, elle a appris son bruit, ce qui lui donne son comportement oscillant et "imprévisible". Le fait d'apprendre le bruit est appelé *sur-apprentissage* et est désastreux pour le modèle qui perd ainsi sa capacité de généralisation. Au contraire, la courbe verte n'apprend pas le bruit des données. Elle apprend moins les données d'apprentissage mais conserve sa capacité de généralisation. L'idéal est donc de trouver un compromis entre apprentissage du jeu de données et capacité de généralisation: ceci est appelé *compromis biais-variance*.

Une explication graphique est visible sur la Figure 9. Le biais représente l'erreur réalisée sur l'apprentissage des données. Ainsi, un biais très faible sous-entend un sur-apprentissage car le bruit aura été appris. Au contraire, la variance représente l'importance des variations de la fonction hypothèse approximée, i.e la sensibilité du modèle aux variations des données et donc au bruit. De ce fait, plus la variance est élevée, plus le modèle perd en généralisation et devient inutilisable. Et, comme observé sur la Figure 8, le sur-apprentissage favorise un comportement à forte variance (représenté par les oscillations). L'objectif est donc de réaliser un compromis: optimiser la diminution du biais et de la variance.

D'un point de vue expérimental, la notion de biais et de variance est représentée par l'erreur de prédiction réalisée sur l'ensemble d'apprentissage et de test. Ainsi, l'objectif est d'arrêter l'apprentissage lorsque la courbe de prédiction du jeu de test augmente de manière significative alors que la prédiction sur le jeu d'apprentissage tend à diminuer.

Bien sur, la problématique du sous-apprentissage existe aussi et représente un modèle trop généraliste qui n'explique pas suffisamment les données d'apprentissage qui discriminent ses prédictions. Un exemple est visible sur la Figure 10 où la fonction créée est bien trop "neutre". Il est évident qu'un modèle sous-entraîné possède des performances faibles.

## 4.2 Limiter le sur-apprentissage

**Important:** Cette partie est fondamentale et doit être comprise pour développer un réseau de qualité, notamment dans le cadre d'architecture très profonde que nous étudierons par la suite.

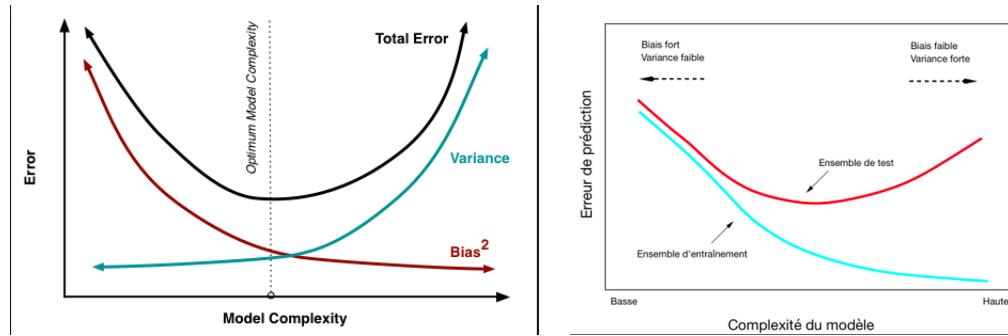


Figure 9: Compromis biais-variance et impact sur le modèle

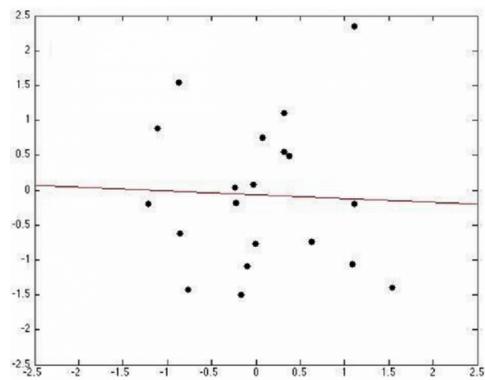


Figure 10: Exemple de sous-apprentissage

## 4.3 Régularisation

Les poids d'un neurone ne sont pas bornés<sup>39</sup>. Cette spécificité permet une grande disparité entre les valeurs que peuvent prendre les poids d'un neurone au risque de voir certains poids *explorer* alors que d'autres seraient *faibles*<sup>40</sup>. Cette différence de valeur est très préjudiciable et favorise un comportement de sur-apprentissage où un neurone sur-réagit aux stimulations portées par les poids de haute intensité. Pour limiter l'explosion des poids, une régulation peut être imposée.

Cette régulation peut être associée à la fonction de coût, lui rajoutant un nouveau critère de discrimination. Ainsi, supposons la fonction de coût définie par Mean Squared Error<sup>41</sup>. Elle est définie par:  $\mathcal{L} = \frac{1}{n} \sum_{i=1}^n (y^{(i)} - \hat{y}^{(i)})^2$ . Avec une régulation, nous obtenons  $\mathcal{L}_{reg} = \frac{1}{n} \sum_{i=1}^n (y^{(i)} - \hat{y}^{(i)})^2 + \epsilon(\omega)$  où  $\epsilon$  dépend uniquement des poids des neurones du réseau. Ainsi, la fonction de coût sera directement dépendant des poids des neurones et tendra à limiter leurs évolutions divergentes.

### 4.3.1 L1-Régulation

La première régulation est la L1-Régulation. Elle s'exprime sous la forme  $\lambda|\omega|$  où  $\lambda$  correspond à l'intensité associée à la régulation. Plus  $\lambda$  est élevé, plus on pondère l'importance d'avoir des poids faibles. Ainsi, dans le cas de Mean Squared Error, nous obtenons  $\mathcal{L}_{reg} = \frac{1}{n} \sum_{i=1}^n (y^{(i)} - \hat{y}^{(i)})^2 + \lambda|\omega|$ .

Cette régulation est très utile dans le cas d'extraction de *feature*. En effet, cette régulation tend à rendre le réseau *sparse* en permettant que des poids deviennent très proches de zéro (annulant ainsi une entrée du neurone). Cette régulation favorise les entrées discriminantes et limite les entrées porteuses de bruit. Il est ainsi possible d'extraire les composantes principales du réseaux. Néanmoins, dans les faits, elle est rarement utilisée car présente de moins bons résultats que la régulation L2 (sauf dans le cas spécifique d'extraction de feature). De plus, n'étant pas quadratique, sa minimisation est difficile.

### 4.3.2 L2-Régulation

L2-Regulation est définie par la contrainte  $\frac{1}{2}\lambda\omega^2$ . La fonction de coût régulée devient donc  $\mathcal{L}_{reg} = \frac{1}{n} \sum_{i=1}^n (y^{(i)} - \hat{y}^{(i)})^2 + \frac{1}{2}\lambda\omega^2$ .

Cette régulation favorise des valeurs de poids proches en limitant les pics de valeurs. Ainsi, elle permet de faire en sorte que le neurone ne se focalise pas sur

---

<sup>39</sup>Dans les cas d'une architecture standard

<sup>40</sup>Cette particularité est aggravée si les données en entrée du neurone ne sont pas normalisées. Les données d'apprentissage peuvent être normalisées en entrée tout en provoquant ce phénomène en interne.

<sup>41</sup>Voir la section 2.4.1 pour plus d'informations

des entrées dominantes tout en délaissant les autres jugées moins pertinentes. Elle limite donc la sur-spécialisation et de ce fait, l'overfitting. Cette régulation est efficace et très utilisée aujourd'hui.

#### 4.3.3 ElasticNet-Régulation

ElasticNet-Régulation est une combinaison linéaire de la Régulation L1 et L2. Elle s'exprime sous la forme  $\lambda_1|\omega| + \frac{1}{2}\lambda_2\omega^2$  où  $\lambda_1$  et  $\lambda_2$  définissent l'importance de cette régulation dans le calcul de la performance du modèle et la pondération entre les deux régulations (comportement équilibré ou pondération d'une des deux régulations). Ainsi, la fonction de coût est dorénavant définie par la relation suivant:  $\mathcal{L}_{reg} = \frac{1}{n} \sum_{i=1}^n (y^{(i)} - \hat{y}^{(i)})^2 + \lambda_1|\omega| + \frac{1}{2}\lambda_2\omega^2$

Cette régulation réunit *le meilleur des deux mondes* (issus de L1 et L2) bien que moins souvent appliquée que la L2-Régulation.

#### 4.4 Max norm constraints

Contrairement à la régulation L1 et L2, Max norm constraints ne s'applique pas à la fonction de coût mais directement au vecteur de poids du neurone. Alors que L1 et L2 pénalisent les poids élevés<sup>42</sup>, Max norm constraints agit directement sur ces derniers. Ainsi, chaque vecteur de poids doit respecter la condition  $\|\omega\|_2 < cnst$  ou *cnst* de petite taille (2,3 voire 4 par exemple). Si l'égalité n'est pas respectée, l'intégralité des poids est normalisée<sup>43</sup> afin de respecter la condition.

Cette régularisation est particulièrement performante associée avec le *DropOut*<sup>44</sup> mais présente la difficulté de paramétrage du seuil (*cnst*) qui est un hyper-paramètres. Sa détermination se fait de manière empirique et expérimentale, ce qui peut rendre sa recherche délicate.

#### 4.5 DropOut

Contrairement à la régulation qui agit sur les neurones de manière isolée, le DropOut[87] est une approche de régulation d'architecture, i.e une approche qui influence l'interaction entre neurones. Le DropOut sélectionne aléatoirement (selon une probabilité  $\rho$ ) des neurones du réseau et leur impose une activation<sup>45</sup> nulle. La sélection des neurones inactifs est renouvelée à chaque donnée d'apprentissage. Lors de l'utilisation du réseau hors apprentissage, chaque sorties des neurones concernées par le DropOut sont multipliées par  $\rho$  et toutes

---

<sup>42</sup>Pénaliser augmente la fonction de coût mais il n'y a pas de contrainte directe imposée à l'intégralité du réseau. Ainsi, L1/L2 favoriseront la limitation des poids les plus élevées en priorité alors que Max norm constraints les corrigera tous sans distinction

<sup>43</sup>Ce n'est pas la normalisation au sens statistiques !

<sup>44</sup>Voir <http://www.cs.toronto.edu/~rsalakhu/papers/srivastava14a.pdf>, partie 5.1

<sup>45</sup>L'activation correspond à la somme pondérée des entrées

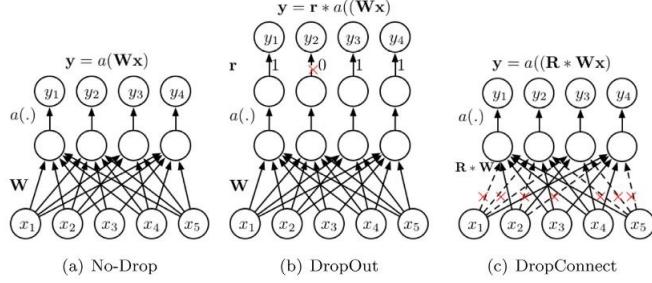


Figure 11: Comparaison d'un réseau sous DropOut et DropConnect

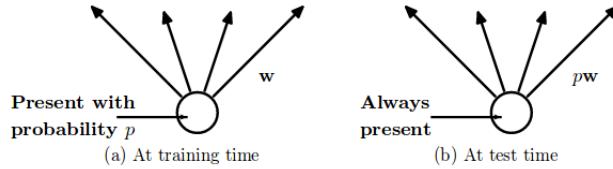


Figure 12: Relation des poids après DropOut

sont actives. Un exemple est visible sur la Figure 11.

Cette méthode permet de limiter l'inter-dépendance (Voir Section 2.6.3) entre les neurones et favorise la création d'un réseau éparse. L'apprentissage du modèle est ainsi plus robuste, plus rapide et moins soumis aux problématiques associées au gradient. Le DropOut est souvent exploité en plus d'une *Régulation* des poids bien que son comportement soit comparable à la régulation L2. Une valeur standard pour  $\rho$  est 0.5. Cette approche est souvent employée pour les réseaux Full-Connected mais peut être appliquée à d'autres architecture notamment les réseaux convolutifs. Néanmoins, l'efficacité de son utilisation sur des couches convolutives est sujet à débat.

**Important:** DropOut est souvent utilisé avec des fonctions d'activation  $f$  tel  $f(0)=0$ . En effet, le DropOut ne force pas la sortie à 0 mais son activation. Ainsi, si une fonction d'activation n'est pas nulle en 0, le neurone ne sera pas (complètement) inactif bien que son comportement soit constant. De plus, le biais est indépendant de l'activation et n'est pas soumis au DropOut.

## 4.6 DropConnect

DropConnect[98] est une généralisation de DropOut. Au lieu de supprimer l'activité d'un neurone en bloquant son activation, cette méthode annule les poids d'entrée du neurone de manière indépendante selon une probabilité  $\rho$ . Ainsi, un neurone peut voir son activation éteinte (comparable à DropOut) ou

juste partiellement. Un exemple de DropConnect est visible sur la Figure 11. Cette méthode est moins utilisée que DropOut et ses résultats plus rares. Il n'est pas ais  de d terminer quelle approche est la plus performante objectivement mais le DropConnect offre plus de souplesse et de configurations possibles.

## 4.7 Dense-Sparse-Dense training

Le Dense-Sparse-Dense[27] training est une m thode comparable   une variante du DropOut/DropConnect. Cette m thode cherche   proposer une m thode d'apprentissage plus efficace dans le cadre de r seaux tr s profonds. L'id e soutenue par cette m thode est qu'un r seau tr s profond est capable d'acqu rir une compr hension plus profonde d'un ph n m ne mais plus enclin   en apprendre le bruit et le comportement instable. Leur postulat est que le bruit est port  par les poids faibles des neurones. Ainsi, l'objectif est d' tre capable d'isoler les poids d'importance qui propagent une information fiable et les poids qui propagent le bruit. Pour cela, cette m thode (Figure 13) se d coupe en 3 phases:

1. **Initial Dense Phase:** L'apprentissage du r seau est fait normalement sans contrainte particuli re. Les m thodes de r gulations (et autres) peuvent  tre exploit es durant cette phase. L'objectif de cette phase n'est pas "d'apprendre" mais de d tecter les poids influents et non influents (en fonction de leurs valeurs absolues).
2. **Sparse Phase:** Durant cette phase, les poids sont class es par couche selon leurs valeurs. Seuls les  $\lambda\%$  plus lev s sont conserv s intacts, les autres devenant nuls.  $\lambda$  est un hyperparam tre   d閞miner (il est conseill  de prendre une valeur entre 25% et 50% comme valeur par d閞faut). Le r seau devient alors parse, permettant d'obtenir un r seau plus robuste et plus conomie. Il est empiriquement montr  qu'un r seau parse tend   avoir des r sultats quivalents voir meilleurs qu'un r seau dense car ce dernier est plus sensible aux probl matiques d'apprentissage. Un apprentissage du r seau est r alis  avec cette architecture.
3. **Final Dense Phase:** Durant cette phase, les connexions coup es sont restaur es. Les poids restaur s sont initi s   0 et le pas d'apprentissage du gradient faible (1/10 du pas initial). En effet, le r seau actuel pr sente une stabilit  et une convergence correcte. Il n'est donc pas souhaitable de "bouleverser" son architecture mais juste de favoriser sa performance en finalisant son comportement dans ses d tails d'apprentissage. Le r seau peut ainsi converger vers un meilleur minimum local voire "en sortir" vers un autre selon le cas.
4. **R p tition:** Ce cycle (3 phases) peut  tre r p t  afin de favoriser une meilleure convergence.

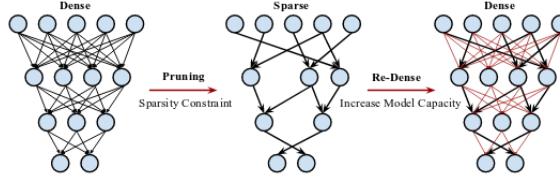


Figure 13: Dense-Sparse-Dense Routine

## 4.8 Batch Normalization

Batch Normalization[39] est une méthode qui vise à augmenter la robustesse du modèle en favorisant sa capacité de généralisation. Supposons un jeu de données d'image de chiens blancs, si une image d'un chien noir apparaît lors de la prédiction, il est évident que le réseau ne fonctionnera pas car la distribution des données d'apprentissage et de prédiction ne correspondent pas. Néanmoins, le comportement des données peut être similaire et donc, la fonction obtenue sur les chiens blancs pourraient être effective. Batch Normalization permet d'aligner les distributions et de limiter cette problématique. Un exemple illustratif est visible sur la Figure 14. La justification de performance de cette approche est très mathématique et repose sur un socle solide de Statistiques. Nous ne l'étudierons pas dans cette introduction.

L'idée de Batch Normalization est de généraliser la normalisation des données à travers le réseau de neurones. La méthode est comparable à une normalisation standard mais présente des spécificités associées à la présence d'un minibatch qui représente un sous-ensemble du jeu d'apprentissage (qui plus est, variable). La normalisation réalisée est visible sur la Figure 15. Veuillez vous référer à la publication associée[39] pour plus de détails mathématiques sur son fonctionnement.

De plus, cette méthode permet de mieux isoler les couches entre elles, limitant l'inter-dépendance, permet d'utiliser un pas d'apprentissage plus important car les données internes au réseau sont normalisées (ce qui limite les valeurs extrêmes et donc les valeurs du gradient) et possède une action de régulation. L'utiliser avec DropOut (ou équivalent) est efficace bien qu'il soit préférable d'utiliser une probabilité plus faible d'éteindre un neurone dans cette configuration. Un gain de vitesse est aussi observé du fait de la limitation des valeurs des données, ce qui favorise une implémentation optimisée.

De même que le DropOut, Batch Normalization fait partie des améliorations majeures proposées dans le cadre du développement d'un réseau de neurones et son utilisation est quasi-récurrente à tout réseau d'ampleur.

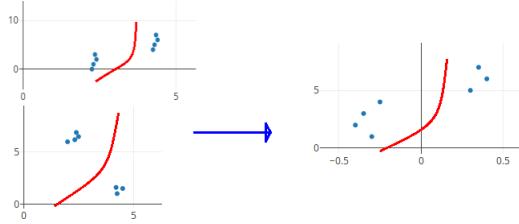


Figure 14: Non-alignement des distributions

**Input:** Values of  $x$  over a mini-batch:  $\mathcal{B} = \{x_{1\dots m}\}$ ;  
 Parameters to be learned:  $\gamma, \beta$   
**Output:**  $\{y_i = BN_{\gamma, \beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m} \sum_{i=1}^m x_i \quad // \text{mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m} \sum_{i=1}^m (x_i - \mu_{\mathcal{B}})^2 \quad // \text{mini-batch variance}$$

$$\hat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \quad // \text{normalize}$$

$$y_i \leftarrow \gamma \hat{x}_i + \beta \equiv BN_{\gamma, \beta}(x_i) \quad // \text{scale and shift}$$

Figure 15: Normalisation par Batch Normalization

#### 4.9 Critère d'arrêt de l'apprentissage

Pour réaliser un bon apprentissage, il est nécessaire de considérer le  *compromis biais-variance* afin de limiter le sur-apprentissage. Bien que la problématique soit facilement assimilable, s'en prémunir reste difficile.

Une bonne pratique repose sur l'utilisation d'architectures de régulation qui tend à limiter le phénomène de sur-apprentissage<sup>46</sup>. Bien que fonctionnelles, ces architectures ne garantissent pas un apprentissage contrôlé. Une approche complémentaire est donc de savoir **quand** arrêter l'apprentissage afin d'obtenir le meilleur état du modèle. Dans le cas idéal, l'arrêt se situe à l'intersection de la courbe de biais et de variance. Mais comment savoir si on se situe dans un intervalle proche de ce point de référence ?

La courbe de biais est associée aux erreurs de prédiction sur les données d'apprentissage alors que pour la courbe de variance, ce sont les erreurs de prédiction sur les données de validation (Figure 9). La courbe de biais décroît logiquement durant l'apprentissage jusqu'à atteindre une valeur asymptotique caractérisée par un résidu de bruit. Au contraire, la courbe de variance va décroître progressivement avant de croître lorsque le modèle commence à perdre sa capacité de généralisation. L'objectif étant d'assurer la meilleure performance prédictive tout en ayant une forte capacité d'abstraction, se concentrer sur le comportement de la

<sup>46</sup>Phénomène présent lorsque la courbe de biais diminue et la courbe de variance augmente

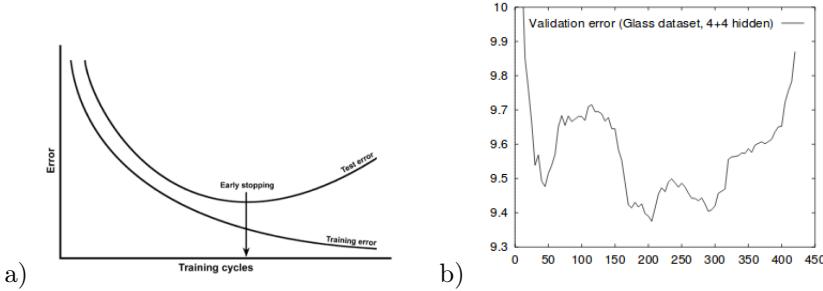


Figure 16: a) Early Stop théorique b) Comportement-type de l'erreur de validation en situation réelle

courbe de variance semble prioritaire. Il est ainsi nécessaire de détecter le minimum global de cette fonction alors que la courbe de biais ne présente qu'une importance secondaire. Il serait même dangereux de se concentrer sur la courbe d'apprentissage car une erreur très faible d'apprentissage est souvent associée à un sur-apprentissage important.

Sur la Figure 9, la courbe de variance est idéalisée. Dans les faits, elle présente souvent du bruits et des oscillations locales qui rendent son étude délicate (Figure 16). La problématique du minimum local est au coeur du problème et il n'existe pas de solution idéale actuellement pour garantir un arrêt optimal de l'apprentissage du modèle. Néanmoins, plusieurs approches existent et présentent des résultats satisfaisants. Ces fonctions de régulation sont appelées **Early Stopping**.

#### 4.9.1 Early Stopping

Les méthodes *Early Stopping* reposent sur la valeur de l'erreur de validation. De ce fait, nous définissons  $E_{opt}(t)$ , l'erreur de validation la plus faible jusqu'à l'instant  $t$ , définie par:  $E_{opt}(t) = \min_{t' \leq t} (E_{va}(t'))$  avec  $E_{va}$ , erreur de validation.

##### 4.9.1.1 Generalization Loss ( $GL_\alpha$ )

La première méthode, nommée Generalization Loss ( $GL_\alpha$ )[71], repose sur l'augmentation relative de l'erreur à l'instant  $t$  par rapport à  $E_{opt}(t)$ . Ainsi, si l'augmentation est supérieure à une valeur donnée, l'apprentissage est stoppé. Ce critère examine la tendance absolue de l'évolution de l'erreur de validation et néglige la tendance évolutive locale.

Pour formaliser mathématiquement ce comportement, nous définissons l'erreur

(en pourcentage) par:

$$GL(t) = 100 * \left( \frac{E_{va}(t)}{E_{opt}(t)} - 1 \right)$$

$$Stop : GL(t) > \alpha$$

Cette approche est peu efficace pour lutter contre les minimum locaux. Exploiter ce critère d'arrêt aura tendance à stopper l'apprentissage lorsqu'un minimum local aura été observé. Bien qu'efficace en cas de comportement convexe de la fonction d'erreur, elle est peu performante dans le cas contraire. Expérimentalement, ce critère limitera la capacité d'exploration du réseau durant sa phase d'apprentissage et peut favoriser un arrêt prématuré.

#### 4.9.1.2 Quotient of Generalization Loss and Progress ( $PQ_{alpha}$ )

La méthode  $GL_\alpha$  ignore le comportement du modèle sur les données d'apprentissage. En ignorant ces informations, ce critère n'a pas connaissance de la situation d'apprentissage du réseau. La variation de l'erreur d'apprentissage peut donc être faible ou élevée sans qu'elle n'ait d'impact sur le critère d'arrêt. Expérimentalement, on observe que le sur-apprentissage arrive souvent lorsque l'erreur d'apprentissage décroît "lentement" après une diminution brutale en début d'apprentissage (Figure 17). De même, intuitivement, il est pertinent de penser qu'une variation importante de l'erreur d'apprentissage est corrélée avec une amélioration significative du réseau ou tout du moins, une variation significative de son comportement prédictif. L'impact de cette variation sur le comportement prédictif doit être considéré avant d'imposer l'arrêt de l'apprentissage. Pour considérer cet aspect, la notion de *Progress* ( $P_k$ )[71] est introduite. Elle évalue l'importance de la variation de l'erreur moyenne sur un intervalle donné par rapport à l'erreur minimale observée.

Supposons un intervalle de  $k$  itérations successives et  $E_{tr}(t)$ , erreur d'apprentissage à l'instant  $t$  alors:

$$P_k(t) = 1000 * \left( \frac{\sum_{t'=t-k+1}^t E_{tr}(t')}{k * \min_{t'=t-k+1}^t E_{tr}(t')} - 1 \right)$$

Nous voulons poursuivre l'apprentissage si l'erreur d'apprentissage s'améliore significativement. De ce fait, la valeur de  $P_k$  doit minorer l'expression de  $GL(t)$ . Pour cela, nous définissons  $PQ_\alpha$ [71] tel que:

$$Stop : \frac{GL(t)}{P_k(t)} > \alpha$$

#### 4.9.1.3 Successive Generalization Error ( $UP_s$ )

Les méthodes précédentes exploitent une approche absolue pour l'analyse de la tendance de la courbe d'erreur. Un procédé complémentaire serait de considérer

les tendances locales de la courbe d'erreur. Pour cela,  $UP_1$  observe la variation de  $E_{va}$  pour un intervalle  $[n, n+k]$  donné et provoque un arrêt de l'apprentissage si la variation correspond à une augmentation de l'erreur de validation.

En généralisant, nous obtenons:

$$UP_1 : \text{stop} : E_{va}(t) > E_{va}(t - k)$$

$$UP_s : \text{stop} : UP_{s-1} \rightarrow \text{stop} \cup E_{va}(t) > E_{va}(t - k)$$

**Remarque:** Il est important d'observer le comportement récursif de  $UP_s$  où  $s$  indique le nombre de variations successives considérées.

Cette méthode favorise la capacité d'exploration du modèle en l'émançant de la considération absolue de sa performance<sup>47</sup>.  $UP_s$  est donc plus performant pour détecter le meilleur minimum local que  $GL_\alpha$ . Néanmoins, cette approche facilite le risque de divergence du modèle et peut être inefficace en cas de variations progressives.

Supposons  $UP_{s=4}$ . Si la courbe d'erreur de validation suit un cycle itératif défini par 3 hausses positives<sup>48</sup> de l'erreur ( $e_1^+, e_2^+, e_3^+$ ) suivie d'une diminution de l'erreur ( $e_4^-$ ) tel que  $\sum_{i=1}^4 e_i^{+-} > 0$ , alors le critère d'arrêt est inefficace et la fonction d'erreur divergera.

Ces deux familles de *Early Stopping* peuvent être combinées afin d'exploiter les deux axes d'analyse de la tendance de l'erreur de validation. Il est donc possible de réaliser une combinaisons linéaire de plusieurs de ces critères et de pondérer les critères selon l'importance qu'on souhaite leur attribuer.

#### 4.9.2 Arrêt supervisé

L'apprentissage d'un réseau de neurones est (généralement) une tâche de longue durée qui rend possible une supervision visuelle. Exploiter des outils de visualisation durant l'apprentissage est une approche à ne pas sous-estimer et souvent, présentera les meilleurs résultats.

Les API de Deep Learning proposent des interfaces de visualisation pour faciliter ce travail, notamment *Tensorboard* associé à l'API de Google, *Tensorflow*.

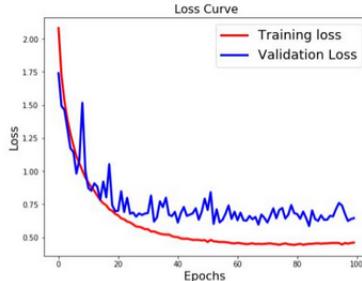
### 4.10 Data Augmentation

Une des difficultés principales des réseaux de neurones est l'exigence d'une quantité importante de données d'apprentissage, quantité qui augmente avec la profondeur du réseau. Il n'est pas rare d'utiliser des jeux d'apprentissage contenant

---

<sup>47</sup>Le critère d'arrêt se limite à observer ses évolutions locales et non absolues

<sup>48</sup>ou moins



On observe que la courbe d'erreur d'apprentissage est comparable à un logarithme inversé. Sur cet exemple, le sur-apprentissage ne semble pas présent.

Figure 17: Exemple de courbe d'erreur d'apprentissage et de validation

des millions voire des milliards d'entités, notamment dans l'étude du langage (Natural language Processing).

Obtenir un jeu de données est difficile, chronophage (et potentiellement cher si il doit être annoté dans le cas du traitement du langage ou de la segmentation d'image). Il est donc indispensable d'utiliser des méthodes afin, à partir d'une entité, d'en créer d'autres de manière artificielle. Il est possible de créer tout type de données de manière artificielle à partir d'une base suffisante le permettant (texte, signal 1D, 2D, ...). Chaque type de données utilise des méthodes différentes propres à son type.

Ainsi, dans le cas d'une image (signal 2D), plusieurs approches sont possibles:

- **Variation géométrique:** Rotation, inversion, symétrie, translation
- **Analyse de densité de la distribution de pixels (via l'histogramme):** contraste, filtrage, effet flou
- **Analyse statistiques:** Création par PCA, Création par ZCA
- **Signal bruité:** Ajout de bruit gaussien, bruit poivre et sel, bruit multiplicatif
- **Création de contenu:** Création de patterns génériques (fusionner deux photos par "collage" brut par exemple - efficace dans le cadre de la segmentation)

Il existe de nombreuses autres méthodes et ce, pour tout type de données. Un jeu d'apprentissage est de qualité si il est le plus représentatif et diversifié possible. Cette étape est donc importante car un jeu de données de qualité est une **condition nécessaire et primordiale** pour un apprentissage de qualité.

**Peu importe la qualité du modèle, si les données sont de médiocre qualité, l'apprentissage sera mauvais.**

## 4.11 Complexité de l'architecture et mémoire

Le Deep Learning a connu un regain de popularité grâce à l'évolution matérielle qui, dorénavant, supporte ce type d'architecture. Néanmoins, les réseaux très profonds demandent des ressources très importantes qui ne sont pas à la portée de tous (particulier comme professionnel). Un modèle complexe demande un temps très important d'apprentissage et le temps de prédiction peut être trop important pour supporter les conditions du temps réel (analyse vidéo ou vocale par exemple). De plus, une contrainte importante de mémoire dédiée peut avoir lieu dans le cadre de structures limitées telle que l'embarqué. Il est donc important de proposer des approches pour optimiser l'architecture d'un réseau. En effet, il est **important** de ne pas associer la profondeur d'un réseau avec une preuve de qualité. Un réseau très profond peut être moins performant qu'un réseau plus simple bien que sa capacité d'abstraction soit plus forte. Les difficultés d'apprentissage sont un facteur déterminant et il est souvent plus aisés d'optimiser un réseau plus petit qu'affronter la complexité d'un modèle théoriquement plus performant mais plus délicat à développer.

Pour cela, différentes méthodes de simplification de modèle existe, notamment *Deep Compression*[26]. Cette méthode reprend l'idée de suppression des poids faibles issue du *Dense-Sparse-Dense training*. L'objectif de cette compression est de limiter le temps d'apprentissage du modèle (de ce fait, sa complexité) et la mémoire nécessaire pour le stocker. L'approche *Sparse* limite la complexité du modèle et l'optimisation de la mémoire est réalisée en deux étapes. Nous ne détaillerons pas ces deux étapes en détails, pour cela, veuillez vous référer au papier associé. *Deep Compression* se déroule ainsi:

1. **Sparse:** Suppression des poids faibles du réseau
2. **Optimisation des poids et des gradients:** Afin de limiter le stockage de valeurs, une approche par clustering (Kmeans) est réalisée afin de réaliser des clusters de poids semblables. Ces poids seront donc, dans un premier temps, représentés par la valeur du centroïde<sup>49</sup> du cluster (souvent associé au barycentre). Les matrices du réseau seront donc associées à des index pointant sur la valeur du centroïde concerné. Les gradients de chacun des poids sont calculés, additionnés entre eux selon la répartition de leurs poids associés parmi les clusters définis précédemment et les gradients de chaque cluster multipliés par le pas d'apprentissage. La valeur des centroïdes sera alors mise à jour par la soustraction de la valeur du centroïde avec celle du gradient du cluster associé.
3. **Application du codage de Huffman:** Afin de limiter le poids des matrices de poids (ou d'index), on applique un codage de Huffman. le

---

<sup>49</sup>L'initialisation du Kmeans est une étape importante à étudier et à ne pas négliger

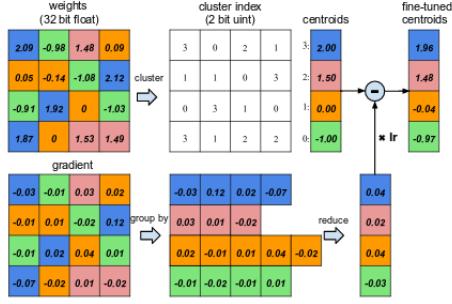


Figure 18: Optimisation réalisée sur les matrices de poids par Deep Compression

codage de Huffman permet de réaliser une compression de données sans perte.

Un exemple illustratif de la phase 2 est visible sur la Figure 18. Il existe d'autres méthodes bien que ce genre d'approche soit, aujourd'hui encore, expérimental et peu appliquée dans l'industrie. Cependant, il est fort probable qu'elles se développeront dans le futur du fait de la complexité grandissante des modèles créés.

## 4.12 Transfer Learning

**Important:** Cette partie va supposer que vous avez une connaissance générale des **réseaux convolutifs**. Si non, veuillez vous référer à la Section 11. On supposera le cas d'un réseau convolutif pour illustrer cette partie.

L'une des grandes problématiques du Deep learning est son temps d'apprentissage. Même sur des modèles jugés simples, l'apprentissage demande des ressources matérielles importantes et ce, pour une longue durée. L'objectif du *Transfer Learning* est d'étudier les capacités d'un apprentissage à se généraliser à différents problèmes. Par exemple, un réseau convolutif ayant appris sur un jeu de d'images spécifiques peut-il être réutilisé sur un autre jeu d'images ? Si oui, dans quelle mesure ? Cette problématique est l'un des sujets de recherche parmi les plus actifs avec des retombées applicatives très importantes. Cette fonctionnalité est très importante dans le cas d'analyse d'image ou de texte (projection vectorielle de mots ou Words Embeddings)

Classiquement, les couches de convolution d'un réseaux convolutifs peuvent être interprétées comme des extracteurs d'attributs d'une image, i.e extraire l'information utile et discriminante. Ces attributs sont alors différenciables par un réseau Feed-Forward standard. Il est évident que le modèle Feed-Forward est entraîné à discriminer les entités selon le jeu de données d'apprentissage.

Il n'est pas pas ou très peu ré-utilisable dans d'autres contextes<sup>50</sup>. Mais il est logique de se demander si extraire des attributs d'une image est généralisable ou spécialisé d'un jeu de données particulier.

Les couches de convolution agissent comme des couches d'abstraction de l'image. Les premières couches isolent grossièrement les attributs de l'image et les couches finales, analysent ces attributs de manière plus spécifique et détaillée. Ainsi, plus le modèle est profond, plus sa représentation finale des données aura un haut niveau d'abstraction spécifique aux données d'apprentissage et ses couches finales seront très affiliées au données d'apprentissage (couches généralistes  $\rightarrow_{\infty}$  couches spécialisées). Les couches de convolution peuvent ainsi être généralisées car l'extraction d'attribut d'une image n'est pas nécessairement associée à une catégorie de données spécifiques (notamment dans les premières couches de convolution) car très généralistes.

Pour illustrer, supposons que nous souhaitons différencier deux voitures. Tout d'abord, nous considérerons la taille, la couleur dominante par exemple, i.e le comportement des premières couches de convolution alors que par la suite, nous étudierons la forme des jantes ou des rétroviseurs. Analyser la taille et la couleur peut être utile pour discriminer deux chats mais analyser la forme des jantes est peu efficace... Ainsi, le *Transfer Learning* permet de limiter le "gaspillage" d'apprentissage mais des précautions doivent être prises selon les spécificités du modèle qu'on souhaite réutiliser et les nouvelles données d'apprentissage.

Aujourd'hui, il existe des jeux de données variées et de grande qualité (ImageNet, Cifar,...). Ces jeux de données sont très diversifiées (avec des centaines voire des milliers de classes). Ils sont donc généralistes<sup>51</sup>. De nombreux modèles pré-entraînés sur ces données sont en open-source et facilitent grandement le temps d'apprentissage.

Trois possibilités s'offrent à l'utilisateur:

1. **Extracteur figé:** Dans cette configuration, on juge que les couches exportées sont fiables et finalisées. On figera donc leurs poids en bloquant les modifications par rétropropagation. Dans le cas d'une extraction de couches de convolution, la sortie obtenue par ces couches est appelée **CNN codes**.
2. **Extracteur variable:** Dans cette configuration, les couches exportées ne sont pas figées et continuent d'apprendre durant la spécialisation du réseau sur le nouveau jeu de données. Le *Transfer learning* agit donc comme une initialisation des poids du réseau (bien plus pertinent que l'aléatoire).

---

<sup>50</sup>Un réseau qui a appris à discriminer un chat ne pourra discriminer une voiture

<sup>51</sup>Si on cherche à étudier des entités d'un même domaine d'activité. Etudier des images de microbiologie à partir d'un modèle ayant appris sur des images issues d'expériences en astrophysique sera difficile quelque soit la diversité de l'apprentissage sur son domaine.

3. **Modèle pré-entraîné:** Bien qu'il y ait des modèles puissants réalisés (en général) par des universités et/ou des entreprises privées, une communauté importante s'est développée pour favoriser la création de modèles déjà pré-entraînés. Ces modèles sont souvent moins "performants" que les modèles réputés et reconnus du fait de la différence de ressources et de temps d'apprentissage du réseau. Néanmoins, il est possible de trouver des réseaux ayant appris sur un jeu de données plus semblables au votre que les modèles standards ayant appris sur des données généralistes. Ces modèles offrent donc une alternative très performante notamment dans le cadre de l'initialisation de poids.

Le choix de l'approche à choisir dépend essentiellement de la taille du nouveau jeu de données et de leurs différences.

- **Nouveau jeu de données similaire au jeu d'apprentissage et de petite taille:** A cause de la petite taille du nouveau jeu de données, le risque de sur-apprentissage est important. Comme le jeu de données d'apprentissage et proche du nouveau jeu de données, il est préférable de figer les couches de convolution et d'apprendre uniquement un nouveau modèle Feed-Forward pour la classification.
- **Nouveau jeu de données similaire au jeu d'apprentissage et de grande taille:** Grâce à la grande taille du jeu de données, il est possible de mettre à jour les poids des couches transférées car le risque de sur-apprentissage est plus faible. Le Transfer Learning joue un rôle d'initialisation des poids dans cette configuration et l'apprentissage spécialisera votre modèle.
- **Nouveau jeu de données de petite taille et très différent du jeu d'apprentissage:** Cette situation est peu propice au Transfer Learning. En effet, le jeu d'apprentissage de petite taille ne permet pas un apprentissage des poids performant, il est donc nécessaire de limiter la mise à jour des poids des couches transférées. Cependant, la différence importante entre les deux jeux de données présente un risque de mauvaise performance due à la sur-spécialisation de ces couches. Afin de trouver un compromis, il est intéressant de ne pas garder **l'intégralité** des couches de convolutions mais d'en garder un sous-ensemble en supprimant les dernières couches qui présentent la spécialisation la plus forte. Ainsi, nous transférons que les couches les plus généralistes qu'on peut supposer performantes pour différencier des images de catégories différentes. Bien sûr, un nouveau modèle Feed-Forward doit être appris intégralement.
- **Nouveau jeu de données différent du jeu d'apprentissage et de grande taille:** Dans cette configuration, la grande taille du jeu de données permet un apprentissage. Néanmoins, la différence entre les deux jeux pose problème quant à la quantité des couches à transférer. Il est souvent préférable, grâce à la capacité d'apprentissage du nouveau jeu de données, d'utiliser un réseau pré-entraîné (autres que les modèles standards) qui a

apris sur des données comparable aux vôtres afin d'initialiser les poids et d'utiliser vos données pour le spécialiser en mettant à jour les couches transférées. Un nouveau modèle Feed-Forward doit être appris.

**Important:** Le modèle Feed-Forward n'est pas impératif. En effet, d'autres algorithmes d'apprentissage automatique peuvent être utilisés notamment le SVM ou le XGboost. Néanmoins, les réseaux de neurones présentent une bonne efficacité et permettent une implémentation plus aisée en permettant la réalisation d'un modèle qui limite les dépendances de concepts.

Il est très difficile de considérer l'ensemble des caractéristiques d'un phénomène, c'est pourquoi la capacité de généralisation d'un modèle de *Machine Learning* est très importante. Il n'est pas capital de représenter l'intégralité des hypothèses possibles<sup>52</sup> mais il est nécessaire de survoler l'ensemble des grandes possibilités de représentation. Plus ce travail de diversité sera fait, plus le jeu de données sera de qualité. Il est donc évident qu'un jeu de données de grande taille aura tendance à être de meilleure qualité. Néanmoins, ce n'est pas une condition suffisante !

## 5 Réseaux convolutifs

**Important:** Tout ce qui a été vu précédemment s'applique à la spécificité des réseaux convolutifs.

### 5.1 Généralités

Les réseaux convolutifs constituent l'une des grandes familles d'architecture associées aux réseaux de neurones. Popularisés par Yann Lecun en 2012, ces réseaux ont montré des résultats remarquables pour l'analyse de données structurées telles que les images (mais aussi les signaux ou textes). Mais pourquoi cette architecture est préférable à un modèle Feed-Forward (Perceptron Multicouche) standard ?

Supposons une image de petite taille soit  $15 \times 15$  par exemple. L'image est codée en RGB (images couleurs). Ainsi, l'image possède  $15 \times 15 \times 3$  valeurs distinctes correspondant à ses pixels. Dans le cas d'un modèle Feed-Forward, les neurones de la couche d'entrée devront avoir une entrée (et donc un poids) associée à chacune des valeurs de l'image soit  $15 \times 15 \times 3 = 675$  entrées. Bien que ça puisse impressionner, les capacités de calculs des technologies d'aujourd'hui peuvent le supporter<sup>53</sup>. Considérons une image de taille un peu plus standard soit  $500 \times 500$ . On obtient donc  $500 \times 500 \times 3 = 750 \times 10^3$  poids par neurones de la couche d'entrée. Il est évident que le nombre d'entrées est bien trop important et rendrait l'apprentissage irréalisable. De même, le risque de sur-apprentissage

---

<sup>52</sup>C'est d'ailleurs un travail irréalisable

<sup>53</sup>Bien sûr, il ne doit pas y avoir trop de couches cachées...

est très important. De ce fait, les modèles Feed-Forward classique ne **peuvent pas se mettre à l'échelle**.

Les couches de convolution permettent d'extraire l'information d'une image et d'en réaliser une représentation à un plus haut d'abstraction. Ces couches présentent deux avantages notables. Tout d'abord, elles permettent de représenter le contenu utile d'une image dans une dimension plus faible<sup>54</sup> que l'image d'origine<sup>55</sup>, ce qui permettrait à un modèle Feed-Forward d'apprendre dessus. Dans un second temps, les couches de convolution réalisent le travail d'extraction d'information réalisée traditionnellement par des méthodes-tiers comme HoG par exemple et ce, avec une capacité d'apprentissage. Alors que les méthodes-tiers sont des méthodes figées et généralistes, les couches de convolution se spécialisent dans la discrimination des images de sa base d'apprentissage. Ainsi, elles sont souvent plus efficaces et mieux optimisées<sup>56</sup>.

Un réseau convolutif possède (dans sa version générale) l'architecture suivante:

- **Couche d'Entrée:** Ceci correspond à la donnée proposée au réseau pour son apprentissage ou une prédiction. L'entrée stocke ainsi l'intégralité des valeurs de la matrice de l'image. Elle sera donc de dimension  $X*Y*3$  dans le cas d'une image de dimension  $X*Y$  en RGB.
- **Couche d'extraction de caractéristiques:**

Le cycle suivant est répété  $k$  fois en accord avec l'architecture du réseau.

1. **Couche de convolution:** Les couches de convolution vont extraire l'information de l'image et la représenter à un plus haut niveau d'abstraction. Il peut y avoir une ou plusieurs couches de convolution. La nature des sorties d'une couche de convolution se présente sous la forme d'une matrice de dimension  $x*y*z$  où  $x,y,z$  dépendant des paramètres de la couche<sup>57</sup>.
2. **Couche d'activation:** Cette couche réalise une activation selon la fonction d'activation utilisée sur les valeurs de la matrice de sortie de la couche de convolution. ReLU est la fonction la plus populaire actuellement mais d'autres peuvent être utilisées.
3. **Couche de régulation:** Afin de limiter le sur-apprentissage, il est utile d'exploiter des couches qui aident à prévenir cette dérive de l'apprentissage. Ces couches n'ont pas de pouvoir d'extraction d'informations et se limitent à une action régulatrice. Ce type de

<sup>54</sup>Après traitement d'une couche de pooling

<sup>55</sup>On peut comparer ce procédé à un pré-traitement des données

<sup>56</sup>Il a été montré qu'elles offrent une capacité de discrimination assez généraliste dans les couches de bas niveau malgré tout

<sup>57</sup> $x$  et  $y$  inférieurs ou égaux à la dimension  $X$  et  $Y$  de l'entrée

couche modifie localement l'architecture du réseau (DropOut par exemple) ou les valeurs de sortie des couches de convolution (Batch Normalisation par exemple).

4. **Couche de redimensionnement:** Les sorties des couches de convolution peuvent être dans un format qui ne facilite pas l'apprentissage et/ou l'optimisation de ressources matérielles. Pour limiter ce problème, un redimensionnement vers une dimension inférieure (ou supérieure) est souvent réalisé. Le redimensionnement, selon l'objectif souhaité, s'applique selon une dimension spécifique du signal donné.
- **Couche de classification (Full-Connected i.e perceptron multicouche):** Cette couche réalise la décision du réseau à partir de l'information extraite des couches de convolution.

Il est important de noter que certaines couches possèdent des paramètres et d'autres non. En effet, les couches ReLu et Pooling appliquent une transformation fixe sur les données (donc aucun paramètre<sup>58</sup>) alors que les couches de convolution et Full-connected "apprennent"<sup>59</sup> durant l'apprentissage et donc, possède des paramètres.

## 5.2 Couche d'Entrée

La couche d'Entrée est représentée par une couche de neurones où un neurone se limite à la propagation d'une valeur de la matrice de l'image observée. Chaque neurone est ainsi défini par une pondération unitaire avec une fonction d'activation linéaire. Il y a autant de neurone que de pixels dans l'image (d'où l'importance d'avoir des images de taille constante).

## 5.3 Couche de Convolution

### 5.3.1 Nature d'une couche de convolution

Les couches de convolution suivent une architecture Feed-Forward modifiée par la condition de connectivité locale, i.e un neurone de la couche de convolution n'est pas connecté à l'intégralité des neurones de la couche précédente (Entrée ou autre couche de convolution) mais uniquement aux neurones "les plus proches". Cette particularité diminue drastiquement le nombre de liaison des neurones, permettant leur apprentissage en temps humain. Le degrés de connectivité locale est un paramètre variable dépendant de la taille du filtre souhaité. Néanmoins, bien que la taille du filtre soit variable, sa profondeur reste constante et dépendante de la profondeur de l'image d'entrée. Par exemple, supposons une

---

<sup>58</sup>Ceci n'est pas entièrement vrai pour les couches de Pooling car il existe des variantes qui exploitent des paramètres. De même, des variantes plus avancées de la fonction ReLU apprennent durant l'apprentissage

<sup>59</sup>Par rétro-propagation

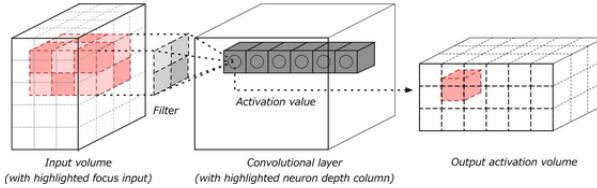


Figure 19: Architecture d'une couche de convolution

image RGB. Sa profondeur est de 3 (channels rouge, vert et bleu). Chaque neurone possédera une connexion avec chacun de ces trois channels en accord avec les limitations du critère de connectivité locale. Une représentation graphique est visible sur la Figure 19

Chaque neurone d'un même filtre possède un même paramétrage. Ainsi, chaque neurone possède le même biais et le même vecteur de pondération. Ils sont donc tous *identiques*. Cette spécificité implique une *invariance par translation* car, comme chaque neurone est identique, la localisation spatiale des pixels n'a pas d'influence sur l'analyse par un filtre donné. De ce fait, une couche de convolution limite grandement le nombre de paramètres tout en conservant les corrélations spatiales locales des images. Différents filtres peuvent être appliqués sur une entrée.

Un filtre représente le vecteur poids du neurone. Les poids des neurones ne sont pas identiques pour chaque channel. Ainsi, dans le cas d'une image RGB, un filtre possédera 3 sous-filtres qui seront appliqués sur chacun des channels de manière indépendante. Les 3 résultats obtenus par les sous-filtres seront unis lors du calcul de la convolution. Dans la configuration standard, on considère souvent un biais nul et une fonction d'activation identité.

Du fait de l'utilisation d'une architecture de neurones, l'apprentissage des filtres se fait par rétro-propagation. Grâce à la connectivité locale du réseau et de l'unicité des pondération par filtre, le nombre de paramètres à apprendre est drastiquement diminué et permet au réseau d'apprendre en temps humain.

### 5.3.2 Interprétation graphique

En vulgarisant, une convolution est une opération mathématique qui décrit une règle sur comment unir deux données. Une convolution prend ainsi une entrée, applique un kernel et réalise une *feature map* en sortie.

Dans le cadre d'une couche de convolution, la convolution est réalisée par un produit scalaire entre un filtre (kernel) et l'entrée. Un exemple est visible sur la Figure 21. Le filtre est de dimension inférieure à l'entrée (dans le cadre d'une image, inférieure à hauteur\*longueur) mais de profondeur égale à la profondeur

de l'image (nombre de channels d'entrée). Un filtre se comporte comme une *fenêtre glissante* afin de parcourir l'intégralité de la données d'entrée. La matrice de sortie obtenue est appelée *feature map*. Il y a autant de *feature map* que de filtre ainsi, si il y a 4 filtres appliquées sur la donnée d'entrée, il y aura 4 *feature map* en sortie.

La *fenêtre glissante* est associée à un hyperparamètre appelé **stride**. En effet, il est possible de définir les intervalles de déplacement. Un stride de 1 implique de réaliser une convolution en appliquant le filtre sur chaque valeur, un stride de deux impliquera un saut de une valeur entre chaque calcul de convolution, etc... Un exemple est visible sur la Figure 20. Le stride est très utilisé pour diminuer la dimension des couches de convolution, notamment pour limiter le nombre d'entrée de la couche Feed-Forward qui réalise la décision.

**Important:** Le kernel est appliqué sur une valeur de la matrice d'entrée en son **centre**, pas l'un de ses sommets. De plus, un filtre ne réalise la convolution que si chaque élément du filtre s'applique à une valeur de la matrice d'entrée. Observons la Figure 22. Sur la représentation de gauche, chaque entité du kernel est associée à une valeur. La convolution peut être définie. A droite, le kernel "dépasse". Le filtre ne peut donc intégralement s'appliquer à la matrice d'entrée: la convolution n'est pas réalisée.

Cette particularité soulève une problématique. Dans cette configuration, la sortie obtenue après convolution ne peut être que de dimension inférieure à l'entrée à cause des effets de bord, effet qui s'aggrave avec des filtres de grandes tailles. Bien qu'une sortie de dimension inférieure ne soit pas un problème en tant que tel (ce serait même bénéfique d'un point de vue temps-machine), elle dénonce une perte d'information de l'entrée sur ses bords. Supposons un kernel de dimension  $K \times K$  et une entrée  $D \times D$ , alors la dimension de la sortie sera de dimension  $(D-K+1) \times (D-K+1)$ <sup>60</sup>.

Afin de traiter ce problème, la méthode du **padding** est appliquée. Cette méthode consiste à appliquer une valeur arbitraire et constante afin de combler les valeurs manquantes de la matrice d'entrée. Par convention, la valeur utilisée est 0 (d'où le nom standard de 0-padding<sup>61</sup>). Un exemple est visible sur la Figure 23. Cette approche permet de considérer l'information de l'intégralité de la donnée d'entrée et de créer une sortie de même dimension que l'entrée.

Le site Web <https://ezyang.github.io/convolution-visualizer/index.html> propose une application intuitive pour visualiser l'impact de différentes configurations sur le comportement des filtres.

En pratique, il faut éviter les filtres de grande taille ( $5 \times 5$ ,  $7 \times 7$  et supérieurs) sauf

---

<sup>60</sup>On supposera un stride de 1

<sup>61</sup>Parfois, la valeur 1 est employée

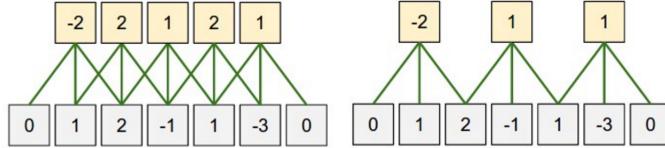


Figure 20: Impact du stride sur la sortie de la convolution

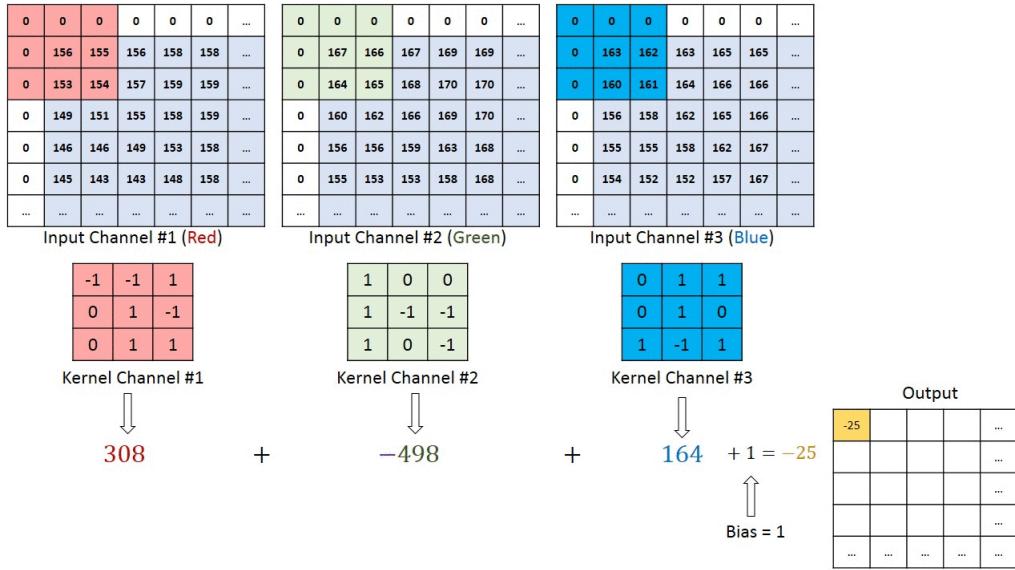


Figure 21: Exemple d'une convolution par filtre

si il y a une vraie pertinence dans le choix de cette échelle de filtre. Les filtres de grande taille sont plus lents à apprendre, gourmand en ressource machine et peuvent être remplacés par des alternatives plus légères avec une efficacité similaire. Il est donc préférable de cumuler des filtres  $3 \times 3$  dans ce type de cas. Cette approche simple demande moins de paramètres, moins de ressources machine et surtout, plus de non-linéarité.

### 5.3.3 Couche de Pooling

Le *Pooling* est une méthode permettant de compresser (et généraliser) une information locale de la donnée d'entrée. Elle est employée afin de diminuer la dimension des *feature map* tout en conservant une capacité de généralisation à la couche. En effet, le Pooling conserve l'information locale importante, permettant de s'émanciper de détails trop spécifiques (et du bruit).

Le fonctionnement d'une couche de Pooling est comparable à celle d'une couche

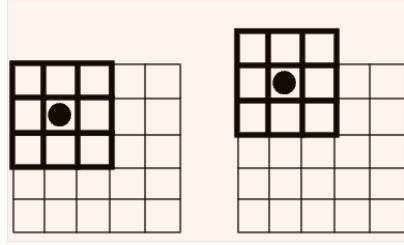


Figure 22: Problématique de la dimension du filtre: gauche (valide), droite (invalide)

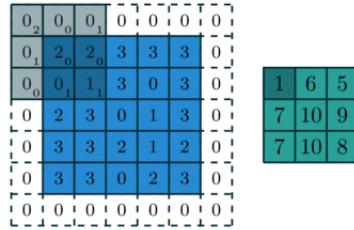


Figure 23: Application du 0-Padding

de convolution si ce n'est qu'elle s'applique sur une *feature map* indépendamment des autres (la profondeur du kernel est de 1). Un filtre (de taille choisie) est appliquée afin de définir *l'environnement local*<sup>62</sup> et un opérateur est appliqué sur les valeurs d'entrée ciblées par le filtre. La nature de cet opérateur est variable: Max, Average<sup>63</sup>. Expérimentalement, la fonction Max est la plus utilisée. Sur la Figure 24, nous pouvons observer l'application d'un Max-Pooling de filtre 2\*2.

**Remarque:** Le Pooling doit être exploitée avec parcimonie. Il est souvent un *bottleneck* pour la vélocité du modèle. De même, son utilisation ne fait pas l'unanimité dans le milieu scientifique. En effet, Geoffrey Hinton, chercheur de renom dans le domaine de l'apprentissage profond, a déclaré: "*The pooling operation used in convolutional neural networks is a big mistake and the fact that it works so well is a disaster.*" (Reddit AMA).

Ce scepticisme repose sur plusieurs défauts de cette approche. Le Pooling est une action non évolutive (fixe donc sans apprentissage), ce qui nuit à la capacité de généralisation/specialisation. De plus, son action est locale à une *feature map*. Il y a donc une perte importante d'informations associées aux relations inter-feature map. Pour finir, le Pooling a une capacité explicative lorsque les couches de convolutions présentent un fort taux d'abstraction. Dans le cas con-

<sup>62</sup>Comparable à une isolation spatiale

<sup>63</sup>Ces deux opérateurs sont les plus fréquents

traire, le filtre est souvent de taille trop faible pour considérer une information discriminante. Augmenter la taille du filtre est envisageable mais le Pooling étant *destructeur*, l'augmentation du filtre aurait des conséquences désastreuses sur la transmission de l'information aux couches supérieures.

Plus récemment, des propositions de fonctions de Pooling ont été faites. Nous pouvons citer:

- **$L_p$  Pooling**[37]: Cette approche est issue des études biologiques des cellules vivantes. Elle est définie par:  $y_{m,n,k} = \left[ \sum_{(i,j) \in R_{mn}} (a_{i,j,k})^p \right]^{\frac{1}{p}}$  avec  $R_{mn}$ , surface du filtre et  $k$ , channel analysé.

Il s'agit de la norme  $L_p$  des valeurs ciblées par le filtre du Pooling. Ainsi, si  $p=1$  alors  $L_p$  correspond à l'average Pooling. Si  $p = \infty$ ,  $L_p$  correspond au max Pooling. Cette méthode permet de réaliser un compromis entre l'isolation discriminante d'une information et la généralisation du message locale.

- **Mixed Pooling**[108]: Cette méthode combine max Pooling et average Pooling au sein d'une combinaison linéaire dont les coefficients pondérateurs dépendent d'un paramètre  $\lambda$  tel que  $\lambda \in \mathcal{U}(0,1)$ . Ainsi, nous obtenons:

$$y_{m,n,k} = \lambda \max_{R_{mn}} a_{i,j,k} + (1 - \lambda) \frac{1}{|R_{mn}|} \sum_{(i,j) \in R_{mn}} a_{i,j,k}$$

Expérimentalement, elle présenterait de meilleurs résultats que l'approche max ou average tout en proposant une plus grande résistance au surapprentissage.

- **Gated max-average Pooling**[108]: Cette approche est similaire à *Mixed Pooling* mais considère l'état de la surface couverte par le Pooling. En effet, *Mixed Pooling* applique un coefficient indépendamment des valeurs couvertes par le filtre de Pooling. *Gated max-average Pooling*, au contraire, va réaliser une pondération évolutive selon les valeurs couvertes par le filtre. On suppose  $x$ , valeurs couvertes par le filtre de Pooling et  $w$ , "Gate" du Pooling. Cette idée est traduite par:

$$f_{gate} = \sigma(w^T x) f_{max}(x) + (1 - \sigma(w^T x)) f_{avg}(x)$$

$$\sigma(w^T x) = \frac{1}{(1 + \exp(-w^T x))}$$

La "Gate" peut être unique à un réseau, une couche, à une région de la couche mais identique sur toute la profondeur ou au contraire, différente sur toutes les profondeurs. Néanmoins, cela ajoute des paramètres à apprendre et donc un ralentissement de la vitesse d'apprentissage. Cette approche présente des résultats plus performants que *Mixed Pooling*.

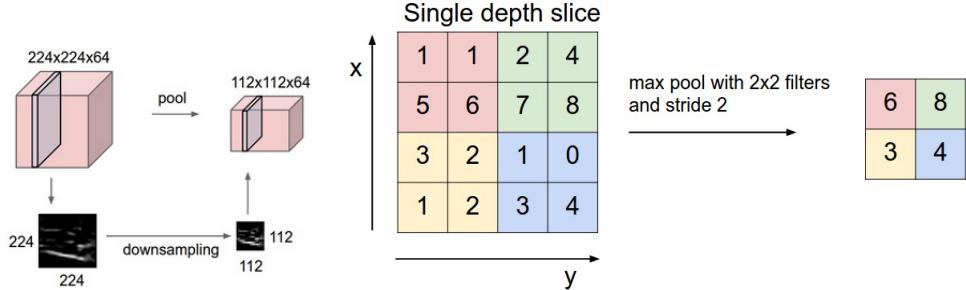


Figure 24: Application du Pooling: Réduction et exemple avec la fonction max selon un filtre 2\*2

- **Stochastic Pooling**[111]: Cette méthode choisit une valeur selon une distribution de probabilité définie par  $p_i = \frac{a_i}{\sum_{k \in R_{mn}} a_k}$  avec  $i \in [1, |R_{mn}|]$  et  $R_{mn}$ , l'ensemble des valeurs isolées par le filtre du Pooling.

Ainsi, plus une valeur est élevée, plus la chance d'être choisie est importante. Cette méthode permet de limiter le sur-apprentissage comparée au max-Pooling en introduisant un comportement probabiliste sur la sélection de la valeur.

**Remarque:** Dans les faits, ces méthodes de Pooling sont assez marginales et peu exploitées. L'écrasante majorité des réseaux actuels exploitant des approches par Pooling reposent sur l'opérateur Max ou Average.

L'opérateur Max est utile lorsque l'on cherche à extraire une information discriminante au détriment des autres. L'objectif est d'isoler les éléments les plus discriminants afin de les analyser par la suite (notamment par un réseau Full-Connected). Cette approche est très utile pour la classification d'images par exemple. En effet, pour classer, un élément discriminant est nécessaire et suffisant. D'autres données moins informatives risqueraient de se comporter comme une forme de bruit.

L'opérateur Average ne se focalise pas sur les éléments discriminants du signal mais sur son comportement général. Sa valeur est donc un résumé global de ses caractéristiques et non d'un aspect discriminant spécifique. Cet opérateur est donc moins destructeur mais possède un pouvoir discriminant plus faible.

La couche de Pooling, selon la configuration, peut conserver des informations utiles pour d'autres couches. Par exemple, dans le cas du Max-Pooling, une mémorisation de la localisation de la valeur max sur la matrice de données peut être réalisée. Cette information permet de réaliser des mises à l'échelle (via Unpooling) avec plus de fidélité mais demande de la mémoire pour stocker les

valeurs.

**Remarque:** Expérimentalement, on a tendance à éviter le chevauchement des filtres de Pooling. Ainsi, on exploite un stride en équation avec la dimension du kernel pour que chaque valeur de la *feature map* soit analysée une unique fois. Par exemple, dans le cadre d'un kernel  $2 \times 2$ , on aura tendance à utiliser un stride de 2.

### 5.3.3.1 Global Pooling

Global Pooling est l'équivalent du pooling mais son application n'est pas limitée par la fenêtre correspondant à la taille du filtre. Il ne s'effectue donc pas sur un sous-ensemble (défini par le filtre) d'une *feature map* mais sur l'intégralité d'une *feature map*. Une *feature map* est donc représentée par une unique valeur à l'issu d'un Global Pooling. Ainsi, supposons une donnée de la forme  $15 \times 15 \times 3$  avec 3, profondeur<sup>64</sup> de la sortie. La sortie sera donc représentée par un vecteur de dimension dans  $R^3$ .

Le Global Pooling est une méthode utilisée pour lutter contre le sur-apprentissage et diminuer le volume des calculs des réseaux convolutifs, notamment de la couche Full-connected réalisant la classification. Les capacités de régulation du Global Pooling s'expriment sous différents aspects:

- **Sans apprentissage:** Étant donné que le Global Pooling considère l'intégralité d'une *feature map*, il n'y a pas d'apprentissage associé à cette couche, ce qui n'ajoute pas de temps d'apprentissage
- **Optimisation de la relation Label - Feature Map:** La couche de Full-connected réalise la prédiction à partir des *feature map*. Les *feature map* sont donc les porteuses d'informations qui doivent retranscrire les spécificités d'un label (catégorie). Il existe un risque important que les feature maps "brutes" soient trop spécifiques du fait des dimensions matricielles élevées de leurs structures, ce qui provoque des dérives vers le sur-apprentissage. L'approche du Global Pooling, en limitant à une valeur par *feature map*, favorise la correspondance entre une *feature map* et une caractéristique générale.
- **Généralisation Spatiale:** En résumant une *feature map* à une valeur, le Global Pooling permet de généraliser l'information. Cette méthode permet donc d'extraire une caractéristique indépendamment de considérations spécifiques due à l'image d'entraînement telles que l'angle de vue, le contraste ou encore la luminosité par exemple.
- **Temps d'apprentissage:** Le Global Pooling limite grandement le nombre de valeurs d'entrée de la couche Full-connected. Cette méthode permet

---

<sup>64</sup>Nombre de *feature map* présents dans la sortie considérée

donc d'en diminuer la taille, ce qui rendra le modèle plus rapide à apprendre, à calculer ses prédictions et plus léger d'un point de vue mémoire.

Le Global Pooling est une approche de plus en plus utilisée pour réaliser la liaison entre l'ensemble des couches de convolution et le réseau Full-Connected, remplaçant ainsi la méthode dite *Flatten* classique qui consiste à vectoriser les features map dans leur intégralité. Néanmoins, Global Pooling est très destructeur et présente des faiblesses notamment dans le cas de données textuelles (classification de textes par réseau convolutif par exemple).

### 5.3.3.2 k-Max Pooling

Afin de corriger le comportement destructeur du Global Pooling, k-Max Pooling[43] applique l'opérateur pour conserver les  $k$  valeurs<sup>65</sup> les plus importantes (au lieu d'une seule pour le Global Pooling). Du fait d'un résultat non unitaire, l'opérateur Average n'est pas exploitable par cette approche.

Le choix de la valeur de  $k$  peut être difficile à réaliser. Une valeur trop faible posséderait les mêmes faiblesses qu'un Global Max Pooling et une valeur trop élevée nuirait à la capacité de généralisation recherchée par l'exploitation de ce type de couche. De plus, comme le Pooling s'exerce sur une *feature map* uniquement, la profondeur est à considérer lors du choix de la valeur de  $k$ . En effet, en supposant  $n$  *feature map* en entrée, la sortie de la couche de k-Max Pooling sera de dimension  $n * k$ .

Les couches de convolution permettent d'extraire l'information utile d'un signal. Néanmoins, elles sont directement dépendantes de la qualité de la donnée en entrée. Une donnée trop bruitée provoquera un résultat médiocre peu importe la qualité du modèle employé. Une couche de Pooling mal exploitée peut être une source de contre-performance du réseau du fait de son grand pouvoir destructeur. Il est donc intuitif de penser que dans un réseau qui cumule plusieurs couches de Pooling, le pouvoir généralisant doit être progressivement exploité pour ne pas perdre d'informations en amont des couches convolutives. En d'autres mots, dans le cadre du k-Max Pooling,  $k$  doit avoir une valeur plus importante au niveau des couches basses et une valeur plus faible au niveau des couches élevées.

Pour répondre à cette problématique, Dynamic K-Max Pooling [43] a été proposé. Cette approche propose une détermination automatique de la valeur de  $k$  en fonction de l'architecture du réseau et de la position de la couche de Pooling dans ce dernier.

La valeur de  $k$  est définie par:  $k_l = \max(k_{top}, \lceil \frac{L-l}{L} * s \rceil)$  avec  $k_{top}$ , valeur minimale de  $k$  pour les couches supérieures,  $L$ , nombre total de couches de convolution dans le réseau,  $l$ , position de la couche où le Pooling est appliqué et  $s$ ,

---

<sup>65</sup>Si  $k$  est égal à 1, le comportement est identique à un Global Max Pooling

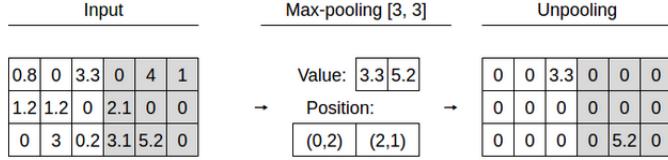


Figure 25: Exemple de Max-Unpooling statique

dimension<sup>66</sup> de la *feature map*.

La fonction proposée permet d'avoir une diminution progressive de la valeur de  $k$  (jusqu'au seuil critique  $k_{top}$  et ainsi, d'augmenter le pouvoir discriminant des couches de Pooling en adéquation avec les caractéristiques informatives des données entrantes. La fonction proposée par [43] est "arbitraire" et ne repose sur aucune véritable justification mathématique si ce n'est l'intuition des auteurs. Elle peut être modifiée afin d'obtenir un comportement plus spécifique sur la vitesse de décroissance. De plus, elle est spécifique à une donnée textuelle. Dans le cadre d'une image, il est pertinent d'envisager une autre fonction qui sera en adéquation avec le comportement d'un signal 2D.

#### 5.3.4 Couche de Unpooling

Le Unpooling est une approche de *Upsampling*, i.e l'augmentation de la dimension de représentation d'une données. Cette approche est utilisée après une couche de pooling lorsqu'une mise à l'échelle est nécessaire. Elle exploite la capacité de la couche de pooling à conserver des informations de position<sup>67</sup> (notamment pour Max-Pooling). La configuration de la *feature map* de sortie dépend de la méthode de pooling choisie. Ainsi, dans le cas d'un max-pooling, seule la valeur à la position correspondant à la position de la valeur max sur la donnée d'entrée (bornée par la surface couverte par le filtre) sera définie, les autres valeurs étant considérées comme 0. Une illustration est visible sur la Figure 25.

L'exemple présenté est statique (réutilise les valeurs max sans apprentissage). Une approche similaire consiste à ne pas utiliser la valeur max de la *feature map* d'entrée et à appliquer un autre filtre, entraînable, afin de définir les valeurs de la nouvelle *feature map*. De ce fait, seules les informations de position sont utilisées. Un exemple est visible sur la Figure 26.

<sup>66</sup>A l'origine, cette approche de Pooling a été proposée pour l'analyse de texte. De ce fait, s correspond à la dimension du texte en entrée. Il est possible de généraliser à un signal 2D comme un image mais il est probable que la fonction doive être modifiée pour mieux correspondre aux caractéristiques d'une image

<sup>67</sup>Dans le cas de l'average-pooling, ce n'est pas nécessaire

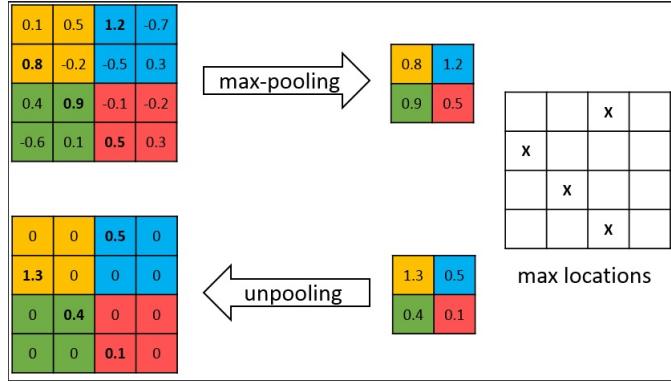


Figure 26: Exemple de Max-Unpooling dynamique

### 5.3.5 Couche de convolution 1\*1

Un cas particulier est la couche de convolution 1\*1. Au premier regard, cette convolution semble non pertinente voire sans intérêt<sup>68</sup>. Néanmoins, du fait de la considération obligatoire de la profondeur de la donnée d'entrée, ce type de convolution possède un intérêt certain. En effet, elle a la capacité de modifier la profondeur de sortie de la couche de convolution où la profondeur correspondra au nombre de filtre présent dans la couche de convolution 1\*1. Supposons une donnée d'entrée de dimension  $X \times Y \times Z$  avec  $Z$ , profondeur de cette donnée. L'application d'une couche de convolution 1\*1 avec  $N$  filtres créera une sortie de dimension  $X \times Y \times N$ . Cette méthode complète l'approche par pooling. En effet, le pooling limite la dimension de la *feature map* et la convolution 1\*1 limite la profondeur de sortie, i.e le nombre de *feature map*.

Cette convolution ne se focalise que sur la valeur ciblée par le filtre sans considération des corrélations locales mais tient compte de l'ensemble des channels de la donnée d'entrée<sup>69</sup>. Elle permet donc d'ajouter de la non-linéarité dans l'architecture du réseau<sup>70</sup>.

### 5.3.6 Couche de convolution dilatée

Les couches de convolution dilatées[109] introduisent un nouveau paramètre: **le taux de dilatation**. Le taux de dilatation détermine l'espace entre chaque valeur effective du kernel. Par exemple, un kernel 3\*3 avec un taux de dilatation de 2 deviendra un kernel de 5\*5 car le filtre ne se focalisera pas sur des éléments adjacents  $a_i, a_{i+1}$  mais sur des éléments distants en accord avec le taux de dilatation. Ainsi, pour une dilatation de 2, les éléments ciblés seront  $a_i, a_{i+2}$ . Les valeurs contenues dans le champ réceptif mais non effectives auront

<sup>68</sup>Pour les lecteurs sensibilisés au traitement du signal, c'est un quasi non-sens

<sup>69</sup>La profondeur est toujours considérée dans son intégralité

<sup>70</sup>Ce qui est souvent favorable aux performances du réseau

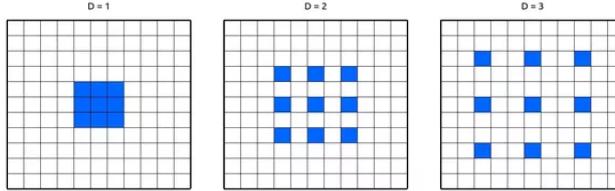


Figure 27: Exemple d'une convolution dilatée

une valeur imposée à 0 (car le poids associé sera nul). La Figure 27 illustre ce type de convolution.

Cette approche permet d'élargir le champ d'analyse de la couche en conservant le même coût de calcul. Elle est très utilisée pour la détection d'entités ou la segmentation car elle offre un bon compromis de performance. Elle permet une analyse générale d'une image sans multiplier les convolutions ou l'utilisation de filtres larges qui sont très gourmands en ressources. Par exemple, supposons un filtre  $3 \times 3$  (stride 1) classique. En combinant 3 couches composées de ce filtre, le champ réceptif de la dernière couche sera de  $7 \times 7$  ( $3 \times 3 \rightarrow 5 \times 5 \rightarrow 7 \times 7$ ). Au contraire, avec une dilatation de 2, la surface couverte sera de  $15 \times 15$ .

### 5.3.7 Couche de convolution transposée

**Remarque:** Cette couche est aussi appelée *fractionally strided convolutions* ou *déconvolution*. L'appellation "déconvolution" est un abus de langage. Cette couche ne réalise pas une déconvolution au sens mathématique du terme.

Une couche de convolution transposée a pour objectif de mettre à une dimension **supérieure** choisie, une donnée d'entrée. Il s'agit ainsi d'une technique de *Upsampling*. C'est très utilisé en segmentation afin de mettre à l'échelle la sortie d'un réseau convolutif.

Son fonctionnement est comparable à une convolution standard avec l'application de padding (ajout de 0 arbitraire) afin de compléter la donnée d'entrée et mettre à l'échelle voulue la *feature map* de sortie. Le padding dépend de la taille du filtre choisie, du stride et de la dimension de sortie du *feature map*. Deux exemples sont visibles sur la Figure 28.

Les couches de convolution transposées permettent uniquement de conserver les dimensions des features map. Ainsi, supposons une entrée N alors l'application d'une couche de convolution sur cette entrée puis d'une couche de déconvolution (avec le même paramétrage de couche que la couche de convolution) permettra d'obtenir une sortie de même dimension que N. Par contre, la sortie n'est **pas égale** à N ! D'où l'abus de la dénomination *déconvolution* qui est faux mathématiquement parlant.

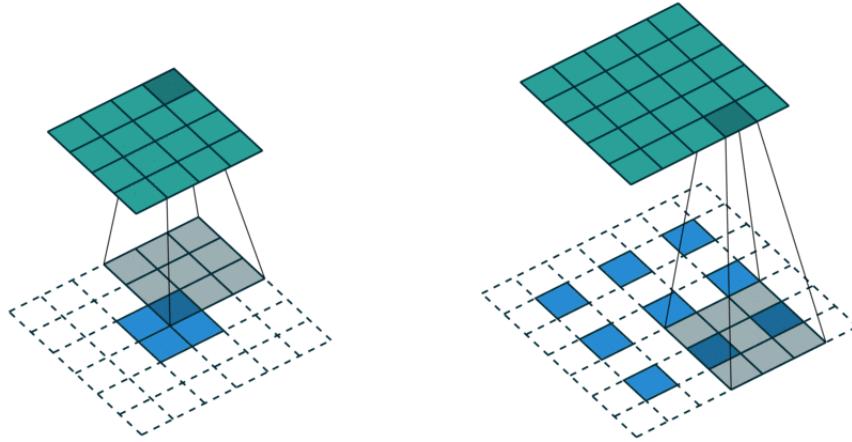


Figure 28: Exemple de convolution transposée: Gauche: entrée  $2 \times 2$  sans stride, Droite: Entrée  $3 \times 3$  avec stride de 1

### 5.3.8 Couche de Depthwise/Pointwise

Bien qu'efficaces, les couches de convolution standards restent gourmandes en temps de calcul et en mémoire nécessaire pour stocker le modèle. Dans des contextes particuliers avec des limitations de ressources tels que l'embarqué ou encore les smartphones/tablettes, l'utilisation de modèle très profond classique n'est pas envisageable pour cause de limitation matérielle. En plus des méthodes standards de réduction vues dans les Sections précédentes, il est possible d'agir au niveau de l'architecture-même des couches de convolutions avec les couches de **Depthwise/Pointwise** (ou separable depthwise). Ce type de couche a été popularisé par la création de l'architecture *MobileNet*[31] créée par Google afin de répondre aux problématiques des prédictions sur smartphone.

Le comportement général de ce type de couche est comparable aux couches de convolution standards. La différence se situe au niveau des spécificités du nombre de filtres et de leurs dimensions. Une couche de depthwise/pointwise se découpe en deux parties: la partie depthwise et la partie pointwise. La couche depthwise est comparable à une couche de convolution standard mais avec **uniquement** un filtre. Dans une couche de convolution depthwise/pointwise, il n'y a qu'une couche depthwise. La couche pointwise est comparable à une couche de convolution standard avec un filtre de dimension  $1 \times 1$  et s'applique sur la sortie *intermédiaire*<sup>71</sup> de la couche depthwise. Cette couche peut être cumulée N fois. Une illustration de cette couche est visible sur la Figure 29.

Étudions dorénavant les bénéfices de ce type d'approche. Supposons une entrée

---

<sup>71</sup>Avant l'étape d'addition par rapport à une convolution standard

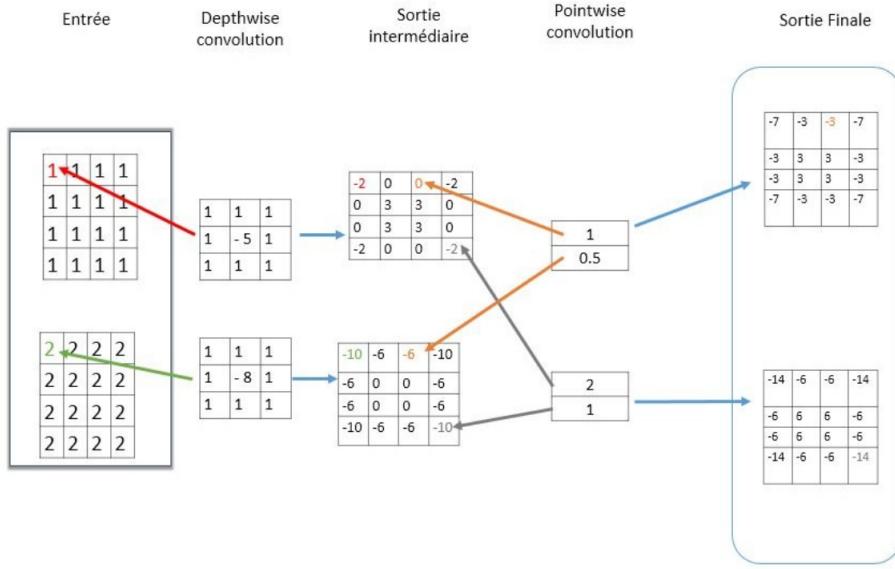


Figure 29: Exemple d'une couche Depthwise/Pointwise avec 2 filtres

(matrice carrée) de dimension  $X \times X \times M$  avec  $M$ , profondeur de l'entrée. On lui applique  $N$  filtres (composés de sous-filtres en accord avec la profondeur de la donnée d'entrée). On supposera que chaque filtre possède une dimension identique de taille  $K \times K$ . Le coût de calcul d'une couche de convolution standard est donc de  $(X \times X) * M * (K \times K) * N$ <sup>72</sup>.

Considérons maintenant une couche depthwise/poitnwise. On a alors:

$$\begin{aligned}
 D_{\text{depthwise/poitnwise}} &= D_{\text{depthwise}} + D_{\text{pointwise}} \\
 D_{\text{depthwise}} &= (X \times X) * M * (K \times K) * 1 \\
 D_{\text{pointwise}} &= (X \times X) * M * (1 \times 1) * N \\
 D_{\text{depthwise/poitnwise}} &= (X \times X) * M * (K \times K) * 1 + (X \times X) * M * (1 \times 1) * N
 \end{aligned}$$

La différence de coût de calcul est donc définie par:

$$\frac{(X \times X) * M * (K \times K) * 1 + (X \times X) * M * (1 \times 1) * N}{X \times X} = \frac{1}{N} + \frac{1}{K^2}$$

Ce résultat permet d'évaluer la pertinence de ce type de couche. En effet, on

<sup>72</sup>On suppose un stride de 1 et un 0-padding nécessaire pour conserver la même dimension entre les données d'entrée et la *feature map* de sortie

peut observer que plus le nombre de couche est importante ( $N$ ) et/ou la taille du filtre grand ( $K$ ), plus l'économie en temps de calcul est important. Cette économie est associée au temps de calcul machine et au temps d'apprentissage, le rapport étant identique pour les deux économies. Ainsi, pour les réseaux très profonds sur des supports de faible puissance, cette approche propose une solution performante.

Néanmoins, cette approche donne des résultats moins performants bien que ce défaut soit négligeable face au temps machine et d'apprentissage. L'utilisation ou non de ce type de couche relève des spécificités du problème à résoudre. Il est intéressant de noter que cette approche a été très sollicitée (et avec succès) sur les supports mobiles comme les smartphones.

#### 5.4 Conversion d'une couche Full-Connected en couche de convolution

Une couche Full-Connected, par un "jeu de couches", peut être remplacée par une approche exclusivement convolutive. Bien que cette approche puisse sembler *tricky*, elle est très utile afin d'optimiser certains réseaux qui exigent un calcul de prédiction rapide en limitant la redondance de calcul (algorithmes de *Tracking* par exemple).

Observons la Figure 30. En haut, un réseau convolutif classique avec deux couches Full-Connected en fin de réseau finalisées par un *Softmax*. Une sortie *Softmax* est caractérisée par une couche composée de neurones comportant autant de sortie que de classes discriminées suivie d'une activation par une fonction *Softmax*. Dans notre exemple, supposons que le réseau discrimine 4 classes, i.e 4 neurones sur la couche de sortie.

La première couche de Full-Connected est connectée à une sortie de dimension  $5*5*16$ . Un neurone possède donc  $5*5*16$  entrées distinctes. Observons le réseau entièrement convolutif (réseau du bas sur la Figure 30). La couche Full-Connected est remplacée par une couche convolutive de kernel  $5*5$  et de profondeur 400. La sortie de cette couche sera donc de dimension  $1*1*400$ . Cette couche de convolution est donc déterminée par 400 neurones où seul un neurone caractérise un filtre. Un kernel, par convention, s'applique sur toute la profondeur. Ainsi, chaque neurone de la couche de convolution reçoit  $5*5*16$  (hauteur\*largeur\*profondeur) valeurs pondérées par des poids distincts (et apprenants). Par conséquent, ces deux architectures sont mathématiquement identiques car le logit formé est issu d'une relation linéaire similaire et la couche présente autant de neurones.

Afin de réaliser la conversion, il faut donc créer une couche de convolution dont la dimension du kernel est égale à la dimension (hauteur et largeur) des features map en entrée. Le nombre de filtre de cette couche de convolution est égale au nombre de neurones de la couche Full-Connected.

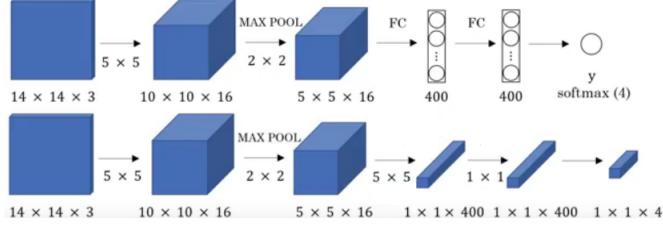


Figure 30: Exemple d'une conversion de couche Full-Connected vers une structure convective

## 5.5 Convolutional Neural Network (CNN) et Fully Convolutional Network (FCN)

Avec l'évolution de la Recherche, une famille d'architecture a été créée: *Fully Convolutional Network*[85] (FCN).

Contrairement à l'architecture CNN standard, FCN n'utilise pas de couche Full-Connected mais exclusivement des couches convolutives<sup>73</sup>. La prise de décision du réseau est donc assurée par des couches convolutives tout comme l'extraction d'attributs.

Un réseau FCN va donc essayer d'extraire les informations et de prendre une décision selon des données **spatialement indépendante** du fait de l'isolation réalisée par le fonctionnement de la convolution. Chaque *sous-décision* de la couche convective est basée sur l'information isolée par la taille du filtre. Au contraire, CNN aura tendance à considérer l'information dans sa **globalité** du fait de l'utilisation de couches Full-Connected qui exploite intégralement les données. Il n'y a donc pas de considération de *localité*. Ces dernières années, les réseaux FCN gagnent en popularité du fait de leur efficacité similaire et de leur plus grande vitesse d'apprentissage. De même, un FCN permet une plus grande robustesse face à des données de dimensions variables<sup>74</sup> tout en réalisant la conversion d'échelle plus rapidement.

## 5.6 Les modèles convolutifs de référence

### 5.6.1 AlexNet

AlexNet[54](2012) est une référence historique du Deep Learning. Il est le modèle précurseur des architectures convolutives (reposant sur l'architecture LeNet développée par Lecun) et a présenté des résultats remarquables en gagnant le 2012 ILSVRC<sup>75</sup> (ImageNet Large-Scale Visual Recognition Challenge) avec 10%

<sup>73</sup>D'où le nom Fully Convolutional Network

<sup>74</sup>Les couches Full-Connected sont dépendantes de la dimension de la donnée d'entrée

<sup>75</sup>Il s'agit d'une des compétitions - voire la compétition - la plus prestigieuse et compétitive de vision par ordinateur

d'erreur en moins que le 2nd. Il constitue un cas d'école pertinent pour apprendre les fondations d'un réseau convolutif (couche de convolution, max-pooling, dropout, ReLu...) et a été défini pour classer 1000 catégories au plus. Du fait de son ancienneté, il tend à devenir "obsolète" aujourd'hui.

### 5.6.2 ZF Net

ZF Net[112](2013) est un modèle basé sur AlexNet et vainqueur du 2013 ILSVRC. La différence majeure se situe sur la taille des filtres, notamment des premières couches. L'idée derrière cette réduction était de limiter un maximum la perte d'information associée à un filtre trop grand (trop généraliste) de la donnée initiale. De plus, il n'a été entraîné que sur 1.3 million d'image et non 15 millions comme AlexNet.

L'apport majeur de ce modèle (et du papier de recherche associé) est la méthode de visualisation DeConvNet (Deconvolutional Network). Cette méthode permet d'examiner les caractéristiques d'activation des différentes couches et d'offrir un aperçu de ce que "voit" et discrimine la couche observée. Cette architecture a permis de mieux comprendre le comportement interne des réseaux convolutifs et leurs critères de discrimination<sup>76</sup>. On le nomme DeConvNet du fait de l'inversion de la transformation réalisée. Au lieu de passer de "pixels à map feature", ce réseau passe de "map feature à pixels".

### 5.6.3 VGG Net

VGG Net[86](2014) est un finaliste (mais perdant) de 2014 ILSVRC. Ce réseau exploite une architecture "simple" mais profonde. En effet, ce modèle se limite à cumuler des couches de convolution utilisant des filtres 3\*3 - stride 1 et des couches de max-pooling 2\*2 - stride 2. La contribution principale de ce modèle a été de montrer que la profondeur est un composant critique de bonne performance. Néanmoins, c'est un modèle lourd (en ressources matérielles et paramètres) bien qu'il puisse être grandement simplifié en supprimant des couches de Full-connected sans une perte significative de performance. Il s'agit d'un modèle encore très utilisé aujourd'hui.

### 5.6.4 GoogleNet/Inception

GoogleNet[92](2014) est le vainqueur du 2014 ILSVRC. L'architecture de ce modèle est très novatrice (elle s'inspire, néanmoins, de l'approche Network in Network[60])<sup>77</sup>. Elle s'oppose au dogme du modèle séquentiel (succession de couches à flux unitaire) et propose une approche en "largeur" (en parallèle). Ce modèle a ainsi proposé une nouvelle forme de *block*: **l'Inception module**. Sa

<sup>76</sup>On est encore loin de comprendre parfaitement le fonctionnement des réseaux neuronaux. C'est toujours un sujet de recherche très intense et d'une nécessité prioritaire

<sup>77</sup>Nous n'aborderons pas ce papier. Il a une importance théorique mais n'apporte pas d'élément applicatif concret dans le cadre de ce cours

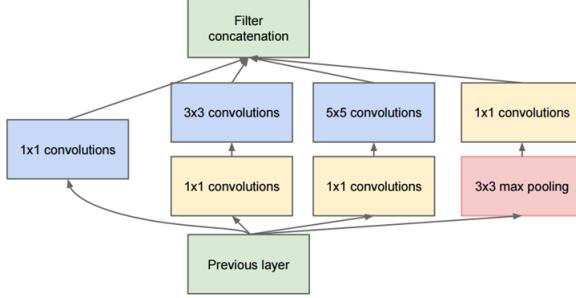


Figure 31: Architecture de l’Inception module

structure est représentée sur la Figure 31.

L’Inception module se démarque par la mise en parallèle de différentes couches de convolution appliquées sur une même source d’entrée dont les *feature map* produites sont **concaténées** à la fin du bloc. Cette mise en parallèle des couches permet de créer différents *flux* indépendants et différenciés les uns des autres. Chaque block peut avoir sa propre topologie, ce qui rend le modèle difficile à implémenter du fait de cette liberté. Évaluer la topologie d’un block est délicat et relève essentiellement de l’intuition. Il est important de relever la présence de convolution  $1 \times 1$ . Une approche naïve serait de les supprimer, leurs importances semblant être secondaires. Néanmoins, ce serait une erreur majeure. En effet, ces convolutions possèdent deux caractéristiques critiques dans le cadre de ce module:

- **Réduction de dimension:** Les convolutions  $1 \times 1$  possèdent la capacités de réguler la profondeur de la sortie d’une couche. Cette capacité est primordiale dans le cas de l’Inception module afin d’éviter une explosion inévitable de la dimension de sortie. Elle permettent ainsi de maîtriser la profondeur souhaitée d’une des branches du modules en sortie pour la concaténation (branche  $3 \times 3$  max pooling) ou de diminuer la profondeur de la donnée d’entrée sur les branches  $3 \times 3$  convolutions et  $5 \times 5$  convolutions. En effet, l’application de filtre volumineux est gourmand en temps machine. Réduire la dimension de la donnée à observer favorise la vélocité du modèle. Les convolutions  $1 \times 1$  permettent ainsi de réguler la dimension des données internes au module en maîtrisant les dimensions de sortie et en optimisant les dimensions de la données d’entrée pour favoriser un bon rapport performance/temps.
- **Non-linéarité:** Les convolutions  $1 \times 1$  sont suivies d’une activation ReLu. De ce fait, elles permettent la mise en place de non-linéarité dans le module. Les couches de convolution étant linéaires dans leurs formes fondamentales, cet ajout ne peut être que bénéfique.

Cette architecture a eu plusieurs améliorations. La plus moderne est l’Inception-v4[91] qui propose une approche liant l’architecture Inception et Resnet.

### 5.6.5 Xception

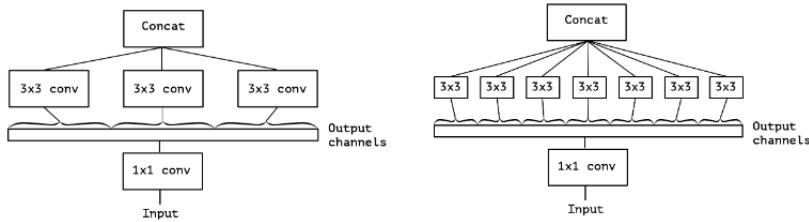
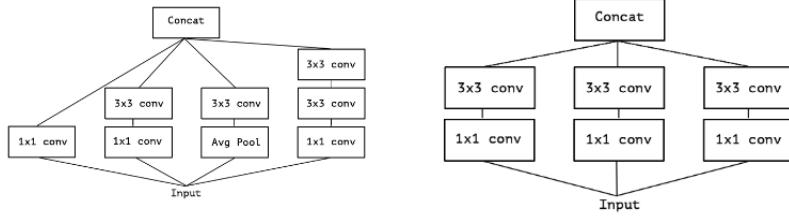
Le modèle Xception[11](2016) a été inventé par François Chollet, chercheur de Google à l’origine du framework *Keras*. Xception propose une approche *extrême* de l’approche *Inception* d’où son appellation.

Afin de comprendre l’idée proposée par François Chollet, revenons sur le modèle Inception standard. Sur la Figure 32, nous pouvons observer deux block Inception. A gauche, un block arbitraire (topologie variable) et à droite, un block simplifié avec une structure uniforme. Supposons un block de N flux. Par un jeu de manipulation, nous pouvons remplacer les N couches de convolution  $1 \times 1$  indépendantes en une couche  $1 \times 1$  unique de dimension  $N \times P$  avec P profondeur des sorties des couches  $1 \times 1$  indépendantes<sup>78</sup>. Les *feature map* de la couche  $1 \times 1$  unitaire seront alors divisés en N parties sans chevauchement et alimenteront les différents flux de manière indépendante. Cette approche peut être poussée à *l’extrême* en associant un flux à une unique *feature map*. Une représentation graphique est visible sur la Figure 33. Le postulat du modèle Xception est une hypothèse plus forte que l’hypothèse d’Inception: il suppose que les corrélations inter-channels (profondeur) et les corrélations spatiales (largeur et longueur d’une *feature map*) peuvent être complètement traitées séparément et non par sous-groupes.

Cette approche isolante est très similaire à une convolution Depthwise/Pointwise. Une différence majeure existe: une couche de Depthwise/Pointwise réalise une convolution standard pour créer les *feature map* intermédiaires qui seront alors traitées par des convolutions  $1 \times 1$ . *Extreme Inception* réalise exactement l’inverse. Les créateurs du modèle ne justifie pas concrètement le réel impact de cette inversion si ce n’est par leur intuition. De plus, dans un block Inception, une activation ReLu est appliquée entre les différentes couches afin d’appliquer de la non-linéarité, ce n’est pas le cas avec les couches Depthwise/Pointwise. Cette spécificité permet de mettre en avant un résultat remarquable. Alors que la non-linéarité est bénéfique dans le cas du block Inception standard[94], elle semble être néfaste pour les blocks Extrem Inception avec Depthwise/Pointwise[11]. Ce résultat a été vérifié expérimentalement. Il est difficile de justifier clairement ce résultat mais l’idée principale repose sur la différence de profondeur. En effet, dans le block Inception standard, la profondeur de la données d’entrée dans un flux est profonde alors que dans un block Extrem Inception, la *feature map* est unitaire. L’hypothèse est donc qu’appliquer une régularisation sur une donnée peu profonde provoque une perte d’informations trop importante.

---

<sup>78</sup>On supposera les sorties uniformes



La spécificité de la couche Depthwise/Pointwise permet ainsi de créer un réseau très simple d'implémentation. En effet, un block Extreme Inception n'est donc rien d'autre qu'une couche de Depthwise/Pointwise. L'architecture générale est standard et ne présente pas de particularité notable si ce n'est l'utilisation des *résidus* (voir Section 5.6.6 pour plus de détails sur cette notion).

### 5.6.6 Microsoft Resnet

Resnet[30](2015) est le vainqueur de 2015 ILSVRC. Son taux d'erreur est exceptionnel (3,6%). Un homme, en fonction de son expertise, obtient un taux d'erreur d'environ 5 à 10%. Cette architecture est donc la première à faire significativement mieux que l'homme dans le cadre de cette compétition. Cette architecture repose sur le **Residual Block**. Une illustration est visible sur la figure 34.

Un *Residual Block* est caractérisé par la non-perte de la donnée d'entrée. Ainsi, la sortie de ce block calcule un changement de faible ampleur (un "Delta") afin d'obtenir une version légèrement altérée de la donnée d'entrée. Pour conserver la donnée d'entrée, une concaténation est réalisée à la sortie du block où la sortie des couches internes est **additionnée**<sup>79</sup> à la donnée d'entrée<sup>80</sup>. Cette

<sup>79</sup>Au sens strict du terme, non concatenée

<sup>80</sup>En cas de dimension différente entre l'entrée et la sortie des couches internes du bloc, du

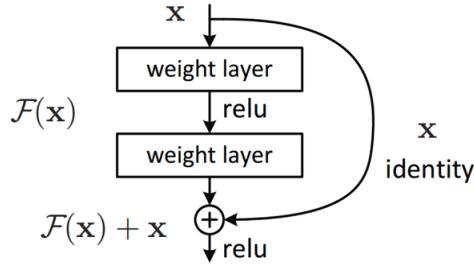


Figure 34: Architecture du Residual Block

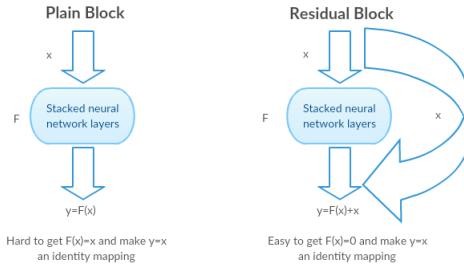


Figure 35: Intuition sur la notion de Résidus

architecture favorise un bon apprentissage en facilitant la propagation d'un gradient "valide", notamment en luttant contre le problème de *vanishing gradient*. L'idée derrière la notion de *Résidus* est qu'il est plus facile d'approximer une fonction qui nullifie les résidus plutôt que d'approximer une fonction identité (Figure 35) . Néanmoins, son efficacité théorique n'est pas démontrée malgré une efficacité empirique et expérimentale incontestable.

**Important:** Les architectures *state-of-the-art* reposent essentiellement sur ce type d'architecture<sup>81</sup>. Il est souvent pertinent de considérer ce type de réseau convolutif lorsqu'un choix par défaut s'impose.

### 5.6.7 ResNet et améliorations

Du fait de l'efficacité des réseaux Resnet, la recherche est active autour de cette architecture. Plusieurs améliorations ont été proposées:

---

0-padding et/ou des convolutions 1\*1 sont utilisées

<sup>81</sup>Plusieurs architectures peuvent être cumulées !

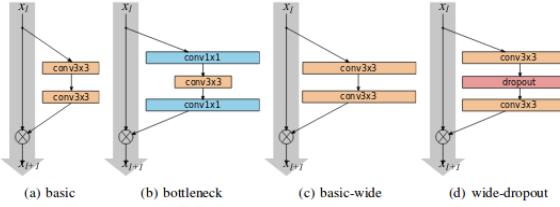


Figure 36: Architecture d'un Wide Residual block (les étapes de Batch normalization et de ReLu ne sont pas affichées) - la largeur des rectangles représentant les couches déterminent graphiquement le nombre de filtres employés pour la couche associée

#### 5.6.7.1 Wide Residual Network

Wide Residual Network[110](2016) propose une architecture qui s'étend en "largeur"<sup>82</sup> et non uniquement en profondeur comme l'approche initiale des ResNets. La démarche principale consiste augmenter le nombre de filtres par couche ou planter des couches spécifiques (dropout, 1\*1 convolution,...). Ainsi, cette architecture crée plus de *feature map* en interne et possède un plus fort pouvoir de discrimination. Néanmoins, il peut y avoir une hausse du nombre de paramètres donc un temps de calculs et d'apprentissage supérieurs. Pour s'opposer à ce problème, la profondeur est minorée par rapport à la largeur. De plus, les capacités de calculs sont optimisées avec les approches en largeur car le calcul des différentes *feature map* au sein d'une même couche est fortement parallélisable, ce qui favorise la vélocité du modèle. Cette architecture propose aussi l'ajout de DropOut ou de convolutions 1\*1 à l'intérieur des blocs. La plus-value de cet article est de montrer qu'un réseau ResNet ne doit pas être que profond mais favoriser son expansion en largeur aussi. Une illustration de cette architecture est représentée sur la Figure 36.

#### 5.6.7.2 Aggregated Residual Transformations for Deep Neural Networks (ResNext)

Le réseau ResNext[103](2016) est le modèle arrivé 2nd au 2016 ILSVRC. Cette architecture propose de lier les caractéristiques des modèles ResNet et Inception. La principale caractéristique de ResNext repose sur son expansion en largeur avec l'ajout de flux internes dans un block. Sa différentiation majeure avec l'approche d'Inception est l'utilisation d'une topologie de bloc identique pour chaque bloc. La différenciation des blocks se fait au niveau du facteur de profondeur qui détermine le nombre de flux parallèles au sein d'un block.

---

<sup>82</sup>Ne pas confondre ! Il y a une augmentation sur un unique flux de couches (ajout de filtres ou d'un DropOut par exemple) uniquement. L'augmentation se fait en largeur car elle ajoute des entités sur une couche ciblée au sein d'un même flux, non en largeur avec l'ajout de flux parallèles comme les modèles Inception !

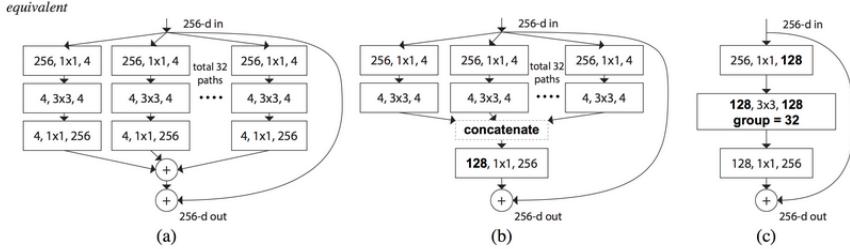


Figure 37: Différentes représentations de l'architecture d'un block du réseau ResNext: 1) Approche ResNet, 2) Approche Inception, 3) Approche par convolution groupée

L'architecture Resnext propose trois formes de blocks équivalents pour structurer son réseau.

- **Approche ResNet:** Avec cette approche, chaque flux réalise une harmonisation de la dimension indépendamment des autres flux via une convolution  $1 \times 1$ . Les *feature map* issues des différents flux du block sont additionnées. La sortie est alors additionnée avec l'entrée du block.
- **Approche Inception:** Avec cette approche, l'harmonisation de la dimension est mutualisée. Ainsi, l'ensemble des *feature map* issues des flux sont concaténées sans application de convolution  $1 \times 1$  et produisent une sortie intermédiaire. L'application de la convolution  $1 \times 1$  est réalisée sur la sortie intermédiaire et les *feature map* obtenues sont additionnées à l'entrée du block.
- **Approche par convolution groupée:** Cette approche repose sur l'implémentation du réseau AlexNet qui a groupé ses couches afin de paralléliser son réseau sur différents GPU. Resnext propose cette méthode en considérant un flux comme un groupe. Ils peuvent donc être partagés sur différents GPU et favoriser la vélocité du modèle. Cette représentation est essentiellement théorique et peu exploitée dans les faits.

Une représentation graphique des différentes architectures du block ResNext est visible sur la Figure 37.

#### 5.6.7.3 Densely Connected Convolutionnal Networks (DenseNet)

Densely Connected Convolutionnal Networks[34](2016) généralise la transmission de l'entrée des blocs ResNet en projetant l'entrée d'une convolution sur l'ensemble des convolutions suivantes et non juste le suivant direct. Ainsi, la  $n^{i\text{ème}}$  couche possédera une entrée issue de  $n$  sources. De même, la  $n^{i\text{ème}}$  couche transmettra sa sortie aux  $(N-n)$  couches suivantes avec  $N$ , nombre total

de couche. Il y a donc  $\frac{N(N+1)}{2}$ <sup>83</sup> connexions dans l'ensemble du réseaux issues des divers blocs au lieu de N d'où le nom de *Densely connected*. De plus, l'entrée est **concaténée** et non additionnée comme dans un bloc ResNet standard. La concaténation implique une augmentation constante de la profondeur de l'entrée transférée à chaque bloc. L'ajout de convolution 1\*1 et de pooling est donc une nécessité pour éviter l'explosion du nombre de données en entrée. Veuillez vous référer à l'article [34] pour plus de détails sur cette régulation.

Le réseau DenseNet, visible sur la Figure 39, est composé de plusieurs Dense block séparés par un block de transition formé d'une convolution 1\*1 et d'un Pooling afin de redimensionner les *feature map*. Ce redimensionnement pose problème car les *feature map* des block précédents ne sont plus de même dimension que les block qui suivent. De ce fait, la propagation se limite à un Dense block et non à l'ensemble du réseau.

La conversion de l'addition vers la concaténation apporte une plus-value importante. La concaténation préserve la structure de la *feature map* peu importe l'éloignement du bloc par rapport au bloc considéré. Cette différence permet de s'opposer au problème de corruption de l'information transférée ou de son effacement qui augmente alors que le nombre de *feature map* additionnées augmente linéairement. De plus, le modèle peut apprendre une combinaison optimal entre les features map concaténées, ce qui permet une meilleure interprétation de l'information utile.

Bien que paradoxale, cette approche permet de limiter le nombre de paramètres nécessaire pour créer le modèle. Dans un réseau neuronal, chaque couche lit une entrée donnée par la couche précédente associée et écrit à la couche suivante. La couche actuelle modifie donc l'état mais transmet aussi l'information importante à conserver. Cette information est conservée, dans les réseaux ResNet, par addition avec l'identité de l'entrée. Cependant, il a été montré que certaines couches contribuent peu voire produisent de l'information redondante (il se peut qu'elles redéfinissent une information perdue déjà calculée précédemment dans le modèle) et donc, peuvent être supprimées. Afin de considérer cette particularité, la structure Densely connected propose une meilleure mémoire de la donnée utile et des modifications réalisées en projetant l'information sur l'ensemble des blocs et non uniquement le suivant. Cette propagation (et conservation) de l'information permet de limiter grandement le nombre de couches/blocs et ainsi de limiter le nombre de paramètres, rendant le modèle plus léger et rapide à apprendre. L'intuition derrière cette idée peut être contre-intuitive. Pour une explication détaillée, veuillez vous référer à l'article associé [34]. La structure d'un réseau Densely Connected est visible sur la Figure 38.

---

<sup>83</sup>Équivalent à la somme de 1 à n

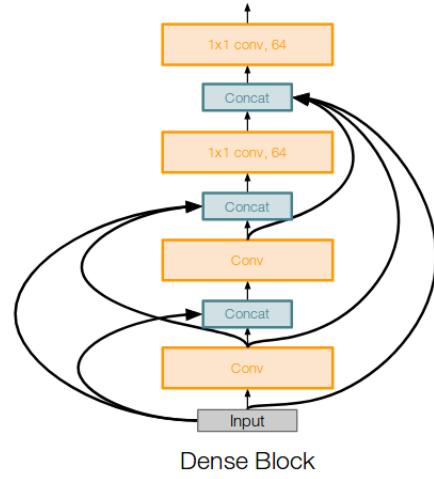


Figure 38: Architecture d'un réseau Densely Connected

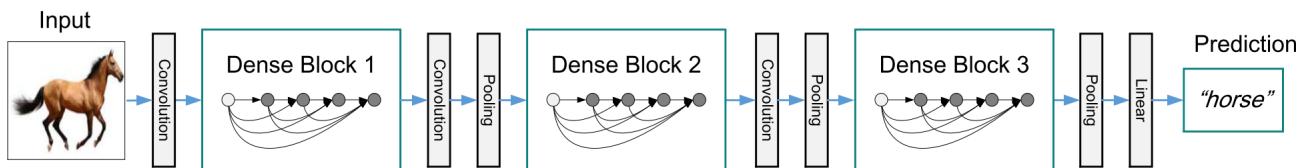


Figure 2. A deep DenseNet with three dense blocks. The layers between two adjacent blocks are referred to as transition layers and change feature map sizes via convolution and pooling.

Figure 39: Réseau DenseNet

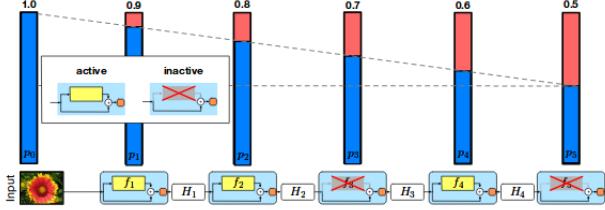


Figure 40: Stochastic Depth routine

#### 5.6.7.4 Stochastic Depth

Stochastic Depth[35](2016) met en relation le DropOut et les spécificités des *Réidual block* des réseaux résiduels. Afin de limiter l’inter-dépendance des couches du réseau, les blocks sont conservés (sinon ils sont équivalents à une couche *identité*) selon une probabilité  $p_l$ .  $p_l$  est une probabilité constante (peu optimale) ou linéaire dégressive telle que  $p_l = 1 - \frac{l}{L} * (1 - p_L)$  (par défaut,  $p_L$  est égal à 0.5 et correspond à la valeur limite du dernier block considéré dans le réseau) avec  $L$ , nombre total de block et  $l$ , le block ciblé. Comme pour le DropOut, durant la phase de test, chaque sortie de residual block (avant union avec la valeur d’entrée du bloc) est active et pondérée selon la valeur du pourcentage d’activation associée. Un exemple illustratif est visible sur la Figure 40.

#### 5.6.7.5 Residual Networks of Residual Networks: Multilevel Residual Networks

Residual Networks of Residual Networks[113](2016) propose une approche basée sur la structure de groupe. Ainsi, le réseau est composé d’une succession de groupes composés de block à la structure similaire. Le créateur du modèle a utilisé des convolutions  $3*3$  uniquement de profondeur variable appartenant à  $[16, 32, 64]$ . Trois groupes sont donc formés en fonction de la largeur des couches. Il n’y a pas d’intersection de groupe, i.e chaque groupe est unique et les groupes sont homogènes. Supposons un réseau avec  $N$  block alors un groupe possèdera  $\frac{N}{3}$  block. Ce réseau se différencie du ResNet en ajoutant une connexion inter-block pour chaque groupe. Ainsi, les *feature map* en entrée du premier block d’un groupe seront additionnées aux *feature map* de sortie de ce groupe. Un exemple de ce réseau est visible sur la Figure 41.

La notion de groupe est variable selon le niveau de coupure  $P$  choisi. Ainsi, pour  $P=1$ , on considère un groupe uniquement. Le réseau est donc semblable à un ResNet standard (pas de liaison inter-groupe). Pour  $P=2$ , on considère 2 groupes. Ainsi, la connexion inter-block est unique et relie l’entrée du premier bloc avec la sortie du dernier bloc du réseau. De même, pour  $P=3$ , on considère 3 niveaux de groupe. De ce fait, une connexion inter-block est faite entre le premier et dernier block de même largeur de couche (16,32 ou 64 dans le cas

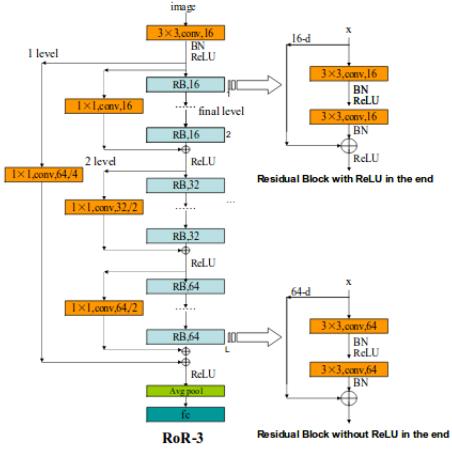


Figure 41: Exemple d'un réseau RoR de niveau 3

présent).

L'architecture du modèle est généraliste et peut être associée à des modèles existants tels que Wide Residual Network par exemple. L'idée portée par l'auteur est de ne pas se limiter à la connexion entre un block et son successeur mais réaliser une connexion entre l'entrée et la sortie d'un *Super-Block* composé d'un ensemble de block de même nature. Cette approche propose ainsi une approche dépendante de l'architecture de groupe de réseau alors que l'approche Densely conenected et Sparsely Connected considère le block de manière unitaire et isolée.

#### 5.6.7.6 Sparsely Connected Convolutional Networks (SparseNet)

Sparsely Connected Convolutional Networks[118](2018) repose sur les fondations du réseau DenseNet à la différence que les sorties des blocks ne sont pas transmises à l'ensemble des couches suivantes mais en fonction d'un offset exponentiel. En effet, pour un block  $B_n$ , seules les sorties des couches  $B_{n-1}, B_{n-2}, B_{n-4}, B_{n-8}, \dots, B_{n-2^k}$  avec  $k$ , plus grand entier non négatif tel que  $2^k < l$ , sont considérées. Une comparaison entre SparseNet et DenseNet est visible sur la Figure 42.

L'approche par concaténation proposée par DenseNet a permis une amélioration significative du modèle ResNet en corrigeant le défaut de la méthode additive. Néanmoins, cette approche pose un problème du fait de l'explosion du nombre de connexions qui est de complexité  $O(N^2)$ . Cette augmentation de paramètres présente un risque d'overfitting et une explosion de la consommation mémoire du réseau lorsqu'il est profond. De plus, diverses expériences ont montré que l'utilisation de l'information transmise par les connexions inter-block est mal exploitée par le réseau avec une part importante de *feature map* transférées

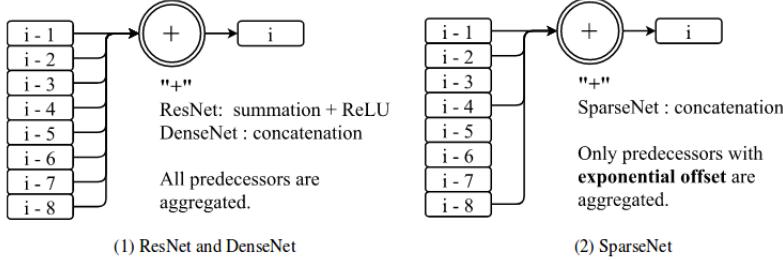


Figure 42: Différence entre un block issu de ResNet, DenseNet et SparseNet

sans réel pouvoir explicatif. Le réseau DenseNet est donc trop dense. L'ajout d'un offset exponentiel d'ordre 2 permet ainsi de limiter ce défaut en limitant la complexité à  $O(\log(n))^{84}$  tout en conservant l'information utile portée par les différentes connexions inter-block. L'hypothèse est faite qu'il est préférable de privilégier l'information portée par les block adjacents que les block très éloignés.

#### 5.6.7.7 Résumé des différentes approches appliquées aux ResNet

Ci-dessous, un tableau récapitulatif de l'ensemble des améliorations étudiées concernant les réseaux ResNet. Certaines de ces méthodes sont cumulatives. Approfondir leurs interactions est une piste de recherche intéressante et pertinente.

Méthodes	Spécificités
Wide Residual	Elargissement du flux d'un bloc Utilisation de couches spécifiques (DropOut...)
Aggregated Residual Transformations	Ajout de flux dans un block
Densely Connected	Propagation de la sortie sur l'ensemble des couches suivantes Concaténation de la sortie du block avec son entrée
Residual Networks of Residual Networks	Propagation de la sortie selon les groupes de block
Sparsely Connected	Propagation de la sortie selon un offset exponentiel Concaténation de la sortie du block avec son entrée
Stochastic Depth	Non-utilisation de block durant l'apprentissage (probabiliste)
Squeeze-and-Excitation	Pondération des <i>feature map</i>

#### 5.6.8 FractalNet: Ultra-Deep Neural Networks without Residuals

Le modèle FractalNet[55](2016) est une approche qui s'émancipe de l'architecture des ResNet (ou dérivés) et se présente comme une alternative viable. Ce réseau

<sup>84</sup>On peut grossièrement considérer que le nombre de connexions inter-block total est borné du fait de la faible croissance de la fonction logarithme

démontre que, bien que très efficace, l'approche par résidu n'est pas la seule approche compétitive et performante. Néanmoins, FractalNet est un réseau récent encore méconnu. Il est donc difficile de juger pleinement de ses capacités. La structure du réseau Fractalnet est comparable au comportement d'une fractale, i.e la structure du réseau suit un modèle similaire peu importe le sous-réseau observé. La structure fondamentale du ce réseau est explicitée par la Figure 43.

FractalNet exploite une alternative au DropOut afin de régulariser son réseau. Le processus, appelé **Drop-path** se divise en deux parties:

- **Local:** L'étape Local désactive des flux (ou entrées de couche *join*) aléatoirement selon une probabilité fixée. Il y a une garantie de l'existence d'au moins un flux pour chaque couche *join*.
- **Global:** L'étape Global sélectionne un unique path pour tout le réseau et désactive tous les autres flux. Cette approche permet de limiter les dépendances entre flux, de masquer l'origine du flux et ainsi, de bloquer le comportement du réseau qui viserait à considérer un flux comme un résidu ou un terme correctif.

La couche *join* peut sembler similaire à l'additivité employée dans un réseau ResNet. Cependant, il existe des différences critiques non négligeables:

- **Non différenciation de la source:** Contrairement au réseau ResNet, le réseau FractalNet ne différencie pas le flux associé à un résidu et le flux associé à la couche interne du block. Cette non-différentiation est permise grâce au transfert direct sans transformation préalable à la couche de convolution suivante.
- **Régularisation par Drop-path:** Ce procédé permet de forcer chaque flux à être *viable*. De ce fait, la régularisation favorise un apprentissage qui punit les flux qui évoluent vers un comportement de *résidu*.
- **Sous-réseau unitaire et performant:** Le Drop-path, du fait de son étape *Global* favorise les performances de sous-réseaux unitaires, i.e qui n'exploite pas les caractéristiques de la couche *join*. En effet, l'étape *Global* force les couches *join* à n'avoir qu'une seule entrée. Ainsi, ces sous-réseaux sont fonctionnels et produisent un signal qui ne devrait pas "recevoir" de résidu pour être efficace.

#### 5.6.9 Classement des architectures standards

Le graphique présent sur la Figure 44 résume les caractéristiques des différentes architectures vues précédemment. Ainsi, en résumé, nous pouvons observer que Inception-v4 fait office d'état de l'art actuellement. Les architectures Inception et Resnet offrent le meilleur rapport efficacité/calcul/mémoire. On peut

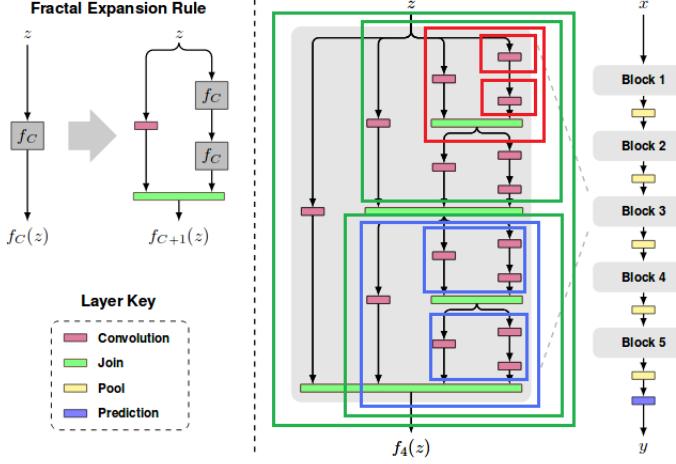


Figure 43: Construction d'un bloc du réseau FractalNet

noter que GoogleNet est significativement moins précis mais nécessite un nombre significativement plus faible d'opérations. Il peut donc être intéressant dans certains contextes. Les modèles VGG sont très lourds, lents et avec une efficacité en déclin. Tout comme AlexNet, ils tendent à devenir obsolètes. Pour une étude complète et détaillée des différents modèles de Deep Learning, veuillez vous référer à l'article [8].

### 5.6.10 Réseaux de neurones compressés

Les modèles étudiés précédemment sont pour la plupart gourmands et lourds. Ils posent un problème de mémoire et de performance sur des supports de faible capacité comme l'embarqué ou les appareils nomades tels que les smartphones. Afin de répondre à cette problématique, une optimisation du temps de calculs et du nombre de paramètres sont nécessaires tout en limitant la perte de la qualité prédictive du réseau. Ce type de réseaux dits *compressés* ont une importance élevée du fait de leurs capacités de commercialisation notamment à travers les applications des supports mobiles (téléphones, tablettes, domotiques...)

#### 5.6.10.1 SqueezeNet

Le réseau SqueezeNet[38](2016) est un réseau développé pour être supporté par les supports les plus *faibles*. Sa structure cherche donc à limiter le nombre de paramètres et son impact mémoire. Ainsi, par rapport à AlexNet (qui lui sert de référence), SqueezeNet est 50x plus léger et son poids peut être 510x plus faible avec application de méthode de compression de réseau, notamment Deep Compression (voir Section 4.11). Sans compression, ce réseau pèse 4.8MB et après compression, 0.47. Cette taille minime lui permet d'être implémenter sur

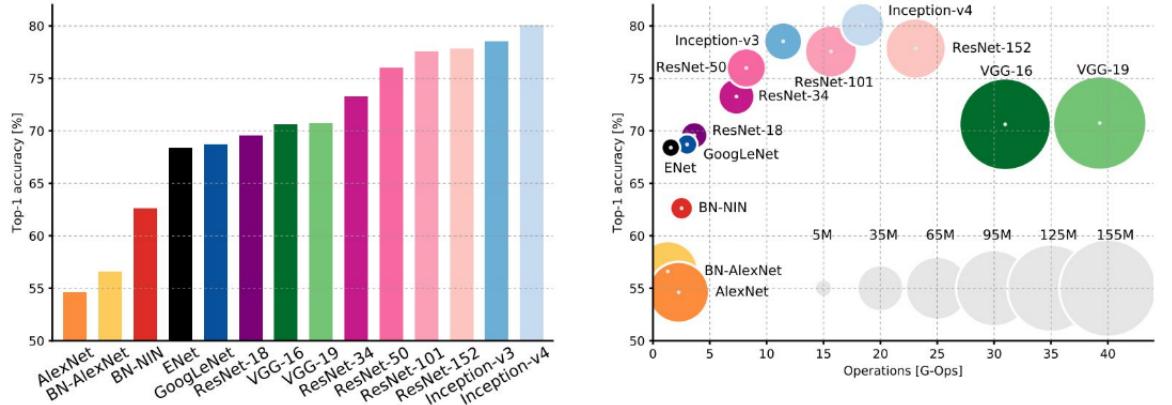


Figure 44: Analyse des réseaux de Deep Learning (2017) - Ce graphique a été créée par Alfredo Canziani, Adam Paszke et Eugenio Culurciello

de nombreuses structures de faible capacité bien que la précision du réseau soit significativement inférieure aux autres réseaux (bien plus volumineux). Il reste cependant satisfaisant pour des tâches de faible complexité ou sans nécessité de précision de haute performance.

La structure de ce réseau repose sur trois principes:

- **Utilisation en majorité de filtre 1\*1 à la place de 3\*3** afin de limiter le nombre de paramètres à apprendre et le nombre de calculs nécessaires.
- **Diminution du nombre de channels entrants dans une couche de convolution.** Cette diminution est réalisée grâce à une couche *squeeze*<sup>85</sup>. Afin de ne pas trop perturber la précision du modèle en supprimant en excès les filtres 3\*3, limiter le nombre de channels à analyser est une alternative efficace. En effet, le nombre de calculs à réaliser pour une couche de convolution 3\*3 est:  $nbr_{inputChannel} * nbr_{filtre} * (3 * 3)$ .
- **Conservation de la dimension de l'entrée** afin de favoriser une bonne précision du modèle. En effet, les principes précédents favorisent une réduction du poids du réseau au détriment de sa performance. Il est donc nécessaire de lutter contre une trop grande détérioration de la précision. Pour cela, la dimension des *feature map* est faiblement modifiée jusqu'en fin de réseau contrairement à la plupart des réseaux plus volumineux qui réalisent de nombreuses transformations internes, notamment via l'application d'un stride supérieur à 1 ou des approches par *Pooling*. Cette idée relève d'une intuition des créateurs du réseau et n'a pas de preuve véritablement démontrée.

<sup>85</sup>Nous verrons par la suite la configuration de cette couche

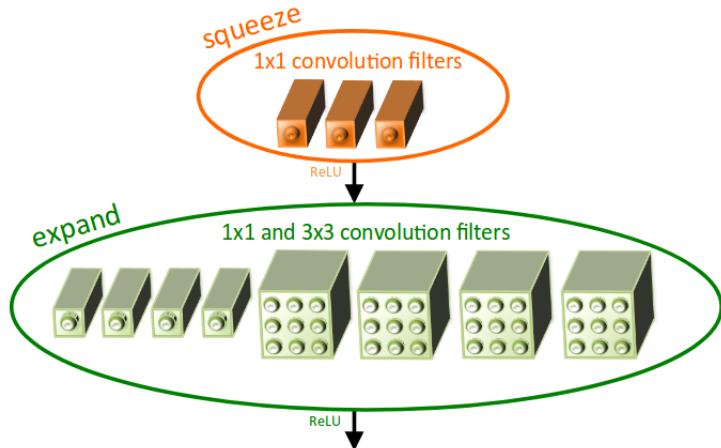


Figure 45: Architecture du Fire Module

Afin de respecter les deux premiers principes, un nouveau type de block est créé: le *Fire Module*. Ce module est séparé en deux parties: la partie *Squeeze* et la partie *expand*. La partie Squeeze est uniquement composée de filtres  $1 \times 1$  (Principe 1), la partie Expand analyse les *feature map* issues de la partie Squeeze et est composée de filtres  $1 \times 1$  et  $3 \times 3$ . Le nombre de filtre  $1 \times 1$  dans la partie Squeeze est inférieur au nombre de filtres ( $1 \times 1$  et  $3 \times 3$ ) dans la couche Expand, ce qui permet de maîtriser la profondeur des entrées et de conserver un nombre de dimension restreint sans négliger le nombre de filtre (Principe 2). L'absence de *Pooling* et de stride permet de garder les mêmes dimension de *feature map* (hauteur et largeur). Le block possède de nombreux hyperparamètres afin de définir la répartition des filtres et l'évolution de la configuration du filtre au fil du réseau. Pour plus de détails, veuillez consulter l'article[38]. Une illustration du block est visible sur la Figure 45.

L'architecture du réseau repose sur une succession de Fire Module régulièrement régulée par des couches de pooling inter-block. Il est intéressant de noter que le nombre de *feature map* de sortie augmente régulièrement jusqu'à la fin du réseau, ce qui traduit une augmentation du nombre de filtres. De plus, ce réseau possède une architecture standard ou résiduelle. Dans le cas d'une architecture résiduelle, deux approches sont possibles. La première consiste à réaliser une connexion interblock à chaque fois que la dimension de l'entrée est identique à la dimension de sortie (*simple bypass*). La seconde crée une connexion interblock pour chaque bloc. En cas de différence de dimensions, une couche de convolution  $1 \times 1$  mettra à l'échelle la profondeur du résidu du block considéré (*complex bypass*). L'ajout de la structure résiduelle sert à palier la perte d'informations associée à la couche *Squeeze* des blocs.

### 5.6.10.2 MobileNet

L'une des architectures les plus populaires est MobileNet[31](2017). Ce modèle exploite les convolutions Depthwise/Pointwise afin de limiter son impact mémoire et accélérer sa vitesse. Les filtres des Depthwise sont de taille 3\*3 uniquement, ce qui est un standard avec le filtre 5\*5. Cette dimension de filtre reste néanmoins l'un voire le meilleur rapport temps/précision/mémoire pour un modèle qui se veut léger et rapide.

L'architecture se veut modulable et adaptatif. Pour cela, deux hyperparamètres ont été instaurés: le **multiplicateur de largeur** et le **multiplicateur de résolution**. Ces paramètres de modifier la structure du réseau tout en conservant son intégrité et sa structure, uniforme. Il est donc possible de paramétriser aisément sa profondeur afin de l'adapter aux différentes capacités des supports d'application.

- **Multiplicateur de largeur:** Ce critère (noté  $\alpha$ ) joue sur la profondeur des entrées et des sorties des couches de convolutions en adaptant leurs dimensions selon un coefficient. Ainsi, en reprenant les résultats obtenus dans la Section 5.3.8, nous obtenons:

$$D_{depthwise/potnwise,\alpha} = (X*X)*\alpha M*(K*K)*1 + (X*X)*\alpha M*(1*1)*\alpha N$$

En pratique, cet hyperparamètre est implémenté de manière à limiter le nombre de *feature map* de sortie d'une couche de convolution. En effet, cette sortie sera l'entrée de la couche suivante. Il est donc évident que si la dimension d'une sortie est minorée, la dimension de l'entrée de la couche suivante le sera aussi.

- **Multiplicateur de résolution:** Ce critère (noté  $\rho$ ) modifie la dimension de l'image d'entrée, réduisant ainsi la représentation interne dans chaque couche du réseau. Nous obtenons ainsi:

$$D_{depthwise/pointwise,\rho} = (\rho X*\rho X)*M*(K*K)*1 + (\rho X*\rho X)*M*(1*1)*N$$

Ce critère est peu explicité dans l'article publié et son implémentation douteuse. Il est possible qu'il soit théorique et au final, non représenté concrètement.

Avec l'application des deux coefficients, l'équation du coût de ce réseau s'exprime donc sous la forme:

$$D_{depthwise/potnwise,\alpha,\rho} = (\rho X*\rho X)*\alpha M*(K*K)*1 + (\rho X*\rho X)*\alpha M*(1*1)*\alpha N$$

### 5.6.10.3 Modèles expérimentaux récents

Afin d'approfondir l'état de l'art, voici une liste de modèles récents prometteurs qu'il peut être intéressant d'étudier:

- ShuffleNet[114] (Dec 2017): Architecture pour support de faible puissance (smartphone par exemple). Ses résultats semblent meilleurs que MobileNet pour un coût machine plus faible.
- EffNet[18] (En cours d'écriture - Mars 2018): Architecture pour support de faible puissance (smartphone par exemple). Plus récent que MobileNet et ShuffleNet, il cherche à corriger leurs défauts. En cours d'écriture, il n'est pas possible de juger les résultats de cette architecture actuellement mais les résultats intermédiaires sont prometteurs.

#### 5.6.11 Les jeux de données de référence

Le Deep Learning nécessitant de nombreuses données, la création de jeux de données de qualité a été au cœur de la Recherche. Bien que ce soit encore une thématique en cours d'étude, de nombreux jeux de données ont été créés pour effectuer des *benchmarks* compétitifs, répondre à une problématique ou tout simplement, faciliter le développement des algorithmes de Deep Learning en soulageant la difficulté de création d'un jeu de données.

L'analyse d'image étant le secteur historique principal du Deep Learning, de nombreux jeux de données ont été créés. Il est intéressant d'en connaître les principaux afin de pouvoir les utiliser dans des projets. Dans cette section, nous allons considérer les jeux de données concernant l'analyse d'image uniquement.

- **MNIST: handwritten digits**[57]: Ce jeu de données est historique et a accompagné l'ascension des réseaux convolutifs. Il est constitué de chiffres entre 0 et 9 écrit à la main représentée en noir et blanc<sup>86</sup>. Souvent utilisé pour évaluer la pertinence d'un modèle, il tend à devenir obsolète car trop "facile" à résoudre. Les meilleurs modèles actuels dépassent 99% sur ce jeu de données.
- **CIFAR10 / CIFAR100**[52][53]: Il s'agit d'un jeu de données généraliste composé d'image 32\*32 réparties en 10/100 classes. Il n'est pas très utilisé mais permet d'évaluer les performances d'un modèle de manière préventive avant d'exploiter des jeux de données plus populaires et plus "exigeants".
- **Pascal VOC**[16]: Ce jeu de données est relativement ancien et n'est plus mis à jour. Il est rarement utilisé dans le cadre d'apprentissage pour de la classification d'image mais très populaire pour la détection d'objet.
- **ImageNet**[14]: C'est l'un des jeu de données les plus riches actuellement. Avec plus de 10 millions d'image pour 1000 catégories, il offre une richesse d'apprentissage de premier ordre. Une partie de ce jeu de données a été convertie en *bounding boxes*. Avec de nombreuses API disponibles, ce jeu de données a l'ambition de devenir une voire la référence pour l'apprentissage générique. Il est associé à une compétition annuelle

---

<sup>86</sup>Des variantes ont été créées en niveau de gris

(ILSVRC) qui est considérée comme l'une voire la compétition la plus prestigieuse en analyse d'image actuellement.

- **MS COCO**[63]: Il s'agit d'un jeu de données générique (2.5 millions d'image pour 91 classes) associé à une compétition d'analyse d'image (COCO). Étant légèrement dans l'ombre d'ImageNet en terme de popularité, ce jeu de données s'illustre avec ses images annotées (*captioning*).

Ces jeux de données sont les références *généralistes*. Il existe de nombreux autres jeux de données généralistes tout comme des spécialisés, notamment d'images de visage (pour la reconnaissance faciale), des images aériennes/satellites ou encore des images issues de la *Physique* (astrophysique/cosmologie, physique fondamentale, mécanique...) et de la Médecine/Biologie (imagerie médicale) par exemple.

## 6 Encoder-Decoder

### 6.1 Généralités

Les *Encoder-Decoder* sont une famille de réseaux dont l'appellation provient du *traitement du signal*. Un réseau *Encoder-Decoder* se compose de deux parties: la partie Encoder et la partie Decoder. Ces deux parties sont formées par deux réseaux indépendants.

L'objectif de l'Encoder est de représenter la donnée d'entrée (définie dans un espace  $\mathcal{X}$ ) dans un espace  $\mathcal{Z}$  souvent de dimension inférieure. La donnée obtenue s'appelle *vecteur de contexte* (context vector ou code). Dans le cas où la dimension de l'espace de sortie est inférieure à celle de l'espace d'entrée, on dit que la représentation de l'entrée est *compressée*.

Le Decoder est le complémentaire de l'Encoder. Il exploite la projection de l'Encoder afin de créer une nouvelle projection dans un autre espace souhaité. Il n'y a aucune obligation à conserver le même espace que celle de la donnée d'entrée ou du vecteur de contexte. La non-obligation de conservation de l'espace permet de rendre l'application de ce type d'architecture très diversifiée, notamment dans le cadre du nettoyage/réduction des données (pré-processing), la traduction multi-langue et la génération de données artificielles.

Ainsi, nous pouvons résumer l'action d'un Encoder-Decoder par:

$$\begin{aligned}\phi : \mathcal{R}^x &\xrightarrow{\mathcal{R}^z} \phi(x) \\ \varphi : \mathcal{R}^z &\xrightarrow{\mathcal{R}^y} \varphi(z) \\ ED : \mathcal{R}^x &\xrightarrow{\mathcal{R}^y} \varphi(\phi(x))\end{aligned}$$

Avec  $\phi$ , la fonction de l'Encoder et  $\varphi$ , celle du Decoder.

## 6.2 Autoencoder

Un *Autoencoder*[6] est un cas particulier d'architecture Encoder-Decoder. Il est caractérisé par un espace de projection du Decoder identique à celui de la donnée d'entrée et l'objectif est de "retrouver" la donnée d'entrée. Ainsi, l'action d'un Autoencoder peut être définie par:

$$\begin{aligned}\phi : \mathcal{R}^x &\xrightarrow{\quad} \mathcal{R}^z \\ \varphi : \mathcal{R}^z &\xrightarrow{\quad} \mathcal{R}^x \\ ED : \mathcal{R}^x &\xrightarrow{\quad} \varphi(\phi(x)) \rightarrow x\end{aligned}$$

L'Autoencoder est composé de structures comparables à un modèle Feed-Forward (Full-Connected) ou convolutif. Son apprentissage est donc possible avec une approche classique. Pour cela, la rétropropagation du gradient et les fonctions de coût standards sont employées. Ainsi, pour une donnée d'entrée binaire  $x$  de dimension  $n$ , nous utilisons la *Binary Cross-Entropy*:

$$\mathcal{L}(\theta) = - \sum_{i=1}^n [x_i \log(\hat{x}_i) + (1 - x_i) \log(1 - \hat{x}_i)]$$

Dans le cas d'une entrée  $x$  à valeur réelle dans  $\mathcal{R}^n$ , nous préfèrerons la *Mean squared error* (MSE):

$$\mathcal{L}(\theta) = \frac{1}{n} \sum_{i=1}^n (\hat{x}_k - x_k)^2$$

Du fait que l'objectif est d'obtenir à nouveau la donnée d'entrée, la labellisation de la donnée d'apprentissage est inutile, ce qui fait d'un Autoencoder, une méthode d'apprentissage *non supervisé*.

Un Autoencoder peut avoir une ou plusieurs couche cachées. Dans le cas d'accumulation de couches cachées, nous parlerons de *Deep Autoencoder*. Les Deep Autoencoder favorisent l'utilisation de fonction d'activation non linéaire, l'exploitation de couche de convolution au détriment des couches Full-Connected et ont un grand pouvoir explicatif. Le risque d'overfitting est donc plus important et demande une attention particulière.

L'approche standard de l'Autoencoder est essentiellement utilisée dans le cadre d'une réduction de dimension<sup>87</sup>. Elle est donc utile pour réaliser une compression de donnée ou du pré-processing de données. Ils sont aussi utilisés dans le cadre d'une recherche de complétion de données en présence de données incomplètes ou partiellement corrompues.

---

<sup>87</sup>Ce modèle est capable d'approximer les résultats d'une ACP linéaire et non linéaire par exemple

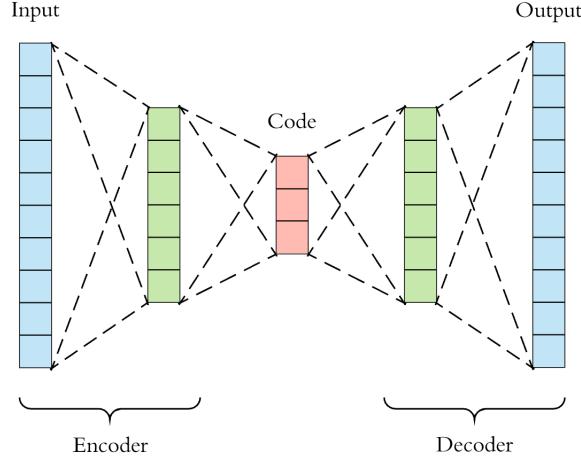


Figure 46: Architecture d'un Autoencoder

### 6.2.1 Autoencoder Undercomplete

L'*Autoencoder Undercomplete* est souvent associé à l'architecture de base d'un Autoencoder. L'idée derrière cette architecture est de réaliser une projection de la donnée d'entrée sur un espace de dimension plus **faible**. Ainsi, l'Encoder est défini par:

$$\phi : \mathcal{R}_x^X \xrightarrow{\quad} \mathcal{R}_z^Z \quad \text{avec } Z \ll X$$

La projection sur un espace de plus petite dimension force l'Autoencoder à extraire l'information importante des données d'entrée. Si la dimension de l'espace de projection est identique à celle de l'espace de la donnée d'entrée, le modèle aura tendance à apprendre la fonction *identité*, ce qui empêchera l'extraction d'information des données. Cette particularité rendra donc inutile l'Encoder et de ce fait, le Decoder car le vecteur de contexte tendra à être équivalent à la donnée d'entrée. Une représentation d'un Autoencoder est visible sur la Figure 46.

### 6.2.2 Autoencoder Régularisé ou Overcomplete

Un *Autoencoder Overcomplete* s'émancipe de la dimension du vecteur de contexte et propose de réaliser une projection sur un espace de dimension identique voire supérieure. Afin de lutter contre le risque de création d'une représentation *identité*, cette architecture *régularise* la fonction de coût pour pousser le modèle à extraire des propriétés des données au lieu d'une simple représentation de l'identité de la donnée d'entrée. Cette approche tend à être plus robuste et plus efficace que la méthode Undercomplete.

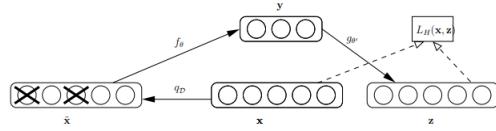


Figure 47: Architecture d'un Denoising Autoencoder

Il existe différentes régularisations possibles afin de réaliser ce type d'Autoencoder. Ces méthodes peuvent être associées et sont encore un sujet de recherche actuel.

### 6.2.2.1 Denoising Autoencoder

Au lieu d'analyser la donnée d'entrée intègre, un *Denoising Autoencoder*[97] exploite la donnée d'entrée *bruitée*. Ce réseau doit donc apprendre à débruiter au lieu d'uniquement *copier* l'entrée.

Il existe différente méthode pour bruiter les données (application d'un bruit gaussien, filtre quelconque, etc... ). En pratique, l'approche la plus populaire consiste à désactiver aléatoirement des neurones de la couche d'entrée en forçant leurs sorties à 0. Cette méthode est comparable à l'application d'un DropOut sur la couche d'entrée du réseau. Une illustration de ce réseau est visible sur la Figure 47.

**Important** Lors du calcul de la fonction de coût, la comparaison est réalisée avec la donnée intègre et non la donnée bruitée ! Comparer à la donnée bruitée reviendrait à réaliser un Autoencoder qui apprendrait à représenter les données bruitées et non intègres.

Un *Denoising Autoencoder* est donc défini par:

$$\begin{aligned} \text{Noise} : \mathcal{R}^x &\xrightarrow{\quad} \mathcal{R}^x \\ \phi_{\text{Noise}} : \mathcal{R}^x &\xrightarrow{\quad} \mathcal{R}^z \\ \varphi : \mathcal{R}^z &\xrightarrow{\quad} \mathcal{R}^x \\ ED : \mathcal{R}^x &\xrightarrow{\quad} \varphi(\phi(\text{Noise}(x))) \rightarrow x \end{aligned}$$

Cette architecture peut être utilisée pour créer un *débruiteur*. En effet, dans la méthode standard, la donnée intègre est corrompue par une architecture particulière du réseau ou l'ajout d'un bruit artificiel en amont. Nous créons donc, en interne, un jeu de données composé d'entité  $[donnee_{integre}, donnee_{bruitee}]$ . Le réseau devient donc robuste à corriger les effets du bruit introduit dans ces données. Neanmoins, la correction du bruit est, dans cette configuration, peu

utile d'un point de vue métier. Une autre approche serait de réaliser un apprentissage *supervisé* où les données d'entraînement sont issues d'un jeu de données corrompus par un bruit spécifique. L'entraînement sur ce jeu de données permettra ainsi d'obtenir un modèle apte à débruiter un message utile soumis au bruit présent dans les données d'entraînement. Ce modèle obtenu aura donc une plus-value métier car capable de réaliser un débruitage *utile*. On peut voir une application directe dans le cadre des télécoms.

### 6.2.2.2 Sparse Autoencoder

Le *Sparse Autoencoder*[69] cherche à rendre son architecture éparse<sup>88</sup>, i.e éviter que l'intégralité des neurones soit forcée d'être stimulée à chaque donnée d'entrée. Cette approche tend à rendre les neurones plus spécialisés en limitant leurs degrés de liberté qui pourraient nuire à la qualité du neurone si les données sont significativement différentes. L'objectif est donc de rendre un neurone sensible à une stimulation spécifique et non l'ensemble des stimulation qu'il reçoit. Ainsi, un neurone doit être inactif la majorité du temps. Cette caractéristique est très importante en présence de données significativement distinctes tels que des jeux de données multi-classes.

Afin de réguler l'activité des neurones, deux approches sont essentiellement employées: La régularisation par la Divergence de Kullback-Leibler ou la k-Sparse constraint.

- **KL-divergence:** La régularisation par la KL-divergence modifie la fonction de coût en rajoutant un facteur correspondant à la divergence de Kullback-Leibler.

Supposons un neurone  $i$  appartenant à une couche cachée et  $a_i$ , son activation. On notera  $a_i(x_j)$  l'activation du neurone  $i$  selon la donnée d'entrée  $x_j$ . On peut donc définir la valeur moyenne d'activation du neurone  $i$  sur un jeu de données de  $m$  éléments par:  $\hat{\rho}_j = \frac{1}{m} \sum_{j=1}^m [a_i(x_j)]$ .

Nous voulons que  $\hat{\rho}_j = \rho$  où  $\rho$  est le paramètre déterminant le degrés de dispersion du réseau.  $\rho$  doit être faible (supposons 0.05). Cela signifie que nous souhaitons que la moyenne d'activation tende vers 0.05. Ainsi, cette condition impose que le neurone soit constamment éteint (activation proche de 0) en dehors de phase d'activation ponctuelle où la valeur tendra vers 1. Cette condition suppose une activation dans  $[0, 1]$  telle que l'activation via une sigmoïde<sup>89</sup> par exemple.

---

<sup>88</sup>On parle de *sparsity constraint*

<sup>89</sup>Il est possible de considérer d'autre fonction telle que tanh en modifiant les conditions d'excitation/repos du neurone. Il est cependant nécessaire de considérer des fonctions qui présentent des bornes asymptotiques vers les infinis.

Le statut d'un neurone peut être simplifié par une approximation binaire: allumé ou éteint. Ainsi, on peut considérer l'état d'un neurone comme un problème explicable par la loi de Bernouilli. La divergence de Kullback-Leibler permet de calculer la différence entre deux distributions. Ainsi, il est possible de déterminer la différence entre la distribution des activations du neurone réel (variable aléatoire de Bernouilli de moyenne  $\hat{\rho}_j$ ) et la distribution idéale souhaitée (variable aléatoire de Bernouilli de moyenne  $\rho_j$ ). Le calcul de la différence est défini par:

$$KL(\rho||\hat{\rho}_j) = \rho \log \frac{\rho}{\hat{\rho}_j} + (1 - \rho) \log \frac{1 - \rho}{1 - \hat{\rho}_j}$$

$KL(\rho||\hat{\rho}_j)$  est égale à 0 en cas d'égalité des distributions et augmente lorsque les distributions se différencient. Son action a donc le même comportement qu'une régularisation classique telle que  $\mathcal{L}_2$  par exemple. La fonction de coût est donc définie par:

$$\mathcal{L}(\theta)_{sparse} = \mathcal{L}(\theta) + \beta \sum_{j=1}^{s_2} KL(\rho||\hat{\rho}_j)$$

On suppose qu'il y a  $s_2$  neurones dans les couches cachées.  $\beta$  est un coefficient qui contrôle l'importance donnée à la condition de dispersion du réseau.

- **k-Sparse constraint**[66]: Un k-Sparse Autoencoder réalise une action comparable à un DropOut/DropConnect. En effet, l'approche proposée par ce modèle consiste à ne considérer que les  $k$  plus grandes valeurs d'activation d'une couche cachée (généralisable à l'ensemble des couches cachées indépendamment les unes des autres) et à mettre à 0, toutes les autres. La rétropropagation du gradient n'est effective que sur les neurones activés. Lors de l'utilisation du modèle, il faudra exploiter les  $\alpha k$  valeurs les plus importantes avec  $\alpha \geq 1$ . Il a été montré qu'un  $\alpha$  supérieur à 1 donne des résultats sensiblement meilleurs<sup>90</sup>. L'impact de la valeur de  $k$  est visible sur la Figure 48.

#### 6.2.2.3 Contractive Autoencoders

Un Contractive Autoencoders [79] repose sur l'idée que le modèle doit être robuste aux changements et éviter des variations importantes dans la représentation du vecteur de contexte (autour des données d'apprentissage). Pour cela, une régularisation par la norme Frobenius de la matrice jacobienne de l'Encoder est utilisée.

La nouvelle fonction de coût est exprimée par:

$$L = \|X - \hat{X}\|_2^2 + \lambda \|J_h(X)\|_F^2$$

---

<sup>90</sup>Sans pour autant exagérer sur sa valeur...

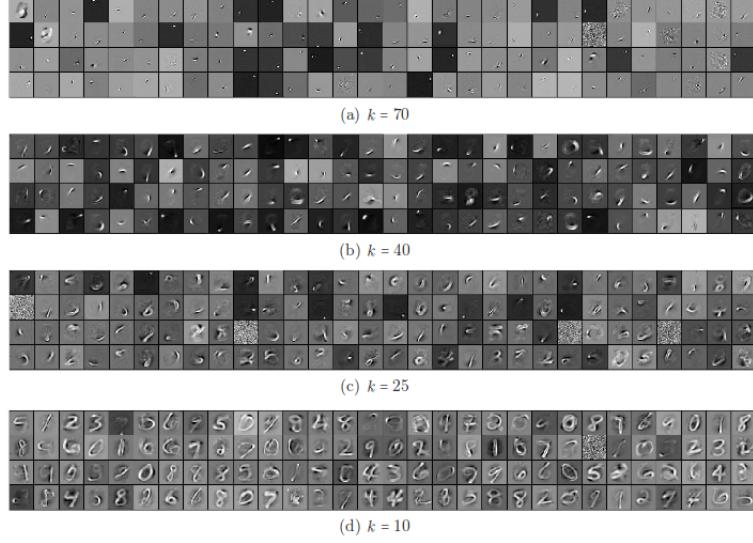


Figure 48: Filtre d'un k-Sparse Autoencoder selon la valeur de  $k$  sur le jeu de données MNIST

$$\|J_h(X)\|_F^2 = \sum_{ij} \left( \frac{\partial h_j(X)}{\partial X_i} \right)^2$$

Détaillons le calcul de la régularisation. Supposons une fonction d'activation sigmoïde. Nous avons donc:

$$Z_j = W_i X_i \quad (1)$$

$$h_j = \phi(Z_j) \quad (2)$$

$$\frac{\partial h_j}{\partial X_i} = \frac{\partial \phi(Z_j)}{\partial X_i} \quad (3)$$

$$= \frac{\partial \phi(W_i X_i)}{\partial W_i X_i} \frac{\partial W_i X_i}{\partial X_i} \quad (4)$$

$$= [\phi(W_i X_i)(1 - \phi(W_i X_i))] W_i \quad (5)$$

$$= [h_j(1 - h_j)] W_i \quad (6)$$

On obtient:

$$\|J_h(X)\|_F^2 = \sum_{ij} \left( \frac{\partial h_j}{\partial X_i} \right)^2 \quad (7)$$

$$= \sum_i \sum_j [h_j(1 - h_j)]^2 (W_{ji}^T)^2 \quad (8)$$

$$= \sum_j [h_j(1 - h_j)]^2 \sum_i (W_{ji}^T)^2 \quad (9)$$

Ce calcul est très semblable au calcul utilisé durant la retropropagation du gradient. En supposant une couche caché de 20 neurones, alors le problème est déterminé par 20 fonctions définies par un vecteur de gradient chacune d'où l'obtention d'une matrice jacobienne. Il est important de noter que cette régulation exploite une norme de la jacobienne. Cette particularité est intéressante car elle permet d'éviter de devoir définir la matrice diagonale de  $[h(1 - h)]^2$ . En effet, Ce facteur dépend d'une variable autre que celle de W. Il faut donc associer à chaque dimension de W, l'entité (unique) de  $[h(1 - h)]^2$  correspondant, ce qui impose l'exploitation d'une matrice diagonale qui peut être délicate à obtenir.

La régularisation de  $\|J_h(X)\|_F^2$  est uniquement destructrice. Elle punit toute variation associée à l'activation des neurones de l'Encoder. Néanmoins, cette régularisation est compensée par  $\|X - \hat{X}\|_2^2$  qui peut avoir un bon résultat.  $\|X - \hat{X}\|_2^2$  possède un résultat amélioré lorsque la variation demandée au réseau lors d'une mise à jour des poids est bénéfique à l'apprentissage. Au contraire, lors d'une variation sans impact,  $\|X - \hat{X}\|_2^2$  stagne. Dans ce dernier cas, l'impact de la régularisation est trop fort et l'optimisation du réseau ne considérera pas cette possibilité d'évolution. Ainsi, seules les modifications de faible ampleur<sup>91</sup> autour des données d'apprentissage seront considérées, les autres variations étant *contractées*. Il serait intéressant d'approfondir les problématiques de convergence de cette approche qui présente une faible capacité d'exploration<sup>92</sup>.

Le facteur  $\lambda$  est important. En effet, ce paramètre régule l'importance donnée à la régulation. Plus  $\lambda$  est élevé, plus les variations seront pénalisées. Un facteur trop important bloquera donc le réseau car aucune variation ne sera jugée suffisamment bénéfique pour compenser la pénalisation de la variation.

---

<sup>91</sup>Une variation massive qui produit un gain de performance important peut cependant être accepté. Tout dépend du degrés de variation et du gain de performance associé

<sup>92</sup>Le compromis Exploration/Exploitation est récurrent en apprentissage automatique, notamment en apprentissage par Renforcement

### 6.3 Variational Autoencoders

**Important:** Comprendre ce modèle d'un point de vue théorique nécessite un solide bagage en probabilité (inférence bayesienne). Nous n'approfondirons donc pas l'origine et les démonstrations théoriques de ce modèle.

Le *Variational Autoencoders*(VAE)[15][49] est l'architecture d'Autoencoder la plus récente. La spécificité de ce modèle est qu'il apprend un modèle de **variable latente**. Un VAE est donc un **modèle génératif à conditions**. Contrairement au GAN<sup>93</sup> qui génère des entités aléatoirement, ce type de modèle est capable de générer des entités sous conditions, permettant ainsi de cibler la nature des résultats que nous souhaitons obtenir.

Un Encoder standard réalise une projection d'un vecteur sur  $\mathcal{R}^{dim}$ . Chaque image d'entraînement est donc clairement définie par un vecteur unique sur  $\mathcal{R}^{dim}$ . Cet aspect est très problématique lorsque nous désirons un modèle **génératif** car le Decoder ne peut s'émanciper des données d'apprentissage. En effet, l'espace de projection aura tendance à être très éparse et à présenter différents clusters éloignés les uns des autres et formés par des entités d'apprentissage similaires. Cette discontinuité et les difficultés d'interpolation associées font que le Decoder n'est pas capable de considérer des vecteurs sur  $\mathcal{R}^{dim}$  non référencés<sup>94</sup>. Il n'est donc pas capable d'analyser des vecteurs de contexte inconnus, ce qui est problématique dans le cadre de la génération de données.

Le *Variational Autoencoders* propose une solution à ce problème en introduisant la notion de *variable latente* dans son vecteur de contexte. Au lieu de prédire des valeurs discrètes, l'Encoder va déterminer des paramètres d'une distribution de probabilité, ce qui permet une généralisation efficace. Ainsi, l'Encoder inférera la moyenne et l'écart-type d'une distribution normale pour chacune des dimensions du vecteur de contexte. L'architecture du réseau est visible sur la Figure 49. L'Encoder prédit un vecteur moyenne et un vecteur écart-type où chaque dimension de ces deux vecteurs correspondent deux-à-deux aux paramètres d'une distribution normale associée à la dimension concernée du vecteur de contexte. Une explication graphique est montrée par la Figure 50. Le Decoder recevra donc un vecteur inféré selon les distributions du vecteur de contexte.

Cette configuration présente une problématique majeure. Ne pas régulariser les paramètres favorisera la détermination de moyennes de distribution très éparques et diversifiées (selon les classes de données) avec un écart-type qui aura tendance à être faible. Cet aspect est préjudiciable car nous souhaitons que les données soient les plus proches possibles afin de faciliter les interpolations nécessaires à la création de nouvelles données. Afin de lutter contre ce phénomène, la fonc-

---

<sup>93</sup>Nous étudierons ce type de modèle par la suite

<sup>94</sup>Le Decoder produira un résultat souvent mauvais car il ne "comprendra" pas le vecteur de contexte proposé

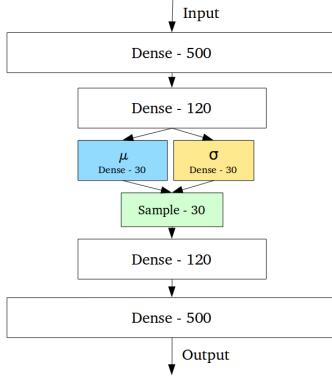


Figure 49: Architecture d'un Variationnal Autoencoder

tion de coût est régularisée par la divergence de KullBack-Leibler qui permet de déterminer les divergences entre deux distributions. Ainsi, dans le cas du VAE, le facteur de régularisation sera définie par la somme des KL-divergence par rapport à la distribution normale centrée en 0 d'écart-type unitaire. L'objectif est donc de minimiser les écarts entre les distributions en les contraignant à être le plus proche possible d'une même distribution. L'union de Mean squared error et de la KL-divergence permet donc de conserver l'association des classes par cluster tout en favorisant une répartition dense.

Pour réaliser une génération de données, il n'y aura qu'à proposer un vecteur latent issu d'une distribution gaussienne unitaire au Decoder. L'extrapolation est possible du fait de la proximité des clusters, il est donc possible d'obtenir une infinité de génération en variant les valeurs obtenues par les différentes distributions. Un exemple de génération est visible sur la Figure 51.

Ce type d'architecture est un sujet de recherche très étudié, notamment en exploitant des architectures de réseaux neuronaux déjà existants pour améliorer les capacités de l'Encoder/Decoder (tels que les LSTM). Les capacités de génération de données sont très convoitées par les domaines créatifs tels que les Arts ou la génération de données artificielles afin de répondre à la problématique du déficit de données d'apprentissage. On peut ainsi noter *PixelVAE*[24] dans le cadre de la génération d'image; *MusicVAE*[1] pour la génération de musique et [83][33] pour la génération de texte.

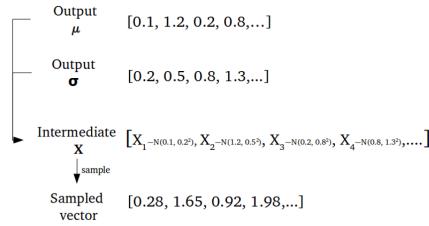


Figure 50: Détermination du vecteur de contexte par un Variationnal Autoencoder

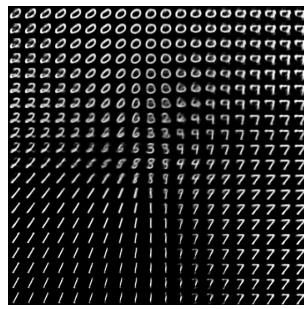


Figure 51: Generation d'image par un Variationnal Autoencoder entraîné sur MNIST

## 7 Réseaux antagonistes génératifs

### 7.1 Généralités

Un Réseau antagoniste génératif[22] (Generative Adversarial Networks (GAN)) est un réseau dont l'architecture s'inspire d'un problème issu de la *Théorie des Jeux*, le *Minimax*<sup>95</sup>. Ses performances sont remarquables dans le cadre de la *génération de données*, notamment les images. Ce réseau est composé de deux sous-réseaux: un *Générateur* et un *Discriminateur*.

L'objectif du *Générateur* est de produire une donnée artificielle alors que le *Discriminateur* doit dissocier les images réelles des images artificielles. Le réseau va ainsi apprendre en cherchant à améliorer ses deux sous-réseaux: un *Discriminateur* plus efficace à détecter les *faux* et un *Générateur* plus efficace pour produire des *faux* "invisibles".

Le *Générateur* prend en entrée, un vecteur *bruitée*  $c(\theta) \in R^{d96}$  issu d'une distribution normale ou uniforme entre -1 et 1 avant de produire une donnée artificielle

<sup>95</sup>Pour plus d'informations sur ce jeu, consultez [http://people.maths.ox.ac.uk/griffit4/Math\\_Alive/3/game\\_theory3.pdf](http://people.maths.ox.ac.uk/griffit4/Math_Alive/3/game_theory3.pdf)

<sup>96</sup>Par convention, d est égal à 100

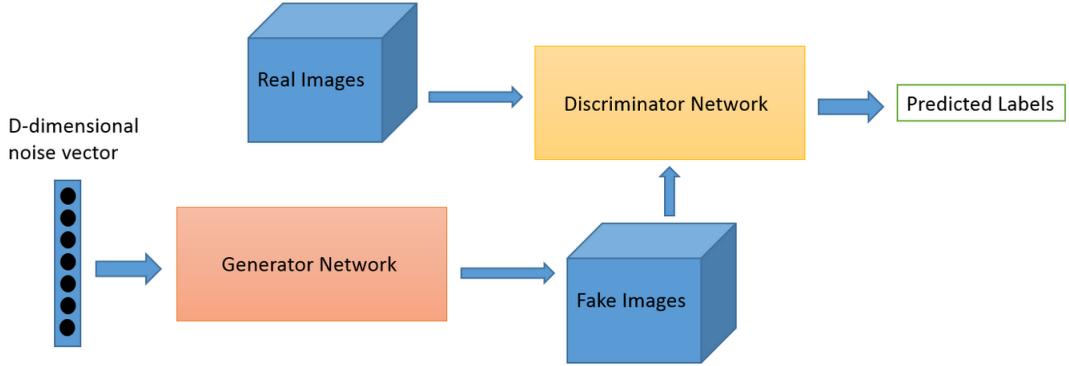


Figure 52: Architecture d'un réseau antagoniste génératif

de même dimension que les données réelles. Le *Desriminateur* est un réseau de classification binaire standard et reçoit en entrée, une donnée réelle et une donnée artificielle qu'il doit discriminer. Une illustration est visible sur la Figure 52.

Lorsque l'apprentissage est effectué, les données générées non discriminées par le *Desriminateur* constituent l'ensemble exploitable par l'utilisateur.

#### 7.1.0.1 Fonction de perte

Supposons:

- $p_y$ , distribution du *bruiteur* associée à une entrée  $y$
- $p_r$ , distribution associée aux données réelles  $x_{real}$
- $D$ , fonction représentant le Discriminateur
- $G$ , fonction représentant le Générateur

Dans un premier temps, nous voulons que le Discriminateur donne une forte probabilité d'être valide à une donnée réelle. Nous voulons donc que  $D(x_{real}) \rightarrow 1$ . Ainsi, nous pouvons considérer la fonction:

$$E_{x \sim p_r(x_{real})}[\log(D(x))] \\ E_{x \sim p_r(x_{real})}[\log(D(x))] \rightarrow 0 \text{ si } D(x_{real}) \rightarrow 1$$

De même, nous voulons que le Discriminateur prédise une faible probabilité d'être valide pour une donnée artificielle soit  $D(G(y)) \rightarrow 0$ . Nous obtenons alors la fonction:

$$E_{y \sim p_y(y)}[\log(1 - D(G(y)))]$$

Nous obtenons notre fonction de perte définie par:

$$L(D, G) = E_{x \sim p_r(x_{real})}[\log D(x)] + E_{y \sim p_y(y)}[\log(1 - D(G(y)))]$$

Une particularité importante est à remarquer. La fonction L est comparable au **négatif** de la fonction *Cross-Entropy*. Ainsi, pour l'apprentissage du Discriminateur, il est nécessaire de réaliser une optimisation par *gradient ascent* et non une descente de gradient standard car la "minimisation" de l'erreur du Discriminateur revient à maximiser cette fonction et non la minimiser comme pratiqué classiquement en Deep Learning. En effet,  $L(D, G) \leq 0$ .

Au contraire, pour l'apprentissage du Générateur, nous souhaitons une valeur élevée pour  $D(G(y))$ . Nous pouvons réutiliser la fonction  $E_{y \sim p_y(y)}[\log(1 - D(G(y)))]$ . L'apprentissage de ce réseau s'oppose ainsi à celui du Discriminateur et se basera sur la minimisation de la fonction de perte.

Le Générateur et le Discriminateur cherchent à optimiser deux fonctions de pertes **opposées**. Nous pouvons ainsi définir un jeu *minimax* basé sur la probabilité que la donnée artificielle soit considérée comme vrai.

En unifiant les différentes conditions, nous obtenons:

$$\min_G \max_D L(D, G) = E_{x \sim p_r(x_{real})}[\log D(x)] + E_{y \sim p_y(y)}[\log(1 - D(G(y)))]$$

Plus spécifiquement, l'apprentissage se fait en alternant l'optimisation des sous-réseaux indépendamment l'un de l'autre selon les fonctions suivantes:

- Générateur: descente de gradient

$$\min_G E_{y \sim p_y(y)}[\log(1 - D(G(y)))]$$

- Discriminateur: gradient ascent

$$\max_D E_{x \sim p_r(x_{real})}[\log D(x)] + E_{y \sim p_y(y)}[\log(1 - D(G(y)))]$$

L'alternance de l'apprentissage peut être de 1(D):1(G) ou de k:1. De nombreux chercheurs ont observé expérimentalement que l'apprentissage était plus stable avec un apprentissage plus soutenu du Discriminateur. La stabilité des GAN est une des problématiques majeures de ce type d'architecture car il est difficile de faire apprendre mutuellement deux réseaux. Si l'un des deux modèles dominent l'autre, la performance de l'ensemble deviendra mauvaise. La création de fonctions de perte plus performantes est un sujet de recherche toujours très actif.

Dans les faits, la fonction de perte du Générateur est problématique. En effet, cette fonction ne permet pas à G d'apprendre efficacement. Lorsque le Générateur produit un faux et que ce dernier est détecté, il est nécessaire que le Générateur apprenne efficacement de son erreur. Or, la valeur du gradient tend à être infinitésimal lorsque la fonction de perte tend vers 0, cas obtenu lorsque la probabilité  $D(G(z))$  est nulle. Au contraire, le gradient augmente lorsque le Générateur produit des faux capables de tromper le Discriminateur. Ce comportement est inadéquat car il ne permet pas au Générateur de bien apprendre

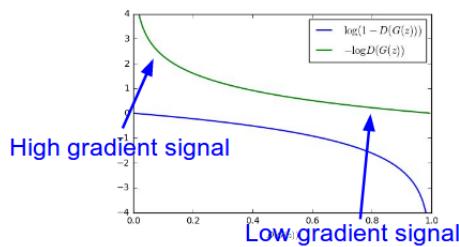


Figure 53: Comparaison des gradients de la fonction de perte du Génératuer

lorsqu'il n'a aucune connaissance. Sur la courbe bleue de la Figure 53, nous pouvons observer le comportement du gradient selon la probabilité prédite par le Discriminateur.

Une alternative consiste à exploiter  $\log(D(G(z)))$ . L'approche du comportement du Discriminateur vis-à-vis du Génératuer est inversée. Au lieu de minimiser la probabilité que le Discriminateur détecte le faux, on cherchera à maximiser le fait que le Discriminateur se trompe sur sa prédiction. Nous obtenons:

$$\max_G E_{y \sim p_y(y)}[\log(D(G(y)))]$$

Le comportement du gradient de la nouvelle fonction de perte du Génératuer est visible sur la courbe verte de la Figure 53. On peut observer que le comportement du gradient est inversé et qu'il favorise l'apprentissage lorsque le modèle est peu performant.

## 7.2 Difficultés d'apprentissage

Bien que très puissants, les réseaux GAN sont reconnus comme étant difficiles à entraîner. En effet:

- **Convergence:** Le risque de non-convergence est important avec ce type de réseau. L'apprentissage est alors caractérisé par un comportement très oscillant qui a tendance à finir par diverger.
- **Diversité:** Les images générées par les GAN ont tendance à être très similaires et à manquer de diversité malgré un jeu de données d'apprentissage représentatif.
- **Génération aléatoire:** Il n'est pas possible de maîtriser la génération d'images. Le comportement est exclusivement aléatoire (contrairement aux *Variational Autoencoder*s).

- **Equilibre des réseaux:** Pour que le réseau GAN fonctionne, il est nécessaire que le Générateur et le Discriminateur soit globalement aussi performant l'un que l'autre. Si l'un des modèles domine l'autre, le GAN ne sera pas capable de fonctionner (et d'apprendre) convenablement.
- **Hypersensibilité:** Les architectures GAN sont très dépendantes d'hyperparamètres difficiles à évaluer et sont sujets au sur-apprentissage

Ces différentes contraintes ont été la source d'une recherche active, notamment au niveau de la fonction de perte qui, dans sa version initiale, est globalement peu performante. De nombreuses améliorations ont été proposées à ce niveau notamment le *Wasserstein GAN*[3] qui repose sur la *distance Wasserstein*<sup>97</sup>. Nous n'approfondirons pas les améliorations apportées à l'architecture GAN dans ce cours.

### 7.3 Deep Convolutional Generative Adversarial Networks (DCGAN)

Les réseaux DCGAN[72], contrairement aux réseaux GAN classiques, exploitent une architecture convulsive profonde pour le Générateur et le Discriminateur. Le Discriminateur repose sur les structures convolutives classiques telles que Inception, ResNet ou encore VGG. Une particularité notable est le remplacement des couches de Pooling par des convolutions avec un stride supérieur à 1 qui permettent d'avoir une réduction de dimension moins agressive en terme de perte d'informations. Au contraire, le Générateur repose sur des méthodes de *Upsampling* afin de pouvoir générer une image qui est de dimension supérieure au signal bruité exploité comme source génératrice. La méthode standard de *Upsampling* est la *Convolution transposée* (aussi appelée Déconvolution<sup>98</sup>).

## 8 Réseaux siamois

### 8.1 Généralités

Les réseaux siamois[51] constituent une famille d'architectures formées de deux (ou plus) sous-réseaux identiques, i.e même configuration avec des paramètres et poids identiques. Lors de l'apprentissage, les sous-réseaux sont mis à jour de manière identique. La nature des architectures des sous-réseaux est variable (convolutif, récurrent, Full-Connected...) et dépend de la nature des données à analyser. Ce réseau est composé de deux parties:

- **Extraction d'attributs:** Cette tâche est réalisée par les sous-réseaux qui vont représenter la donnée d'entrée sous la forme d'un vecteur d'attribut

---

<sup>97</sup>Cette métrique permet d'évaluer la distance entre deux distributions de probabilités.

<sup>98</sup>Avec abus

résumant l'information portée. Les sous-réseaux étant identiques, le comportement des extracteurs est identique peu importe la donnée analysée (donnée de référence et donnée à analyser). Ainsi, dans le cas de la reconnaissance faciale, les deux visages à comparer seront représentés par un vecteur définissant les attributs physiques des deux individus.

- **Mesure de similarité:** Cette tâche est réalisée par une métrique de distance qui évalue la similarité entre deux vecteurs d'attributs. On peut citer la *Mean Squared Error* et la *Distance Cosinus* par exemple<sup>99</sup>. La sortie est donc une valeur comprise entre 0 et 1 qui détermine le degrés de similarité entre deux entités.

Néanmoins, ce type de métrique est peu effective dans les faits car elle favorise des convergences peu effectives, notamment en tolérant que les valeurs de sortie des entités jugées similaires et dissimilaires soient "proches". Il est préférable de favoriser un écart important entre les résultats positifs et négatifs, i.e imposer une *marge*. Cette particularité est réalisé par la *Contrastive Loss function*[25]<sup>100</sup>. Soit  $\vec{X}_i$ , une valeur d'entrée et  $G(\vec{X}_i)$ , la sortie obtenue après action d'un sous-réseau et  $y$ , valeur binaire de prédiction. *Contrastive Loss function*<sup>101</sup> est ainsi définie<sup>102</sup> par:

$$(1 - y) * \frac{1}{2} D_w^2(\vec{X}_1, \vec{X}_2) + y * \frac{1}{2} (\max(0, m - D_w(\vec{X}_1, \vec{X}_2)))^2$$

$$D_w(\vec{X}_1, \vec{X}_2) = \|G(\vec{X}_1) - G(\vec{X}_2)\|_2$$

Un exemple de réseau siamois est visible sur la Figure 54. Les sous-réseaux suivent une architecture Full-connected. Les vecteurs caractéristiques sont dans  $R^2$ .

L'objectif de ce type de réseau est d'évaluer la similarité entre deux entités, ce qui s'oppose avec la tâche classique des réseaux de neurones qui classifient une entité. De nombreux problèmes nécessitent une comparaison entre deux entités (peu importe si le réseau est capable de les classifier), notamment la reconnaissance faciale (*Face Recognition*), de signature ou encore, d'écriture (par rapport au style graphique si écrit à la main ou au style syntaxique). Plus récemment, de nouvelles thématiques ont été abordées telles que l'évaluation de la pertinence d'une réponse à une question donnée<sup>103</sup> ou encore le *Tracking*. Ce type de réseau est très exploité dans le domaine du **One-Shot Learning**.

---

<sup>99</sup>Il existe de nombreuses autres métriques utilisables !

<sup>100</sup>Il existe aussi des métriques qui respecte cette condition en ne se basant pas sur une marge

<sup>101</sup>On supposera qu'une similarité parfaite équivaut à un résultat de 0

<sup>102</sup>Voir la Section 8.2 pour plus de détails sur la notion de marge.

<sup>103</sup>Approche très utile dans le cadre d'un apprentissage d'un ChatBot

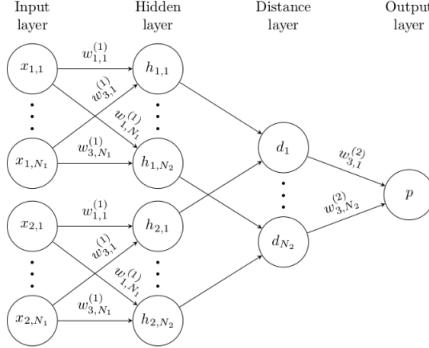


Figure 54: Exemple simple d'un réseau siamois

## 8.2 Triplet Loss

Une autre fonction populaire - comparable à la *Contrastive Loss function* - est la *Triplet Loss*[82]. La *Contrastive Loss function* cherche à minimiser la distance si les deux images sont de la même classe et à maximiser (associer une valeur supérieure à une marge) si les deux images sont de classes différentes. Cette métrique considère des couples d'entités et réalise sa discrimination en fonction de la concordance des entités de ce couple. Ainsi, elle maximise (ou minimise) les distances entre entités de manière dissociées.

Au contraire, *Triplet Loss* exploite des triplets d'entités (A,P,N) avec A, image de référence, P, image positive (de même classe) et N, image négative (de classe différente). L'objectif de cette métrique est de permettre que la projection de l'image positive soit plus "proche" de la référence que l'image négative, i.e maximiser la différence entre (A-N) et (A-P). Cette approche considère ainsi les distances de manière *relative*. En effet, sa référence est basée sur la différence entre (A-N) et (A-P) et non des distances *absolues*. Cette métrique est donc plus *laxiste* car elle ne forcera pas la réduction de distance qui peut être trop exigeante. Par exemple, dans le cas de la *Contrastive Loss function*, l'apprentissage va chercher à diminuer au maximum la distance entre deux entités similaires alors que dans le cas de *Triplet Loss*, l'apprentissage se limitera à garantir une bonne séparation entre les entités similaires et dissimilaires. La condition est donc moins stricte.

Supposons un triplet (A,P,N). Nous voulons faire en sorte que l'image positive soit plus proche de la référence que l'image négative. Supposons f, la fonction de transformation des sous-réseaux. Ainsi, nous obtenons:

$$\underbrace{\|f(A) - f(P)\|_2}_{D(A,P)} \leq \underbrace{\|f(A) - f(N)\|_2}_{D(A,N)}$$

$$\underbrace{\|f(A) - f(P)\|_2}_{D(A,P)} - \underbrace{\|f(A) - f(N)\|_2}_{D(A,N)} \leq 0 \quad (1)$$

Cette approche présente une faiblesse majeure. En effet, la condition tolère le cas où:

$$D(A, P) \rightarrow 0$$

$$D(A, N) \rightarrow 0$$

Pour contrer ce phénomène, on instaure une *marge* qui forcera un éloignement des projections des entités négatives. Ainsi, nous obtenons:

$$D(A, P) + \alpha \leq D(A, N)$$

$$D(A, P) - D(A, N) + \alpha \leq 0 \quad (2)$$

Supposons un exemple. Si on utilise l'équation (1), si  $D(A,P)=0.3$  et  $D(A,N)=0.33$ , alors la condition est remplie. Néanmoins, la discrimination est très faible et le risque de faux-positif et faux-négatif élevé à cause de cette proximité. Supposons l'équation (2) si  $D(A,P)=0.3$  et  $\alpha = 0.3$  alors il faut que  $D(A, N) \geq 0.6$  pour que la condition soit satisfaite, ce qui favorise une discrimination importante.

Il reste à formaliser cette condition afin de la rendre exploitable par un réseau de neurones. Nous pouvons réécrire l'équation (2) telle que pour un triplet  $(A, P, N)$ :

$$\mathcal{L}(A, P, N) = \max(\underbrace{\|f(A) - f(P)\|_2}_{D(A,P)} - \underbrace{\|f(A) - f(N)\|_2}_{D(A,N)} + \alpha, 0)$$

$$\mathcal{L}(\text{minibatch}_{(A,P,N)}) = \sum_{i=1}^n \mathcal{L}(A^{(i)}, P^{(i)}, N^{(i)})$$

**Attention:** Si les triplets  $(A, P, N)$  sont choisis aléatoirement, la condition peut être facilement satisfaite et nuire à la performance du modèle du fait de l'apprentissage trop "facile". Il est important de créer des triplets "difficiles" à entraîner en utilisant des entités qui se "ressemblent" !

## 9 Réseaux récurrents

### 9.1 Généralité

En cours

### 9.2 Long Short-Term Memory (LSTM)

En cours

## 9.3 Gated Recurrent Unit (GRU)

En cours

## 9.4 Architecture-type

Du fait du comportement sériel de l'architecture récurrente, plusieurs variantes sont réalisables:

- **One-to-One:** Cette architecture est la plus simple et correspond à un réseau récurrent composé d'une cellule uniquement. Ainsi, pour une entrée, le réseau produit une sortie.
- **One-to-Many:** Cette architecture prend en entrée une valeur unitaire et produit une sortie multiple. Elle est exploitée lors d'une tâche de prédiction d'un phénomène sériel à partir d'une valeur donnée (série temporelle par exemple). Dans le cas où l'on veut prédire les valeurs  $t+1, \dots, t+n$ , il faudrait un réseau One-to-N.
- **Many-to-One:** Cette architecture est l'inverse du *One-to-Many*. À partir d'un ensemble sériel, on cherche à prédire une unique valeur. On peut exploiter cette approche lorsque l'on veut prédire une valeur  $t+1$  à partir d'un contexte formé des valeurs  $t-n, \dots, t$ .
- **Many-to-Many:** Cette architecture est caractéristique d'un *Encoder-Decoder*. L'objectif est de prédire une succession d'états à partir d'une succession d'états. La dimension de l'entrée peut être différente de la dimension de sortie (M-to-N). Cette architecture est à la base des modèles de *Natural Machine Translation* car elle permet de s'émanciper de la contrainte de la continuité dimensionnelle entre l'entrée et la sortie (i.e N=M).
- **Many-to-Many stricte:** Cette architecture est identique au *Many-to-Many* mais impose une continuité dimensionnelle entre le signal d'entrée et le signal de sortie. Cette approche est souvent représentée comme la version naïve du *Many-to-Many*, notamment pour les tâches de traduction.

Les différentes architectures sont illustrées sur la Figure 55. Elles sont indépendantes de la structure de la cellule choisie (LSTM, GRU ou RNN vanilla).

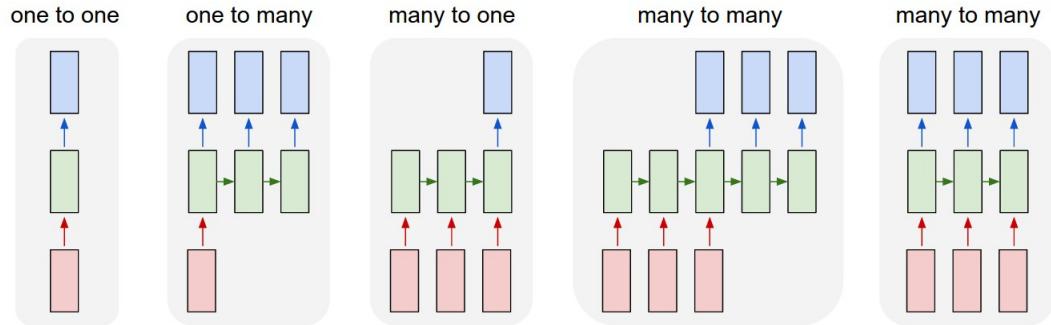


Figure 55: Architecture-type d'un réseau récurrent

## 10 Deep Learning et Attention

### 10.1 Généralités

**Attention:** Cette partie va se reposer sur des architectures avancées de Deep Learning. Afin de comprendre le contexte, il est nécessaire d'avoir une connaissance générale de l'architecture des *Neural Machine Translation* (Section ??), des *Encoder-Decoder* (Section 6) et des réseaux récurrents (Section 9), particulièrement les conventions d'écriture matricielle.

Pour réaliser une tâche, un réseau de neurones n'a pas nécessairement besoin de se focaliser sur l'intégralité de l'information pour produire sa décision. Par exemple, lorsqu'une traduction d'un texte est réalisée, le réseau doit se *concentrer* sur le mot qu'il traduit et son contexte uniquement. Dans le cas contraire, le risque majeur est qu'il n'arrive pas à isoler la partie discriminante nécessaire à sa prédiction et de ce fait, que sa prédiction soit mauvaise.

Ce comportement est représentable en Deep Learning grâce au procédé nommé **Attention**. L'*Attention* permet au réseau de se focaliser sur un sous-ensemble de l'information afin de mieux cibler l'information nécessaire à la tâche qu'il réalise. L'*Attention* est responsable d'une amélioration significative des réseaux de neurones, notamment pour la traduction automatique où il apporte une solution élégante à la contrainte dimensionnelle du réseau traducteur. En effet, il a été montré que plus un texte est volumineux, plus le réseau se doit d'être volumineux[10][88], ce qui est très problématique pour les contraintes matérielles actuelles. La qualité de la prédiction suit la courbe décrite par la Figure 56 lorsqu'un réseau de taille fixe voit la dimension de sa donnée d'entrée augmenter. L'*Attention* permet de conserver une performance stable malgré l'augmentation de la dimension de la donnée d'entrée pour une architecture fixée et permet ainsi, de s'émanciper de la condition de proportionnalité entre dimension du réseau et dimension de l'entrée.

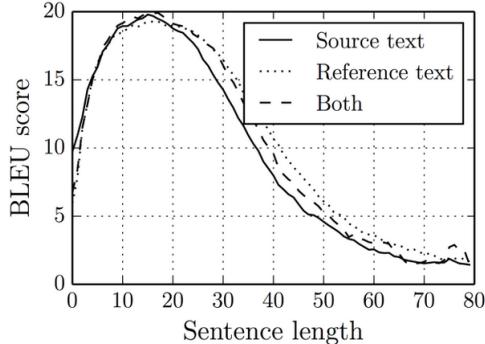


Figure 56: Performance de la traduction selon la dimension du texte d’entrée

Formellement, l’*Attention* consiste à construire dynamiquement une *information de contexte* (noté  $c(t)$ ) à partir d’un ensemble de contextes locaux issus d’une donnée source (notés  $h_{i \in [1, n]}$ ). Ainsi,  $c(t) = f \circ g(h_{i \in [1, n]}, q(t))$  avec  $f$ , fonction d’*Attention*,  $g$ , fonction de proximité et  $q(t)$ , contexte d’état auquel les différents contextes locaux sont comparés. La détermination de  $c(t)$  peut être issue d’une approche déterministe ou probabiliste.

L’*Attention* est un procédé très populaire depuis 2017 et voit son évolution rapide. Néanmoins, deux grandes familles d’architecture sont discernables: les approches *Hard Attention* et *Soft Attention*.

## 10.2 Soft Attention

Les approches *Soft Attention* (Figure 57) reposent sur une hypothèse déterministe qui exprime le *vecteur de contexte*  $c(t)$  sous la forme d’une somme pondérées des différents contextes locaux  $h_{i \in [1, n]}$ . Ainsi,

$$c(t) = \sum_{i=1}^n \alpha_i h_i$$

$$\alpha_j(t) = align(h_j, q(t)) = \frac{\exp(e_j)}{\sum_{j' \in [1, n]} \exp(e_{j'})}$$

$$e_j = score(h_j, q(t))$$

Les coefficients  $\alpha_i$  sont obtenus par application de la fonction *Softmax* afin de normaliser leurs valeurs et de rendre possible leurs interprétations probabilistes. La fonction *score(.)* est la fonction qui évalue la proximité entre le vecteur référence et les contextes locaux. De nombreuses méthodes<sup>104</sup> ont été proposées

---

<sup>104</sup>Nous en étudierons certaines par la suite

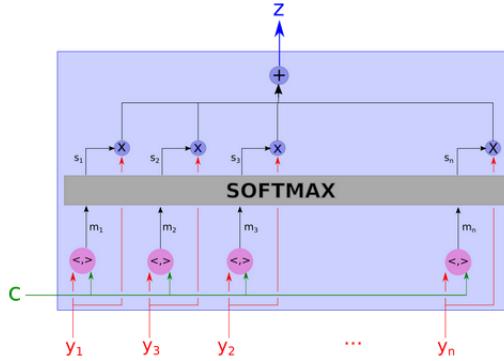


Figure 57: Schématisation de l'approche Soft Attention

à ce jour.

Cette méthode est différentiable. De ce fait, le mécanisme d'*Attention* apprend par *rétropropagation du gradient* comme le reste du réseau. Elle est donc évolutive, souple et permet une approche *End-To-End*. Néanmoins, son impact sur le temps d'apprentissage est non négligeable. De plus, son hypothèse initiale impose que l'*Attention* puisse être convenablement représentée par une combinaison linéaire. Du fait de sa simplicité d'apprentissage et d'implémentation, *Soft Attention* est plus répandue que *Hard Attention*. Néanmoins, les performances sont variables selon les données exploitées.

### 10.2.1 Fonction de proximité

La fonction de proximité, appelée *fonction d'alignement*, permet de calculer la proximité entre le vecteur référence et les différents contextes locaux. De nombreuses méthodes sont proposées mais les plus répandues sont:

1. **Produit scalaire:**  $score(h_j, q(t)) = h_j^T q(t)$
2. **Produit scalaire pondéré:**  $score(h_j, q(t)) = h_j^T W_a q(t)$
3. **Produit scalaire normé**<sup>[96]</sup>:  $score(h_j, q(t)) = \frac{h_j^T q(t)}{\sqrt{n}}$
4. **Couche Feed-Forward**<sup>[106]</sup>:  $score(h_j, q(t)) = v_c^T \tanh(W_c[h_j; q(t)])$

<sup>[96]</sup>Le facteur de normalisation est pertinent en cas de données à forte dimension afin de limiter l'impact d'un gradient faible, ce qui est nuisible à l'apprentissage

<sup>[106]</sup> $v_c^T$  est utilisé afin d'obtenir une valeur unitaire

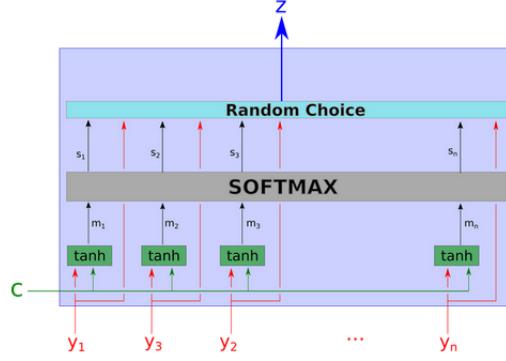


Figure 58: Schématisation de l'approche Hard Attention

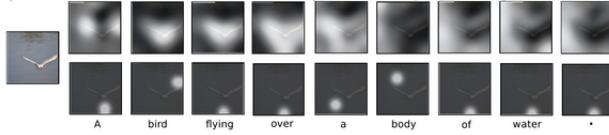


Figure 59: Visualisation du comportement de Soft Attention et Hard Attention

### 10.3 Hard Attention

Les approches *Hard Attention* (Figure 58) reposent sur une méthode de *sampling* stochastique. De ce fait, elles sont probabilistes. Ces approches extraient un<sup>107</sup> vecteur local selon une densité de probabilités. Ainsi  $Z_i \sim h_i, \alpha_i$ .

Du fait de son hypothèse initiale, les valeurs des gradients sont estimées par des *méthodes de Monte-Carlo* et ne peuvent exploiter les gradients rétropropagés car non dérivables. Les méthodes *Hard Attention* sont donc plus difficiles à implémenter et à exploiter au sein du réseau neuronal. L'*Apprentissage par Renforcement* est très utilisé dans le cadre de ce type d'approches. Sur la Figure 59, nous pouvons observer le comportement général des deux types d'*Attention*. Alors que *Soft Attention* a une focalisation diffuse caractéristique d'une somme pondérée, *Hard Attention* se focalise sur une zone spécifique due à la sélection unitaire d'un contexte local. Ce type d'approche est moins exigeant en temps de calculs, ce qui la rend utile pour les tâches temps-réel.

### 10.4 Réseaux récurrents

Un réseau récurrent est caractérisé par une structure temporelle où la mémoire conservée d'un événement  $x$  à l'instant  $t$  tend à diminuer lorsque  $t$  augmente.

<sup>107</sup>Voir deux parfois selon la méthode

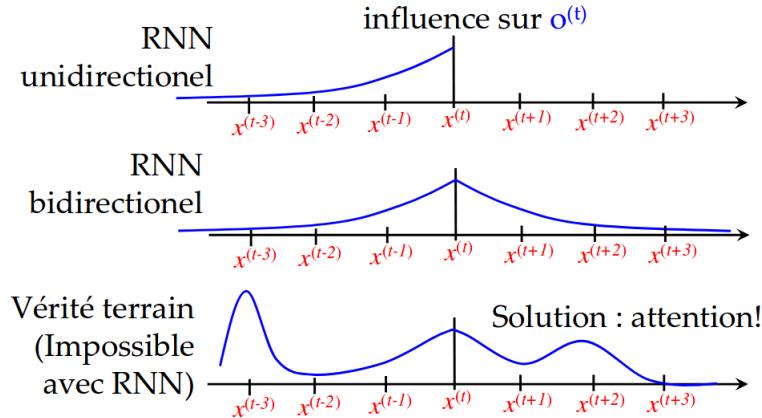


Figure 60: Attention et réseaux récurrents

De même, la projection<sup>108</sup> d'une séquence dans un espace de dimension  $R^d$  impose une destruction d'informations qui peut être critique pour la conservation de son contexte. Pour palier à ce défaut, une méthode simpliste (LSTM Bidirectionnelle) a été de lire les données temporelles pour  $t$  croissant et  $t$  décroissant. Néanmoins, cette solution favorise grandement les données en début de séquence et en fin de séquence. Pour les séries de grande dimension typique des textes, cette méthode est insuffisante pour garantir une prédiction de qualité. L'exploitation de l'*Attention* est, à ce jour, la solution la plus performante pour combler cette problématique (Figure 60).

#### 10.4.1 Approche Bahdanau

L'architecture d'*Attention* selon *Bahdanau*[5] est une méthode de *Soft Attention* démocratisée dans le cadre des *Neural Machine Translation*. Ainsi, les vecteurs de contexte locaux correspondent aux sorties intermédiaires de l'Encoder et le vecteur référence correspond à l'état caché à l'instant  $t$  du Decoder.

L'approche Bahdanau repose sur l'architecture standard d'un NMT. Ainsi, contrairement à une cellule RNN classique qui considère uniquement l'état caché précédent et l'entrée actuelle, une cellule associée à l'*Attention* selon Bahdanau recevra, en plus, un vecteur de contexte  $c(t)$  **dynamiquement déterminé**<sup>109</sup>. La méthode de détermination du vecteur de contexte  $c(t)$  est identique à celle décrite dans la partie 10.2. Ainsi, nous avons:

$$h_{i,\text{classic}} = \phi_\theta(h_{i-1}, x_i)$$

<sup>108</sup>En exploitant l'état caché des cellules du RNN

<sup>109</sup>Dans un NMT classique, le vecteur de contexte est fixe et correspond au dernier état caché de l'Encoder.

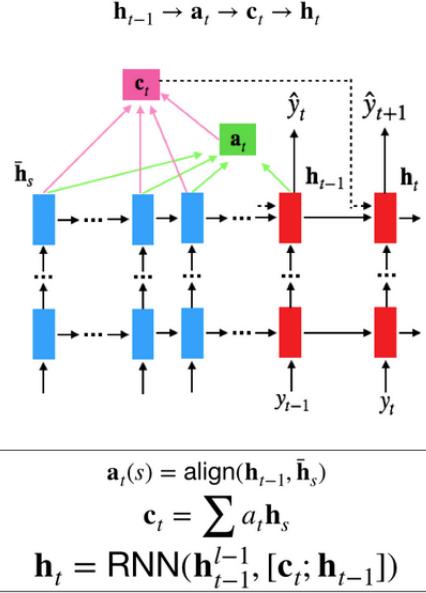


Figure 61: Attention selon Bahdanau

$$h_{i,bahdanau} = \phi_\theta(h_{i-1}, x_i, c_i)$$

Afin d'uniformiser l'entrée, il est nécessaire de ramener à deux entrées uniquement. La méthode traditionnelle est de concaténer deux entrées afin de former un unique vecteur. Néanmoins, l'approche est peu détaillée dans les articles de recherche et le choix des vecteurs est variable selon les sensibilités de chacun.

Ce mécanisme est illustré sur la Figure 61. On peut constater que l'attention appliquée à l'instant t est calculé à partir de l'état caché précédent soit à l'instant t-1. Il n'y a donc pas de considération de l'état actuel (donc du dernier mot à l'entrée du réseau) pour déterminer le contexte.

#### 10.4.2 Approche Luong

L'architecture d'*Attention* selon *Luong*[65] est très similaire à l'approche *Bahdanau*. La spécificité se situe au niveau de la relation entre le vecteur de contexte et l'état caché du Decoder du réseau dans le cadre de la prédiction de la sortie.

Supposons un *Neural Machine Translation*. Nous définissons  $\hat{h}_s$ , vecteurs de contexte locaux issus de l'Encoder et  $h_t$ , vecteur de référence défini par l'état caché du Decoder à l'instant t. Le vecteur  $c(t)$  est le vecteur de contexte obtenu par la somme des vecteurs  $\hat{h}_s$  pondérés par les poids d'attention  $\alpha_s$ .

Supposons  $y_t$ , prédiction du réseau à l'instant t soit un mot dans le cadre d'un

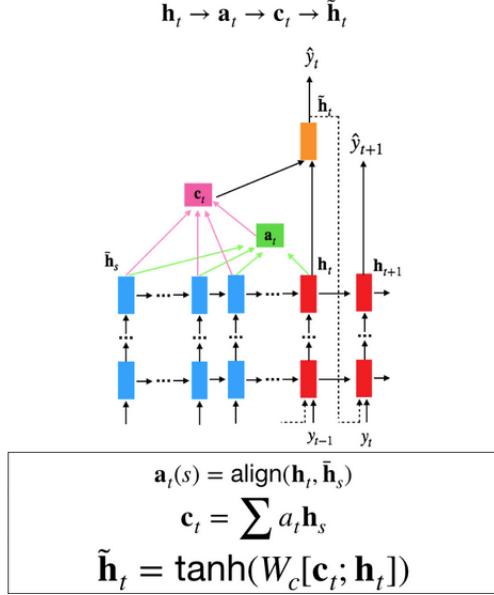


Figure 62: Attention selon Luong

NMT. Dans son architecture standard,  $y_t = \text{softmax}(W_{dict}h_t)$ . L'approche Luong rajoute une couche Full-Connected entre la sortie  $h_t$  et  $y_t$  qui appliquera l'attention. Ainsi:

$$y_t = \text{softmax}(W_{dict}\tilde{h}_t)$$

$$\tilde{h}_t = \tanh(W_c[\mathbf{c}_t; h_t])$$

Ce mécanisme est illustré sur la Figure 62. On peut constater que l'attention appliquée à l'instant  $t$  est calculé à partir de l'état caché actuel soit à l'instant  $t$ . Il y a donc considération de l'état actuel, i.e le dernier mot en entrée du réseau, pour déterminer le contexte.

#### 10.4.2.1 Attention globale et locale

L'*Attention* est dite *globale* lorsqu'elle exploite l'intégralité des vecteurs de contexte. Dans le cadre de vecteurs de grande dimension et/ou de séquences de données de grande taille, la contrainte calculatoire peut être trop importante pour permettre l'exploitation de cette approche, notamment pour le (quasi) temps-réel comme la traduction automatique.

Au contraire, l'*Attention locale* constitue un compromis entre *Soft Attention* (tous les vecteurs) et *Hard Attention* (un seul vecteur) en ne considérant qu'un sous-ensemble des vecteurs de contexte. Ainsi:

$$c(t) = \phi_\theta(\hat{h}_{i,i \in [p_t-D;p_t+D]})$$

avec  $D$ , hyperparamètre défini empiriquement<sup>110</sup> et  $p_t$ , référentiel inféré par le modèle pour chaque mot en entrée du Decoder.

*Luong* propose deux approches pour déterminer  $p_t$ : l'approche *monotonic* et *predictive*.

L'approche *monotonic* repose sur l'hypothèse que la position du mot-cible du Decoder est grossièrement alignée avec le mot source de l'Encoder. Ainsi,  $p_t = t$  et constitue une constante du réseau. Cette méthode est assez peu efficace dans le cadre de la traduction automatique du fait des différences sémantiques et syntaxiques des langues.

L'approche *predictive* apprend à prédire la position du référentiel afin de cibler le sous-ensemble le plus pertinent dans le cadre du mot-cible en entrée du Decoder. Supposons  $S$ , dimension de la séquence d'entrée du Decoder. Il est logique que  $p_t$  soit dans  $[0, S]$ . Ainsi, nous avons:

$$p_t = S \cdot \text{sigmoid}(\varphi(h_t))$$

$\varphi$  est la fonction qui interprète l'état caché actuel pour définir la dimension de la fenêtre déterminée par  $p_t$ . Pour cela, une couche Full-Connected est utilisée. Nous obtenons donc:

$$\varphi(h_t) = v_p^T \tanh(W_p h_t)$$

avec  $v_p$  et  $W_p$ , paramètres du modèle à inférer durant l'apprentissage.

Afin de favoriser l'alignement autour de  $p_t$ , les poids associés à l'*Attention* sont modifiés. Ainsi, plus la position du mot est éloignée du référentiel, plus son importance sera diminuée<sup>111</sup>. Supposons  $\alpha_i$ , poids d'*Attention* du i-ème vecteur de contexte local alors:

$$\alpha_t(j) = \text{align}(h_j, q(t)) \cdot \exp\left(-\frac{(j - p_t)^2}{2\sigma^2}\right)$$

avec  $\sigma$  empiriquement défini à  $\frac{D}{2}$ ,  $p_t$  nombre réel et j, entier appartenant à la fenêtre centrée en  $p_t$ .

#### 10.4.3 Généralisation pour l'exploitation d'image: Image Captioning

La notion d'*Attention* est applicable à toute forme de données structurées. Ainsi, une image, un texte, un signal sonore ou même quelconque peuvent être exploités par une architecture exploitant l'*Attention*. La différence se situe au niveau de

---

<sup>110</sup>Cette détermination peut être dure à réaliser et constitue une faiblesse importante de cette méthode

<sup>111</sup>Néanmoins, un mot éloigné peut avoir un poids important si son alignement est très bon malgré une distance importante.

la détermination des vecteurs de contexte obtenue par le réseau extracteur.

Dans le cas d'un réseau récurrent exploité comme extracteur (notamment pour les *Neural Machine Translation*, les vecteurs de contexte peuvent être associés aux vecteurs d'états cachés des cellules RNN<sup>112</sup>. Néanmoins, dans le cas d'un réseau convolutif (exploité pour l'*Image Captionning* par exemple), il n'y a pas de "structure vectorielle" explicite. Il est donc nécessaire de proposer une méthode d'isolation des différents vecteurs de contexte.

Une image, à travers un réseau convolutif, est explicitée à travers les *feature map* créées par les couches de convolution. De ce fait, l'information d'une image conserve la structure de la donnée initiale soit une forme matricielle. Une sortie d'une couche convulsive est un ensemble de *feature map*. Une même localisation spatiale sur les différentes *feature map* correspond à un résultat d'analyse d'une même partie spatiale de la donnée en amont. De ce fait, elles sont associées à une même entité alors que deux localisations distinctes sur une même *feature map* correspondent à deux analyses de deux parties distinctes de l'image en amont.

Cette spécificité permet d'isoler la structure vectorielle nécessaire à la création des vecteurs de contexte. Supposons une sortie d'une couche de convolution de profondeur D et dont les *feature map* sont *carrés*. Nous avons donc une sortie de la forme  $M \times M \times D$ . Nous obtenons ainsi  $M \times M$  vecteur de contexte de dimension D. Cette séparation permet ainsi de *découper* une image en un ensemble de vecteurs représentatifs de ses caractéristiques. Une contrainte importante est le *choix* de la couche convulsive à exploiter<sup>113</sup>. Le même comportement est applicable sur un signal quelconque car le comportement des *feature map* est indépendant de la nature de la donnée d'entrée. La Figure 63 illustre cette approche.

## 10.5 Réseaux convolutifs

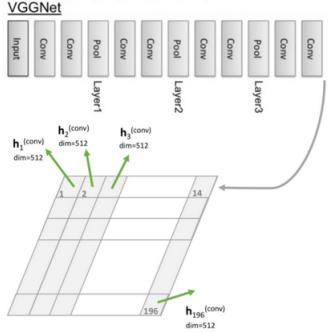
L'*Attention* a été popularisée grâce à sa gestion des *vecteurs de contexte*. Cependant, il est possible d'appliquer ses concepts sur d'autres architectures neuronales, notamment les réseaux convolutifs.

Dans le cadre des RNN, les entités porteuses d'informations sont les vecteurs de contextes obtenus par les prédictions de cellules (souvent LSTM ou GRU) du réseau. L'équivalent dans un réseau convolutif sont les *feature map* obtenues en sortie des couches de convolution. Appliquer l'*Attention* à ce type de réseau revient donc à pondérer l'importance de chacune des *feature map* produites durant l'apprentissage.

---

<sup>112</sup>Ou LSTM, GRU...

<sup>113</sup>Il est logique de favoriser les dernières couches avec un haut pouvoir d'abstraction



La dimension du vecteur de contexte est 512 car la profondeur de la sortie de la 4ième couche d'un réseau VGGNet est de 512. La valeur est différente selon le réseau et la couche convulsive exploitée.

Figure 63: Extraction des vecteurs de contexte à partir d'une feature map

#### 10.5.1 Spatial Transformer

Les données d'apprentissage (et à prédire) peuvent présenter des particularités géométriques variables. Ainsi, l'entité discriminante peut être décentrée au fil des images, son alignement peut être variable selon un angle  $\theta$  ou encore, être représentée à une profondeur variable par rapport au référentiel du cadre de l'image (premier plan et second plan par exemple). Cette variabilité pose de grande difficulté d'apprentissage au modèle tout en forçant l'augmentation de sa complexité pour considérer les géométries variables des images. Afin de résoudre cette problématique, *Spatial Transformer*[40] (STN) propose une approche élégante.

Au lieu de traiter les données *brutes*, *Spatial Transformer* va réaliser une transformation géométrique de la donnée d'entrée (i.e *feature map*) afin de la standardiser. Il réalise donc une action comparable à un pré-traitement qui facilite grandement le travail de discrimination du modèle. Ce module n'a que peu d'impact sur la vitesse du modèle.

Son comportement peut être grossièrement comparé à l'action d'une couche de Pooling. Néanmoins, *Spatial Transformer* possède des avantages qui rend cette approche bien plus robuste. En effet, elle est:

- **Modulable:** Elle est applicable à toute architecture avec facilité.
- **Apprenante:** Elle peut être entraînée par rétropropagation du gradient. Son utilisation garantit donc l'aspect End-To-End du réseau.
- **Evolute:** La transformation réalisée par STN est dynamique selon la donnée d'entrée. Son comportement est donc adaptatif et bien plus robuste qu'une approche statique comme le Pooling par exemple.

### 10.5.1.1 Transformation d'image

Afin de comprendre le fonctionnement du STN, la notion de *transformation linéaire* doit être abordée.

Supposons un point X de coordonnées (x,y). Matriciellement, les coordonnées peuvent ainsi être décrites par:

$$X = \begin{bmatrix} x \\ y \end{bmatrix}$$

Soit M, une matrice 2\*2 définie par:

$$M = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

La transformation linéaire T est définie par la matrice  $X' = T(X) = MX$ .

Ainsi, supposons:

$$M = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

Alors:

$$X' = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} = X$$

Nous pouvons observer que la transformation réalisée est la *transformation identité*. En modifiant les valeurs de la matrice M, nous pouvons réaliser différentes transformations du vecteur représenté par la matrice X. Ainsi, par exemple:

$$\text{Scaling}^{114}: X' = \begin{bmatrix} p & 0 \\ 0 & q \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} px \\ qy \end{bmatrix}$$

$$\text{Rotation}: X' = \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x \cos \theta - y \sin \theta \\ x \sin \theta + y \cos \theta \end{bmatrix}$$

Il est possible de cumuler différentes transformations sur un même vecteur. Supposons 3 transformations définies par les matrices Q, R et S. Nous avons donc  $X' = Q[R(SX)] = MK$ ,  $M = QRS$ .

**Attention:** L'ordre des transformations est significatif car le produit matriciel n'est pas une opération commutative<sup>115</sup>!

Supposons que nous voulons réaliser une *translation* vectorielle. Cette transformation pose problème car, avec le modèle décrit précédemment, chaque dimension du vecteur transformé correspond à une fonction linéaire des composantes

---

<sup>114</sup>On parle d'*isotropic scaling* lorsqu'un même coefficient k est appliqué sur chaque dimension du vecteur.

<sup>115</sup>C'est à dire que  $A * B \neq B * A$

du vecteur initial. Or, dans le cadre d'une translation, nous ne réalisons pas une transformation linéaire. La translation est une transformation standard et très exploitée. Corriger ce défaut est une nécessité dans le cadre du SNT.

Pour résoudre ce dilemme, une solution consiste à exploiter une nouvelle dimension de projection vectorielle pour représenter le vecteur à transformer. Ainsi, supposons X dans  $R^2$  alors nous représenterons X dans  $R^3$  tel que:

$$X = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

De ce fait, nous pouvons construire une matrice représentant une transformation T telle que:

$$M = \begin{bmatrix} a & b & 0 \\ c & d & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Afin de réaliser une translation, nous effectuons donc:

$$X' = \begin{bmatrix} a & 0 & e \\ 0 & d & f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} ax + e \\ dy + f \\ 1 \end{bmatrix}$$

Bien que fonctionnelle, cette approche impose la conservation de la troisième dimension qui ne sert que dans le cadre d'un jeu d'écriture. Afin de corriger ce défaut, il est nécessaire de modifier la matrice M. Supposons M telle que:

$$M = \begin{bmatrix} a & 0 & e \\ 0 & d & f \end{bmatrix}$$

Alors:

$$X' = \begin{bmatrix} a & 0 & \Delta \\ 0 & d & \Delta \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} ax + \Delta \\ dy + \Delta \\ 1 \end{bmatrix}$$

Nous observons que la translation selon  $\Delta$  a bien été réalisée et que l'unicité des dimensions est respectée. Nous avons donc généralisée notre modèle de transformation en permettant la réalisation de *transformations affines*.

#### 10.5.1.2 Interpolation bilinéaire

Le modèle décrit dans le paragraphe précédent permet la réalisation de transformation vectorielle via l'aide d'un produit matriciel. Néanmoins, la structure d'une image a une contrainte importante: les coordonnées doivent être entières car un pixel possède des coordonnées entières. Or, suite à une transformation, les coordonnées obtenues peuvent être décimales. Comment pouvons nous "cibler" le pixel correspondant à des coordonnées non entières ?

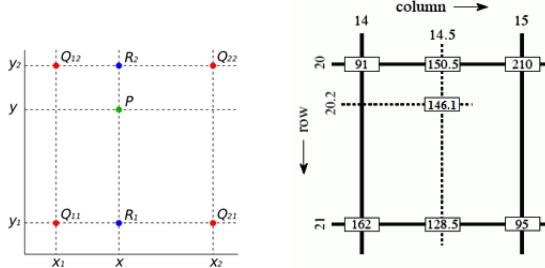


Figure 64: Illustration de l'interpolation bilinéaire

Pour résoudre ce problème, l'*interpolation bilinéaire* est utilisée. Elle exploite les 4 pixels les plus proches afin de définir une valeur pour le pixel "abstrait". Le résultat est ainsi plus lisse et permet la création d'image plus réaliste.

Supposons un point P de coordonnées  $(x, y)$  et 4 points de références tels que  $Q_{11} = (x_1, y_1)$ ,  $Q_{21} = (x_2, y_1)$ ,  $Q_{12} = (x_1, y_2)$ ,  $Q_{22} = (x_2, y_2)$ . Ces points sont représentés sur la Figure 64.

Schématiquement, nous pouvons considérer P comme un point présent sur la surface d'un carré dont les sommets sont pondérés par une valeur propre. Afin d'approximer P, il est donc nécessaire d'évaluer sa distance par rapport à chacun de ces sommets et de réaliser une moyenne pondérée de leurs valeurs selon les distances. Pour cela, nous définirons  $R_1$  et  $R_2$ , comme intensité du pixel intermédiaire entre deux pixels successifs selon l'axe des abscisses. Par la suite, nous définirons l'intensité de P comme valeur d'une moyenne pondérée de  $R_1$  et  $R_2$ . Cette approche permet donc de considérer la position spatiale de P et d'approximer sa valeur.

Nous définissons::

$$R_1 = \frac{x_2 - x}{x_2 - x_1} Q_{11} + \frac{x - x_1}{x_2 - x_1} Q_{21}$$

$$R_2 = \frac{x_2 - x}{x_2 - x_1} Q_{12} + \frac{x - x_1}{x_2 - x_1} Q_{22}$$

Et ainsi:

$$P = \frac{y_2 - y}{y_2 - y_1} R_1 + \frac{y - y_1}{y_2 - y_1} R_2$$

#### 10.5.1.3 Le module SNT

Afin de réaliser une transformation affine d'une image, 3 étapes sont nécessaires:

- **Création du grille d'échantillonnage:** La grille est composée de vecteurs de coordonnées  $(x, y)$  telles que  $x_{max}, y_{max}$  égales aux dimensions de l'image

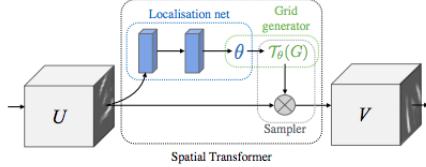


Figure 65: Illustration du module SNT

initiale. En d'autres mots, il y a création d'un *meshgrid* de même dimension que l'image initiale.

- **Application de la matrice de transformation:** La matrice de transformation est appliquée sur les différents vecteurs du *meshgrid* obtenu lors de l'étape précédente.
- **Correction de la grille:** Afin de pouvoir obtenir une image valide, il est nécessaire de corriger les pixels "non entiers". Pour cela, une méthode d'interpolation est nécessaire.

Le module SNT est composé de deux parties: *Localisation Network* et *Grid generator*. Une illustration de son architecture est visible sur la Figure 66.

L'objectif de *Localisation Network* est de produire la matrice de transformation  $\theta$ , i.e une matrice  $(2,3)^{116}$ . Pour cela, un réseau Full-Connected (ou éventuellement convolutif) est exploité.

Ce réseau reçoit la *feature map* en entrée et il produit une sortie de dimension  $(6,)$  correspondant aux paramètres de la matrice de transformation. Ce modèle est capable d'apprentissage du fait de son architecture. Il apprend à réaliser la *meilleure* transformation à appliquer sur une donnée d'entrée afin de favoriser la meilleure qualité prédictive. Il est **important** de comprendre que la transformation réalisée est dynamique et propre aux spécificités de la donnée d'entrée. Le module applique donc différentes transformations selon la donnée. Le modèle n'apprend donc pas à appliquer une transformation unique pour toutes les données mais quelle transformation réaliser selon la donnée. Bien que rien n'impose l'application d'une transformation particulière, nous avons observé expérimentalement que le comportement du réseau est comparable au comportement humain, i.e centrer l'entité discriminante et uniformiser son inclinaison selon les caractéristiques du jeu d'apprentissage. Par exemple, sur la Figure 66, nous pouvons observer le type de transformation dans le cadre du jeu de données MNIST.

La seconde partie correspond au *Grid generator*. Il est chargé de réaliser la transformation affine de la donnée d'entrée selon la matrice de transformation

---

<sup>116</sup>Voir la partie précédente pour justifier l'utilisation d'une telle matrice.

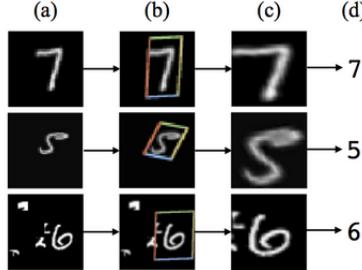


Figure 66: Illustration d'une transformation par SNT avec le dataset MNIST

obtenue par *Localisation Network*.

En d'autres mots, il va produire le *meshgrid* et réaliser le produit matriciel avec la matrice de transformation. Afin de permettre des transformations affines (et le produit matriciel avec une matrice (2,3)), un ajout interne d'une dimension aux vecteurs du *meshgrid* est réalisé<sup>117</sup>. L'action réalisée est donc:

$$\begin{bmatrix} x^s \\ y^s \end{bmatrix} = \begin{bmatrix} \theta_{11} & \theta_{12} & \theta_{13} \\ \theta_{21} & \theta_{22} & \theta_{23} \end{bmatrix} \begin{bmatrix} x^t \\ y^t \\ 1 \end{bmatrix}$$

Cette transformation peut produire des valeurs non entières pour  $x^s, y^s$ . Le module SNT doit donc *uniformiser* les valeurs obtenues. Comme nous l'avons vu précédemment, l'application de *l'interpolation bilinéaire*<sup>118</sup> permet de corriger ce problème. Il est important de savoir qu'il existe d'autres méthodes<sup>119</sup> d'interpolation utilisables. Néanmoins, afin de conserver la capacité d'apprentissage du module (et des couches du réseau en amont), il est nécessaire que la méthode choisie soit pleinement **differentiable** pour permettre la transmission du gradient !

**Remarque:** Au lieu de réaliser un *meshgrid* de même dimension que la donnée d'entrée, il est possible d'augmenter ou de diminuer sa dimension. Cette particularité permet ainsi de faire du *Upsampling* ou du *Downsampling*. Du fait de sa grande efficacité et de son comportement dynamique, SNT est une alternative très sérieuse au Pooling. Néanmoins, SNT est difficilement utilisable dans le cadre d'analyse de données autres que des images. Le Pooling reste donc la référence pour l'analyse du texte ou du signal.

---

<sup>117</sup>Voir la partie précédente pour une explication détaillée de cette particularité

<sup>118</sup>Cet algorithme est pleinement différentiable !

<sup>119</sup>Etudier l'interpolation bi-cubique peut être intéressant.

### 10.5.2 Squeeze-and-Excitation Networks (SENet)

Le modèle SENet[32](2017) est le vainqueur du ILSVRC 2017. Cette approche est extrêmement prometteuse du fait de la simplicité du concept utilisé et des bénéfices qu'il apporte.

Les *feature map* retranscrivent les informations obtenues après l'application d'une couche de convolution sur une donnée d'entrée. Toutes les *feature map* obtenues sont pondérées identiquement. L'idée de SENet est de les pondérer selon leurs capacités explicatives. L'approche utilisée par ce réseau est très simple:

- **Etape Squeeze:** Contraction d'une *feature map* en une seule valeur numérique représentative de cette dernière (via Global Pooling par exemple) et création d'un vecteur combinant l'ensemble des valeurs des features maps. Pour une entrée de profondeur  $P$ , le vecteur de sortie sera dans  $R^P$ .
- **Etape Excitation:** Le vecteur obtenu lors de l'étape Squeeze sera interprété par un modèle Feed-Forward (2 couches<sup>120</sup>) dont la sortie sera de même dimension que l'entrée. Ce vecteur correspondra au vecteur d'excitation des *feature map*. Les nouvelles *feature map* sont obtenues par multiplication de leurs valeurs par le coefficient associé (1 par *feature map*).

La Figure 67 schématisé la différence entre un module ResNet standard et un module ResNet complété par une structure Squeeze-and-Excitation<sup>121</sup>. Le coût de calcul de cette modification est inférieur à 1%, ce qui est considéré comme équivalent. De plus, cette modification peut se généraliser à tout modèle ou structure de bloc. Les auteurs ont montré qu'avec un réseau ResNet-50 avec des SE-blocs, on peut obtenir les mêmes performances qu'un modèle ResNet-101, ce qui est considérable dans le cadre d'une diminution de paramètres et donc de l'optimisation d'un modèle afin de le rendre plus rapide et léger.

#### 10.5.2.1 Concurrent Spatial and Channel Squeeze and excitation

La recherche est active autour de cette architecture. Une amélioration significative, nommée *Concurrent Spatial and Channel Squeeze and excitation* a été proposée par [80]. L'approche initiale pondère les *feature map* les unes par rapport aux autres uniquement. *Concurrent Spatial and Channel Squeeze and excitation* propose une approche réalisant une pondération selon les *feature map* et selon les dimensions internes des *feature map*. Pour cela, deux blocks ont été introduits: sSE et scSE.

---

<sup>120</sup>La sortie de la première couche possède un nombre de sortie minoré par un ratio afin de limiter la taille du réseau

<sup>121</sup>Il est important de noter que l'addition avec l'entrée est réalisée après application de Squeeze and Excitation

La version initiale réalise un GlobalPooling. De ce fait, chaque *feature map* est résumée par une valeur unique. Cette approche considère l'importance d'une *feature map* dans sa globalité et non l'importance de la localisation spatiale des valeurs qui la constituent. Le block sSE (Channel Squeeze and Spatial Excitation) propose de corriger ce défaut en pondérant l'importance de la localisation spatiale au sein des *feature map* selon l'information portée sur la profondeur d'une même position spatiale (différentes valeurs d'une même position spatiale à travers les différentes *feature map*). Pour cela, l'utilisation d'une convolution  $1*1$  est exploitée au lieu d'un GlobalPooling. On obtient donc une matrix de même dimension que les *feature map*<sup>122</sup> de profondeur 1 au lieu d'une valeur unique. Néanmoins, cette approche ne considère plus l'importance d'une *feature map* dans sa globalité mais uniquement l'importance de l'information portée sur une localisation spatiale des *feature map*. Pour palier à ce nouveau problème, le block scSE (Concurrent Spatial and Channel Squeeze and Channel Excitation) a été créé. Ce block est comparable à un modèle inception liant le block original (nommé cSE par [80]) et le block sSe. Les *feature map* pondérées selon les dimensions ou la généralisation sont par la suite additionnées pour obtenir les *feature map* finales. Une illustration de ces deux blocks sont visibles sur la Figure 68.

**Remarque:** Cette approche peut s'appliquer pour tout type de convolution (1D, 2D ou plus). Néanmoins, l'efficacité n'a pas véritablement été démontrée pour une autre donnée que l'image. Il peut être intéressant d'approfondir son analyse dans le cadre du texte, spécialement pour la tâche de classification. En effet, l'analyse d'image exploite des réseaux souvent profonds alors que la classification de texte utilise des réseaux "peu" profonds mais très larges. Cette différenciation est importante car nous ne connaissons pas le lien entre la profondeur et l'efficacité de la méthode Squeeze-and-Excitation.

## 11 L'analyse d'image et ses formes

### 11.1 Les différentes thématiques de l'analyse d'image

L'analyse d'image est un domaine d'application vaste et dont la recherche est très active. Les applications industrielles sont très riches et très diversifiées telles que l'imagerie médicale, l'automatisation des transports (véhicules autonomes) ou encore la robotique.

Les principales thématiques de l'analyse d'image<sup>123</sup> sont:

- **Classification:** Cette thématique est la plus classique. L'objectif est de catégoriser les images selon l'entité qu'elle représente. La classification

---

<sup>122</sup>De même dimension que la source d'entrée

<sup>123</sup>Les noms des thématiques sont relativement flous et inconstants dans la littérature. Il est possible que certaines personnes en utilisent d'autres voire associent des noms à un autre problème. Retenez le contexte des problèmes plutôt que les noms !

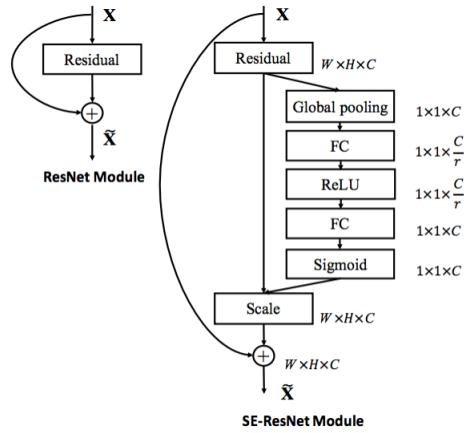


Figure 67: Comparaison entre un module ResNet standard et un module SE-Resnet

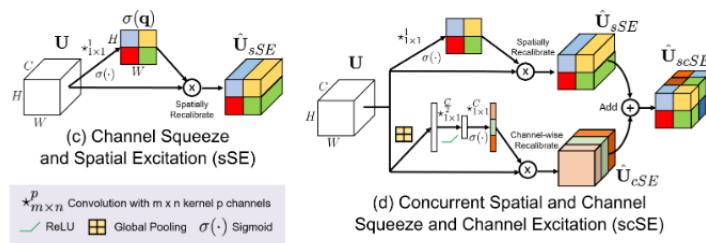


Figure 68: Illustration des blocs sSE et scSE

est non-absolue, i.e l'image est catégorisée comme positive à une classe si une entité caractérisant cette classe est représentée. Néanmoins, d'autres entités peuvent être représentées sur cette image aussi. Par exemple, une image représentant une voiture en ville peut être labellisée "voiture" malgré la présence d'autres entités comme un autre type de véhicule, une personne etc... Généralement, le label définissant l'image est le label défini comme le plus probable. Néanmoins, il est possible d'assouplir cette limitation en ne considérant pas la classe la plus fiable mais les différentes classes dont la prédiction est supérieure à une valeur seuil. Cette tâche est la plus basique et la plus exploitée aujourd'hui. Le prestige des compétitions de classification sont comparables au 100m en athlétisme et constitue la vitrine de l'analyse d'image. En effet, elle mettent en avant les innovations d'architectures neuronales et pose les fondations des évolutions permanentes qui alimenteront par la suite, les autres problématiques. Les réseaux décrites dans la Section 5 sont les solutions State-of-the-art.

- **Object Detection:** Cette thématique cherche à localiser spatialement une classe d'entité sur une image. L'objectif est donc de délimiter une sous-partie de l'image représentant la classe à détecter. Ce type d'analyse est binaire, i.e une surface peut être classée comme positive ou négative. Il n'y a donc qu'un type d'entité détectable possible. La surface classée positivement est souvent délimitée par un rectangle. L'un des problèmes les plus étudiés correspond à la reconnaissance faciale (*Face Recognition*).
- **Object Recognition:** Cette thématique généralise *Object detection*. Elle cherche à détecter de multiples entités correspondant à différentes classes dans une image et à déterminer leurs localisations. Cette méthode est très utilisée par les outils de mesure et de surveillance notamment par les drones et les caméras de surveillance.
- **Object Tracking:** Cette problématique est associée aux contraintes de mouvements et de temps réel. L'objectif est de repérer une (des) entité(s) et de réaliser un suivi en temps réel en gardant l'attention sur ces objets. Cette problématique est associée au support vidéo et peut être vue comme une généralisation de *Object recognition* avec une contrainte de temps-réel. Néanmoins, la considération de la dimension temporelle soulève plusieurs difficultés notamment la définition d'un phénomène temporel défini sur plusieurs images successives ou la variation d'une entité à suivre comme un individu qui porte puis enlève un chapeau. La classification de l'entité n'est pas impératif. Sous contraintes de vitesse, de nombreux algorithmes de *Tracking* néglige la classification. En effet, la présence du phénomène de *tracking* est associée à la présence d'une entité "de valeur"<sup>124</sup>, ce qui est une information souvent suffisante à l'exploitation métier. Le *Tracking* se découpe en deux problématiques principales qui sont le *Tracking* monocible et le *Tracking* multi-cible.

---

<sup>124</sup>On ne sait pas de quelle classe est l'entité précisément mais on sait que c'est une entité d'importance.

- **Semantic Segmentation:** Cette approche cherche à analyser une image au niveau du pixel. C'est-à-dire qu'elle cherche à assigner un pixel à une classe. La discrimination spatiale des entités sur l'image est ainsi bien plus importante. Néanmoins, la différenciation se fait au niveau sémantique et non des instances, de ce fait, seulement une différenciation de classe est réalisée et non des entités-même<sup>125</sup>.
- **Instance Segmentation:** Cette approche est similaire à *Semantic segmentation* mais la discrimination se fait au niveau des entités et non des classes. Ainsi, en reprenant l'exemple d'une foule, l'objectif de *Instance segmentation* est de discriminer l'ensemble des individus de manière unitaire et non un ensemble correspondant à une classe "individus". Cette méthode est très utilisée en imagerie médicale notamment dans l'analyse de cellules cancéreuses.
- **Object Proposal:** Cette approche correspond à une analyse exploratoire. Elle cherche à isoler des sous-parties d'une image susceptible de contenir une entité de valeur.
- **Text Detection:** Cette thématique est associée à la détection de texte dans une image. Elle a pour objectif de détecter, localiser et extraire les entités textuelles présentes sur une image. Cette problématique est souvent associée à la transcription d'un texte manuscrit (en photo) ou de fichiers non éditables caractérisant un texte tels qu'un scan sous un format d'image (.jpeg ou .png par exemple). Une thématique connexe (*Handwriting recognition*) est souvent associée et consiste à unir un texte à un auteur selon les spécificités du style d'écriture<sup>126</sup>.
- **Landmark Detection:** Cette problématique correspond à la détermination de la position d'un point de référence d'une entité et de son orientation par rapport à un système de coordonnée (par exemple, l'étude de points spécifiques déterminant la posture d'un individu (*Pose Estimation*)). Cette problématique est très utilisée par les systèmes autonomes afin d'avoir une meilleure compréhension de son environnement local et des interactions à réaliser. L'exemple populaire de ce type d'outils sont les filtres de photo/vidéo d'applications mobiles telles que Snapchat ou Instagram.
- **Image Captionning:** Cette problématique est associée à la génération d'une description textuelle d'une image. Elle unit l'analyse d'image (Computer Vision) et le Traitement du langage naturel (Natural Language Processing).

Un exemple illustratif de ces différents problèmes est visible sur la Figure 69.

---

<sup>125</sup>Supposons une foule d'individus dense, Semantic segmentation ne sera pas capable de différencier chacun des individus et généralisera à une surface globale classifiée comme "individu".

<sup>126</sup>Cette approche se base quasi-exclusivement sur l'analyse du style graphique d'écriture. Le style syntaxique n'est pas traité.

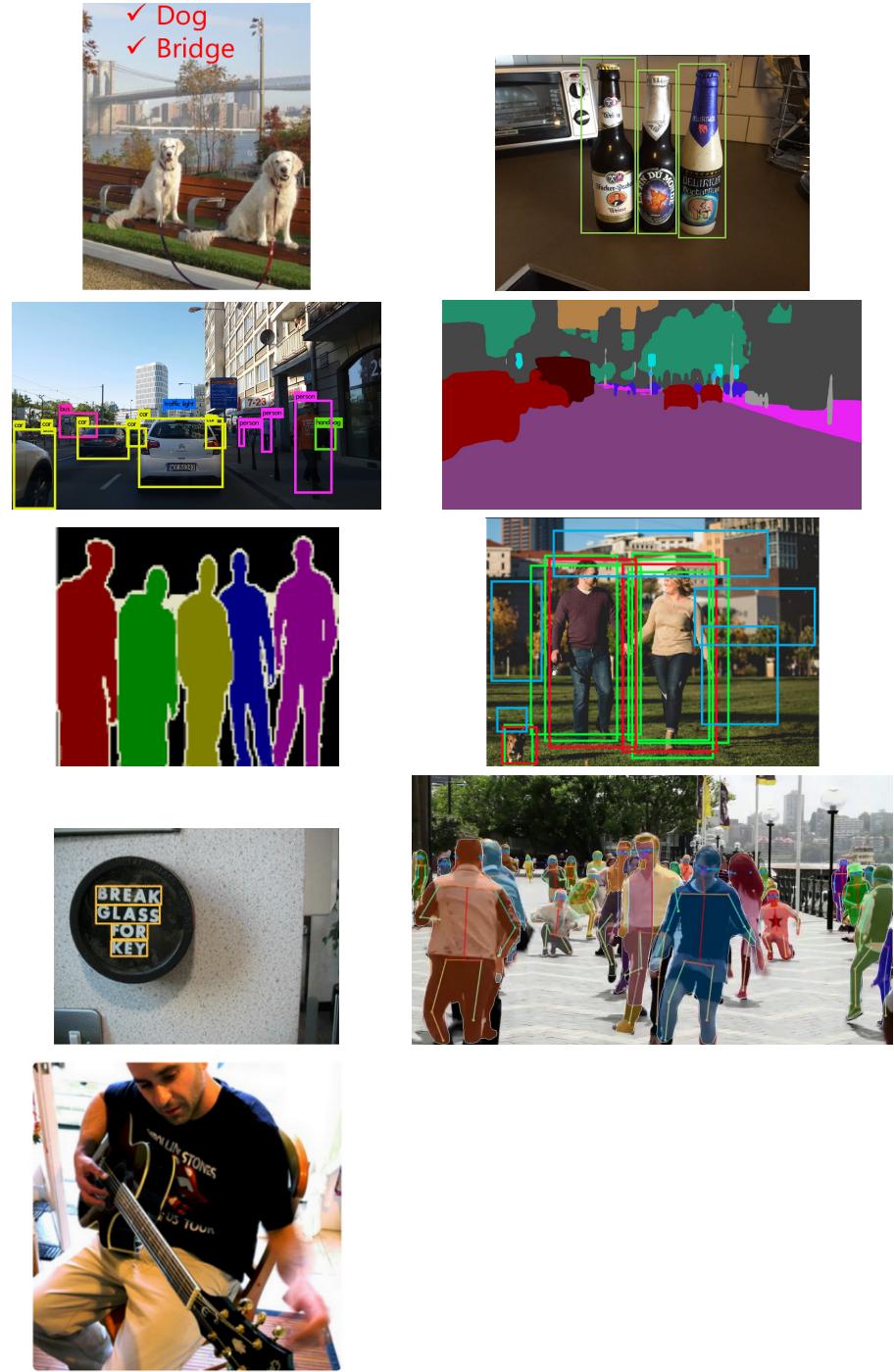


Figure 69: Exemple d'analyse d'image; 1) Classification, 2) Object detection,  
 3) Object recognition, 4) Segmantic segmentation, 5) Instance segmentation, 6)  
 Object proposal, 7) Text Detection, 8) landmark Detection (Pose estimation),  
 9) Image Captionning

## 11.2 Généralités théoriques

### 11.2.1 Théorie - Object Detection, un problème de Régression

Supposons une problématique d'*Object Detection*. L'objectif est donc de détecter la classe d'une image et de localiser l'entité de valeur sur l'image. Pour isoler l'entité de valeur, nous chercherons à réaliser un rectangle qui délimitera sa surface.

Un rectangle peut être défini de plusieurs manières<sup>127</sup>. Dans notre configuration, nous définirons un point qui déterminera le centre  $(b_x, b_y)$  de l'entité de valeur et  $(b_h, b_l)$ , hauteur et largeur du rectangle. Une illustration est visible sur la Figure 70.

L'objectif du réseau prédictif est donc de prédire une classe d'image et 4 coordonnées spatiales  $(b_x, b_y, b_h, b_l)$ . Supposons un réseau qui doit discriminer 3 classes: (1) chien, (2) chat, (3) voiture. La première problématique est de définir la structure de la sortie du réseau (et donc du label des données). Pour rappel, il est nécessaire d'avoir les différentes coordonnées du rectangle délimiteur (bounding boxe) et le label de la classe d'image sachant qu'une image peut ne pas représenter une classe d'objet. Une représentation peut donc être:

Soit  $y$ , la sortie du réseau prédictif tel que:

$$y = [p_c, b_x, b_y, b_h, b_l, c_1, c_2, c_3]$$

- $p_c$ : valeur binaire - Si une classe est représentée,  $p_c=1$  sinon 0.
- $b_x, b_y, b_h, b_l$ : valeur numérique - Coordonnées du rectangle délimiteur où  $b_x, b_y$ , coordonnées du point central de l'entité.
- $c_1, c_2, c_3$ : valeur binaire - Si la classe i est représentée,  $c_i=1$  sinon 0. Seule une classe peut être représentée sur une image.

Supposons une image représentant un chien, alors la sortie du réseau prédictif sera de la forme:

$$y = [1, b_x, b_y, b_h, b_l, 1, 0, 0]$$

De même, supposons une image ne représentant aucun classe, nous obtiendrons:

$$y = [0, b_x, b_y, b_h, b_l, 0, 0, 0]$$

Il est **important** de savoir que dans le cas d'une image non classée, le contenu du vecteur de sortie ne présente **aucune importance** en terme de valeurs exceptée la valeur de  $p_c$ . Les valeurs du réseau peuvent donc être arbitraires, de même que celles du vecteur référence pour l'apprentissage.

---

<sup>127</sup>La manière de représenter le rectangle est déterminée par la méthode employée dans le jeu d'apprentissage. Cette méthode doit être identique pour toutes les données !

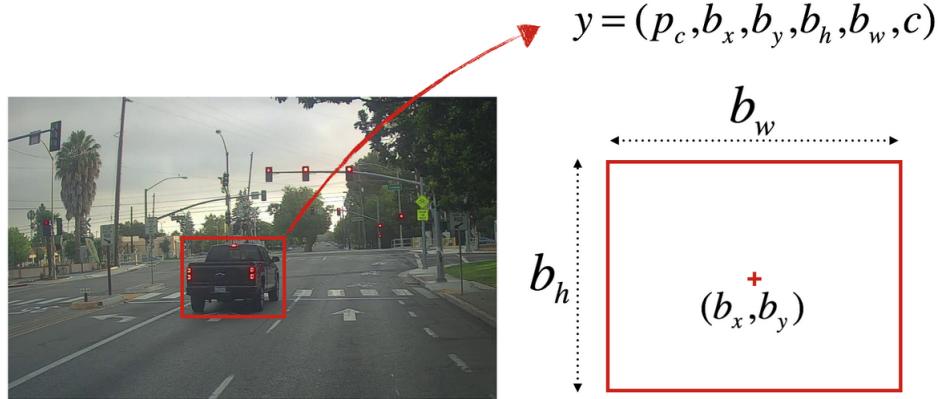


Figure 70: Exemple - Object Detection

Il reste à définir une fonction de coût. Observons que nous avons 3 types de problématiques: le problème bi-classe ( $p_c$ ), le problème de régression ( $b_x, b_y, b_h, b_l$ ) et le problème multi-classe ( $c_1, c_2, c_3$ ). Il est donc nécessaire d'utiliser une fonction de coût qui soit capable de considérer ces trois problèmes. De même, elle doit être capable de considérer la particularité de l'absence de classe sur une image.

Définissons  $y$  comme le label d'apprentissage et  $y'$ , label prédit. Une solution peut être définie par:

Si  $y_{p_c} = 1$ :

$$\mathcal{L}(y, y') = \mathcal{L}_{binary}(p_c, p'_c) + \mathcal{L}_{regression}(\sum b, \sum b') + \mathcal{L}_{Nclasse}(\sum c, \sum c')$$

Si  $y_{p_c} = 0$ :

$$\mathcal{L}(y, y') = \mathcal{L}_{binary}(p_c, p'_c)$$

$\mathcal{L}_{binary}$  doit être capable de mesurer l'erreur d'un problème binaire. *Cross entropy loss* est donc une possibilité. Pour  $\mathcal{L}_{regression}$ , *Mean Squared error* est une possibilité et pour  $\mathcal{L}_{Nclasse}$ , *Negative Logarithmic Likelihood*<sup>128</sup>. Il est possible d'utiliser d'autres fonctions tant qu'elles respectent les particularités de chaque problématique.

### 11.2.2 Théorie - Landmark detection, un problème de Régression

Cette approche est similaire à une généralisation de l'*Object Detection*. Supposons un problème de détection de points de références pour une analyse de visage par exemple. Ces points peuvent déterminer la position des yeux, des

<sup>128</sup>Il ne faut pas oublier le danger du  $\log(0)$  !



Figure 71: Exemple - Landmark Detection

lèvres etc... Un exemple est visible sur la Figure 71.

Au lieu de déterminer un rectangle, cette problématique cherche à déterminer la localisation de différents points soit des couples de la forme  $(pr_x, pr_y)$ . Ainsi, supposons la présence de 128 points de références, il faudra donc 128 couples  $(pr_x, pr_y)$  soit 256 valeurs. Il est important de considérer la possibilité de présence ou non de la classe qu'on cherche à déterminer. On supposera un système qui reconnaît qu'une classe spécifique. Il faudra donc une variable binaire comme vu dans la section précédente. Ainsi, le vecteur de prédiction sera de la forme:

$$y = [p_{classe}, \sum_{128} (pr_{x,i}, pr_{y,i})]$$

### 11.2.3 Sliding Windows

*Object Recognition* se distingue de *Object Detection* par le nombre d'entités détectables sur une image. En autorisant la détection d'un nombre indéterminé d'entités, la dimension de sortie des réseaux neuronaux n'est plus déterminable à l'avance. Il est donc nécessaire de proposer une approche qui permet de résoudre des problèmes de multi-détection en se ramenant à un problème mono-détection. Une solution (historique) s'appelle *Sliding Windows*.

Supposons une image  $I_{raw}$  de dimension  $N*N$ . L'algorithme du *Sliding Windows* consiste à définir une *fenêtre* (window) de dimension  $M*M$  telle que  $M \leq N$  qui isolera une sous-partie de l'image initiale que nous appellerons  $I_{crop,i,j}$  avec  $i,j$  coordonnées du point de référence<sup>129</sup> de  $I_{raw}$ . Nous supposerons que ce point de référence est indiqué par le coin supérieure gauche de la fenêtre. Dans notre exemple, il y a  $N*N$  points de référence. Nous souhaitons que notre fenêtre soit intègre, i.e qu'elle soit contenu dans l'image initiale. Il y a donc  $(N-M+1)*(N-M+1)$  position possible pour notre fenêtre. Ainsi, nous faisons *glisser* notre fenêtre sur les  $(N-M+1)*(N-M+1)$  positions possibles. Une illustration est visible sur la Figure 72.

Nous obtenons ainsi un ensemble d'images caractérisant l'image initiale. Chacune des images est analysée par un réseau prédictif (convolutif) et une classification est réalisée. Le réseau prédictif doit déjà être entrainé sur la tâche à

---

<sup>129</sup>On peut associer un point de référence à un pixel de la matrice d'image

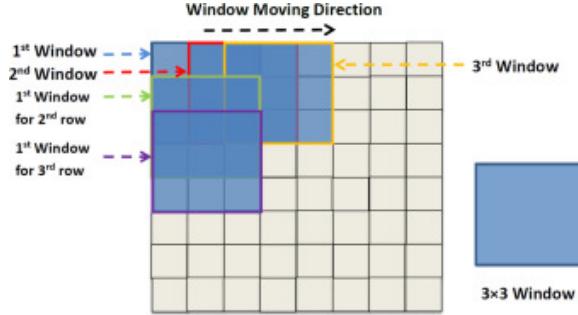


Figure 72: Exemple - Sliding Windows

réaliser et peut être binaire (classification mono-classe) ou multi-classe.

Ce cycle est réalisé avec différentes dimensions de fenêtre (grande et petite) afin de réaliser une discrimination efficace de l'image. De même, le pas de glissement de la fenêtre peut être différente qu'unitaire afin de balayer plus grossièrement l'image à traiter. Une particularité est **importante**. Du fait de la variation de la dimension de la fenêtre, la dimension de l'image à analyser par le réseau diffère. Un réseau de neurones peut traiter une donnée avec une dimension fixe uniquement. Il est donc important de s'assurer que le réseau reçoit une donnée de dimension unique. Deux méthodes sont possibles: exploiter autant de réseaux distincts qu'il y a de dimension possible aux fenêtres. Cette méthode est sans doute la plus efficace mais particulièrement gourmande en temps d'apprentissage. La seconde approche est de réaliser un redimensionnement de l'image avec des méthodes de *Downsampling* ou *Upsampling*.

Dans les faits, cette méthode n'est pas exploitable avec des approches neuronales du fait du temps calculs bien trop élevé. Le nombre d'étape *Forward* est bien trop important à cause de la quantité d'images obtenues par le fenêtrage. Cette méthode est, cependant, encore employée avec des outils de classification plus rudimentaires et donc rapides mais la qualité de la prédiction est diminuée. Malgré tout, l'idée du *Sliding Windows* est encore au coeur des algorithmes à l'état de l'art. Seule son implémentation diffère afin de le rendre exploitable avec des approches neuronales. De plus, *Sliding Windows* présente un problème majeur: la position de ses fenêtres. En effet, les fenêtres sont de tailles fixes et le déplacement de la fenêtre est fixé. Il est donc probable que de nombreuses entités ne puissent être parfaitement localisées. Un exemple de cette limitation est visible sur la Figure 73. Ce type d'approche est donc peu efficace en cas d'entités aux dimensions variables. Il est, cependant, possible d'exploiter l'algorithme du *Sliding Windows* avec différentes dimensions de fenêtre mais le coût en calcul devient rapidement trop important du fait de la multitude de dimension nécessaire pour envisager une détection viable.



Figure 73: Limitation du Sliding Windows

#### 11.2.4 Sliding Windows et réseau convolutif

Dans la Section 5.4, nous avons vu comment convertir une couche Full-Connected en couche de convolution. La méthode *Sliding Windows* est trop gourmande en temps machine comme vu dans la Section précédente. Une implémentation intégralement convulsive est possible pour limiter le coût de calcul de cet algorithme.

Observons la Figure 74. Le réseau du haut présente un réseau convolutif standard. L'image d'entrée est de  $14 \times 14$  (on négligera la profondeur pour l'exemple) et la sortie  $1 \times 1$ . L'intégralité de l'information portée par l'image d'entrée est donc résumée par la sortie de ce réseau.

Supposons maintenant une image de dimension plus grande ( $16 \times 16$ ). En gardant la même configuration de réseau que celui décrit précédemment, on observe une sortie de la forme  $2 \times 2$ . Du fait du comportement de fenêtre glissante des couches de convolutions, l'action de *Sliding Windows* est réalisée par le calcul-même des convolutions. Observons le rectangle rouge de la donnée d'entrée, il correspond à une matrice  $14 \times 14$  (similaire à la matrice d'entrée de l'autre réseau). Nous observons au fil du réseau que l'information portée par cette surface de la matrice est expliquée par le carré supérieure droit de la sortie. En prenant l'exemple du rectangle mauve, il va s'agir du carré inférieure gauche. Nous pouvons donc constater que ce réseau est similaire à l'action du *Sliding Windows* pour une fenêtre carrée de taille 14 et de stride 2 (réalisée par l'étape du Pooling). Cependant, il n'a réalisé qu'une étape Forward pour réaliser ces prédictions. Un gain de temps majeur est donc réalisé en limitant la redondance de calculs.

#### 11.2.5 Region Proposal

Au lieu de se limiter à l'optimisation de calcul du *Sliding Windows*, une approche visant à optimiser le nombre de fenêtres (de dimension **non fixe**) à analyser serait intéressante. Cette approche est exploitée par les algorithmes *Region Proposal*.

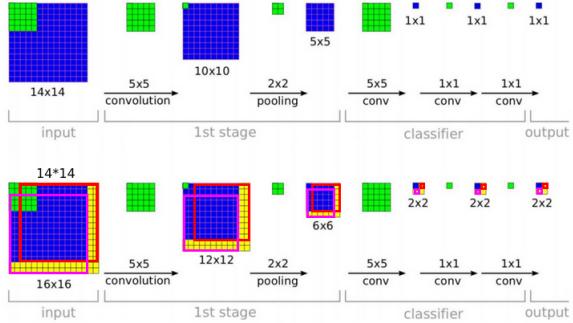


Figure 74: Exemple - Convulsive Sliding Windows

L'objectif de ces algorithmes est de déterminer des régions d'intérêts sur une image, i.e déterminer des *bounding boxes* susceptibles de contenir une entité. Il est **important** de comprendre que ces algorithmes ne réalisent pas de classification. Ils isolent des parties de l'image qui pourraient contenir une entité sans analyser la nature de l'entité isolée. La condition principale de ces algorithmes est d'avoir un grand *Rappel*. En effet, il est possible de filtrer des faux positifs avec les algorithmes de classification. Par contre, si une entité n'est pas isolée par une *bounding box*, elle ne pourra *jamais* être détectée par l'algorithme d'*Object Detection*. Il est donc nécessaire de ne pas être trop sévère sur la création des fenêtres. Ainsi, les *bounding boxes* créées peuvent être bruitées, se superposer mais il est probable que l'une d'elles isolent l'entité de manière satisfaisante.

*Region Proposal* repose sur la notion de **segmentation** d'image. La *segmentation* permet de grouper différentes régions de l'image selon des critères de similarité tels que l'analyse de couleur, de texture<sup>130</sup>... Ces régions unies discriminent des surfaces d'intérêts et permettent de définir des *bounding boxes* de dimension variables et spécifiques à ces zones d'intérêts tout en étant moins nombreuses qu'une recherche exhaustive comme réalisée par *Sliding Windows*.

Il existe différents algorithmes de *Region Proposal*. Le plus populaire et utilisé est **Selective Search**, notamment du fait d'être la méthode employée par des méthodes d'*Object Detection* parmi les plus performantes<sup>131</sup> (R-CNN et Fast R-CNN).

#### 11.2.5.1 Selective Search

*Selective Search* est un algorithme de *Region Proposal* dans le cadre de la détection d'objet. Son objectif est d'avoir un fort *Rappel* et d'être rapide. La

<sup>130</sup>Ces analyses relèvent de l'analyse d'image traditionnelle. Elles ne seront pas approfondies dans ce cours

<sup>131</sup>Ces méthodes tendent à devenir obsolètes avec les dernières avancées



Figure 75: Exemple - Graph-based Segmentation

segmentation de l'image est réalisée par une méthode *graph-based* développée par Felzenszwalb and Huttenlocher[17]<sup>132</sup>. Un exemple d'application de cette méthode est visible sur la Figure 75.

*Selective Search* exploite une image segmentée et réalise une procédure itérative en deux étapes:

- Création des *bounding boxes* correspondant aux différentes surfaces segmentées
- Union des surfaces segmentées qui présentent une similarité élevée

Cette procédure est répétée jusqu'à ce que l'image segmentée ne soit plus modifiée par l'étape d'union ou que l'image présente une unique catégorie. Un exemple est visible sur la Figure 76.

La mesure de similarité de *Selective Search* est une combinaison linéaire de 4 autres métriques sous-jacentes: similarité par couleur, par taille, par texture et par forme. Nous ne détaillerons pas ces métriques, veuillez vous référer à l'article associé.

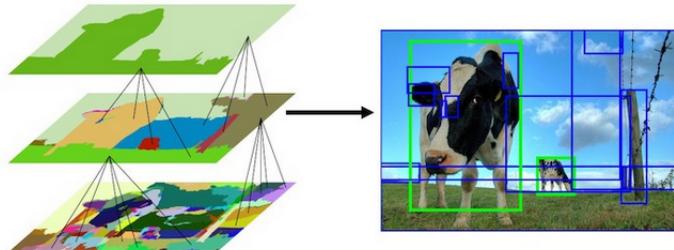
#### 11.2.6 Anchor Boxes

Bien qu'efficace, *Selective Search* est un point faible de plusieurs algorithmes de l'état de l'art. Plus lent que l'étape Forward d'un réseau convolutif, il est le facteur limitant à la vitesse des systèmes actuels (R-CNN et Fast R-CNN). Une solution alternative a été créée: **Anchor boxes**.

**Anchor Boxes** définit des fenêtres prédéfinies avec des tailles différentes (64\*64, 128\*128 par exemple) et des ratios différents (1:1, 2:1, 1:2 par exemple) centrées en un même centre. Une illustration est visible sur la Figure 77.

---

<sup>132</sup>Nous ne détaillerons pas cette méthode. Il faut juste retenir que c'est une technique de segmentation d'image. Vous pouvez lire l'article de recherche indiqué si vous souhaitez approfondir son étude.



La suppression des bounding boxes doublons ou inexacts n'est pas réalisée par Selective Search

Figure 76: Exemple - Selective Search

Supposons l'image d'entrée comme une surface quadrillée selon chaque pixel. Une image 200\*200 serait donc équivalent à un quadrillage 200\*200. Nous définissons nos *anchors* selon l'exemple décrit sur la Figure 77. Il y a donc 9 *anchors* distincts. Un centre d'*anchors* est positionné sur chaque case du quadrillage. Il y aura donc  $200*200*9=360\ 000$  *anchors* distincts. Dans les faits, il n'est pas nécessaire d'imposer une aussi grande sensibilité<sup>133</sup>. Il est donc possible d'appliquer un stride sur le positionnement des centres. Par exemple, supposons un stride de 20. Il y aura donc  $11*11$  centres soit 1089 *anchors* distincts. Bien que ce nombre puisse impressionner, il est faible (ou équivalent selon le cas) comparé aux nombre de fenêtres obtenues par *Sliding Windows* et *Region Proposal*. Grossièrement, nous pouvons donc dire que *Anchor Boxes* réalise le même travail (détermination de *bounding boxes*) que *Region Proposal* excepté qu'il produit des *bounding boxes* de dimensions prévues à l'avance. Ce type d'approche est efficace car il est plus facile d'adapter les dimensions d'une fenêtre que de les prédire intégralement. Ainsi, on peut créer des *anchors* dont la forme est grossièrement adaptée à la forme de l'entité à observer - par exemple, une fenêtre haute et peu large pour un individu - et le réseau s'occupera d'affiner les dimensions pour coller au mieux à l'objet observé.

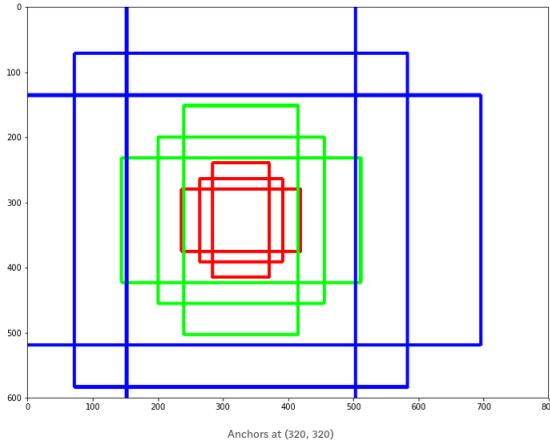
### 11.2.7 RoI Pooling

La création de *bounding boxes* soulève une problématique importante: la dimension variable des fenêtres. Cet aspect est critique car un réseau neuronal ne peut accepter qu'une entrée à valeur fixe. Il est donc important de proposer une méthode d'uniformisation des dimensions des *bounding boxes*.

Une autre condition d'optimisation est caractéristique de cette méthode. En effet, il est tout à fait envisageable d'extraire chaque sous-partie d'une image discriminée par une *bounding boxes* et de réaliser un redimensionnement (crop-

---

<sup>133</sup>Un pixel ne "sépare" pas deux entités distinctes en général...



3 tailles: 128x128, 256x256, 512x512  
3 ratios: 1:1, 1:2, 2:1

Figure 77: Exemple - Anchor Boxes

ping, warping...). Bien que parfaitement fonctionnelle, si on suppose qu'il y a 1000 *bounding boxes* créées, cette technique va imposer 1000 étapes Forward aux couches de convolution suivantes pour extraire les attributs de l'image<sup>134</sup> analysée. En effet, avec cette approche, on considère une sous-partie de l'image d'origine, pas une *feature map* issues d'un réseau convolutif qui a traité cette image. Ceci est très coûteux en temps de calcul. En effet, il est probable que de nombreuses *bounding boxes* se chevauchent. Calculer leurs sorties d'un réseau convolutif reviendrait à réaliser de la redondance calculatoire en réalisant les mêmes calculs plusieurs fois sur une même surface de la matrice de l'image d'origine. *RoI Pooling* propose une approche qui permet de s'émanciper de l'image initiale brute et d'exploiter les *feature map* qui lui sont associées uniquement. Cette particularité est très importante car elle permet de réaliser l'extraction d'attribut via un réseau convolutif **avant** le redimensionnement et non après.

L'algorithme *RoI Pooling* considère une entrée composée des *feature map* obtenues lors de l'extraction d'attributs par les couches de convolution en amont et d'une matrice de dimensions  $N \times 5^{135}$  qui déterminent les *bounding boxes* définies indépendamment (avec *Selective Search* par exemple).

La méthode se divise en 3 parties:

- Isolation d'une sous-partie de la *feature map* en accord avec la *bounding boxes*.

---

<sup>134</sup>Sous-partie de l'image de départ délimitée par une *bounding boxes*

<sup>135</sup> $N$  correspond au nombre de *bounding boxes* prédites

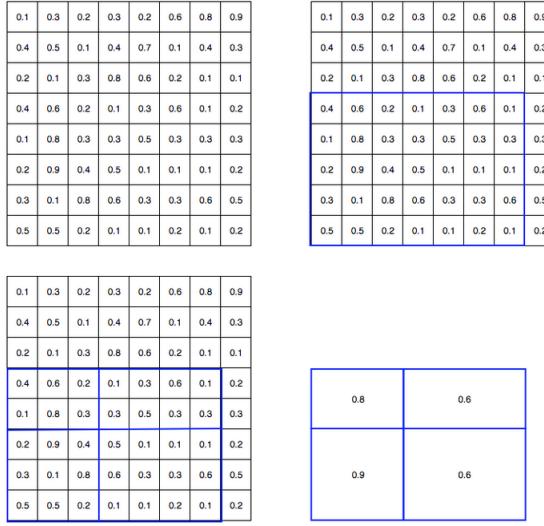


Figure 78: Exemple - RoI Pooling

- Division de la surface délimitée par la *bounding boxes* en parties équivalentes en accord avec la dimension de sortie souhaitée. Par exemple, pour obtenir une sortie de dimension  $2 \times 2$ , il faudra réaliser 4 groupes. Il est possible que ce ne soit pas toujours possible, il n'y a donc pas de condition d'égalité stricte des groupes. La dimension correspondante est donc un **arrondi**. Cette spécificité provoque donc de légers décalages qui n'ont pas de réelles influences pour les problèmes d'*Object Detection* mais nuisent **grandement** pour les problèmes de *Segmentation* qui évalue une entité au pixel-près<sup>136</sup>.
- Sur chaque sous-ensemble, on réalise un Max-Pooling, i.e extraire la valeur la plus élevée.

Supposons que nous voulons uniformiser les dimensions de sortie à  $2 \times 2$  sachant que la dimension des *feature map* est à  $8 \times 8$  et cela, pour toutes *bounding boxes* possibles. Un exemple de cette application de *RoI Pooling* est visible sur la Figure 78.

#### 11.2.8 Intersection over Union (IoU)

Afin d'évaluer la performance des prédictions réalisé dans le cadre des thématiques vues précédemment, il est nécessaire de définir une métrique de performance. En effet, les métriques standards telles que l'Accuracy ou la Précision

<sup>136</sup>Une approche plus conservatrice a été développée pour la Segmentation sous le nom de RoI-Align - Voir Section ??

sont inefficaces pour juger de la validité des *bounding boxes*. Afin de résoudre ce problème, la métrique *Intersection over Union* est utilisée. Une illustration est visible sur la Figure 79.

Supposons une image d'apprentissage. Une *bounding boxes* de référence est définie et l'objectif de notre modèle est de la reproduire. Ainsi, si la *bounding boxes* prédite est strictement identique à la *bounding boxes* d'apprentissage, alors la prédiction peut être jugée comme *parfaite*. Deux *bounding boxes* identiques signifient que leurs coordonnées sont identiques, et de ce fait, il en va de même pour la surface qu'elles englobent. Cette particularité est à l'origine de la métrique IoU.

Considérons l'intersection de deux *bounding boxes*. Une intersection entre la *bounding boxes* prédite et d'apprentissage de même dimension que cette dernière signifie que l'entité a été entièrement détectée. Néanmoins, ce résultat possède une faiblesse majeure car elle ne considère pas la précision de la *bounding boxes* prédite. Ainsi, supposons une image quelconque. Si la *bounding boxes* prédite englobe l'intégralité de l'image, alors l'intersection entre les deux surfaces sera de la dimension de la *bounding boxes* d'apprentissage mais le résultat sera très mauvais. Il faut donc un critère discriminant la précision de la *bounding boxes* prédite. Ce problème est corrigé par la considération de la surface d'union des deux *bounding boxes*. En effet, si les deux *bounding boxes* sont identiques, alors la surface d'union et d'intersection seront identiques. Au contraire, si elles diffèrent, la surface obtenue sera supérieure à la valeur de la *bounding boxes* d'apprentissage.

On obtient donc une relation permettant de considérer ces deux spécificités:

$$IoU(s, s') = \frac{s \cap s'}{s \cup s'}$$

Si les deux surfaces sont identiques, alors  $IoU(s, s')=1$  et tend vers 0 lorsqu'elles diffèrent. Il s'agit donc d'une métrique normalisée, ce qui facilite son exploitation. Il est commun de considérer une *bounding boxes* prédite comme de qualité si  $IoU(s_{ref}, s_{pred}) \geq 0.5$ . C'est la valeur de référence pour la plupart des compétitions officielles exploitant cette métrique.

#### 11.2.9 Non-Max Suppression

Lors de la prédiction des *bounding boxes*, une même entité peut être associée à plusieurs *bounding boxes*. Ce phénomène est illustrer sur la Figure 80. Il est donc nécessaire de filtrer ces différentes prédictions afin de ne garder que les meilleures et associer à une entité, une unique *bounding boxes*. Un algorithme de tri existe et s'appelle *Non-Max Suppression*.

Supposons un problème d'*Object Detection* à 3 classes. Une prédiction de sortie

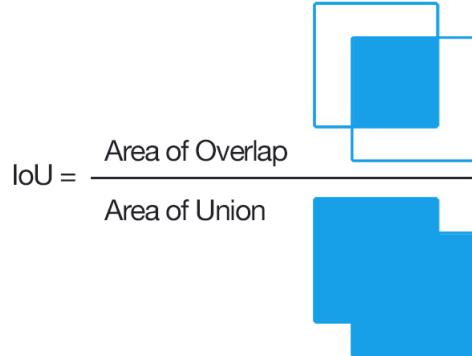


Figure 79: Illustration de la métrique Intersection over Union (IoU)

est donc de la forme:

$$y = [p_c, b_x, b_y, b_h, b_l, c_1, c_2, c_3]$$

Cet algorithme s'effectue en deux étapes. La première est la suppression de toute *bounding boxes* dont la confiance de la détection est inférieure à une valeur-seuil définie (supposons 0.7) donc si  $p_c < 0.7$ . La seconde étape est une procédure itérative qui réalise la sélection de la meilleure *bounding boxes* pour chaque entité-cible.

Supposons tout d'abord un problème mono-classe (il n'y a donc pas de variable  $c_i$  dans le vecteur de prédiction). L'objectif est de supprimer les *bounding boxes*-doublons qui localisent une même entité. Il est donc probable qu'il y est des chevauchements de surface entre chacune de ces *bounding boxes*. L'idée est donc de supprimer les *bounding boxes* qui partagent une trop grande surface soit définir une valeur-seuil pour la valeur de l'IoU. Ainsi, cette partie se découpe en deux phases. Tout d'abord, on choisit la *bounding boxes* avec la confiance la plus importante puis on supprime toutes *bounding boxes* dont l'IoU avec la *bounding boxes* ciblée est supérieure à la valeur-seuil choisie (disons 0.5). Cette étape est ainsi répétée itérativement jusqu'à ce que toutes les *bounding boxes* soit traitées (suppression ou préservation).

Dans le cas d'un problème multi-classe, le cycle est séparé selon la classe prédite par la *bounding boxes*. Ainsi, si un IoU entre deux *bounding boxes* est supérieur à la valeur-seuil (0.5) mais que la classe prédite diffère, alors il n'y aura pas de suppression de *bounding boxes*. On réalise donc le cycle en ne considérant que la classe 1, puis on recommence avec la classe 2 etc... Cette spécificité permet de mieux traiter le cas d'entités de nature différente mais proche spatialement<sup>137</sup>. Un exemple concret est visible sur la Figure 81

---

<sup>137</sup>Par exemple, un cycliste où l'on localise le vélo et l'individu.



Figure 80: Illustration de la problématique de la superposition de bounding boxes

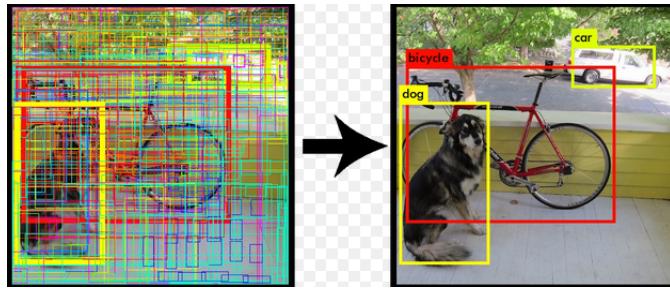


Figure 81: Illustration de Non-Max Detection

Néanmoins, une faiblesse majeure existe. En effet, une hypothèse est faite que deux *bounding boxes* qui se chevauchent et de même classe identifient la même entité. Bien que vrai dans de nombreux cas, il y a d'autres cas où c'est problématique, notamment les nuées denses telles qu'une foule d'individus, une nuée d'oiseaux ou d'abeilles par exemple.

#### 11.2.10 Feature Pyramid

*Feature Pyramid*[61] (FPN) propose une nouvelle méthode d'extraction d'attribut d'une image. En effet, un dilemme existe entre résolution de la *feature map* et l'information sémantique produite. Plus les couches de convolution se succèdent, plus l'information sémantique se développe au détriment de la résolution qui diminue<sup>138</sup>. Ceci est problématique car ça favorise les entités de grande taille sur l'image au détriment des plus petites qui seraient "noyées" par la perte de résolution. Il est donc nécessaire d'apporter une solution afin de conserver l'information portée par les *feature map* de grande résolution.

---

<sup>138</sup>Pas de Padding et application de Downampling récurrente

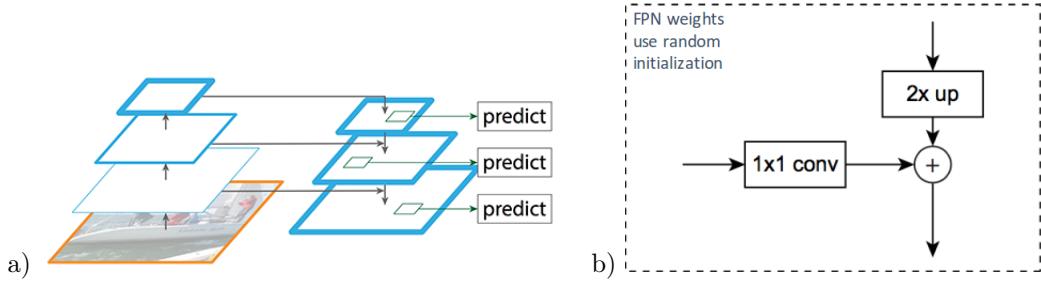


Figure 82: Illustration de l’architecture Feature Pyramid: a) Architecture générale b) Architecture des connexions latérales

*Feature Pyramid Networks* propose une approche pyramidale. Elle se décompose en deux étapes: l’étape *Bottom-up* et l’étape *Top-down*.

- **Bottom-up:** Cette étape est comparable à une succession de couches convolutives standards dont le *stride* augmente de manière à diviser par deux la dimension des *feature map* en sortie. La valeur sémantique des *feature map* augmente au fil des étages de la pyramide mais sa résolution diminue du fait de l’impact des *stride*. Chaque étage peut produire une ou plusieurs *feature map*.
- **Top-down:** Cette étape réalise un *Upsampling* sur les *feature map*. La résolution de l’image est grossière<sup>139</sup> mais l’information sémantique est, quant à elle, forte. Le premier étage de la pyramide (première couche de convolution) n’est pas considéré durant cette étape car trop gourmande en mémoire (et temps de traitement). Chaque étage issu de *Bottom-up* ou *Top-down* traite des données de même dimension deux à deux.
- **Connexions latérales:** L’étape *Top-down* souffre du *Upsampling* qui ne peut retranscrire la résolution de l’image avec fidélité. Pour corriger ce défaut, des connexions latérales sont produites. Une connexion relie deux étages respectivement, i.e l’étage 3 de *Bottom-up* est lié à l’étage 3 de *Top-down*. Une liaison extrait les *feature map* de l’étage correspondant, réalise une convolution 1\*1 afin de contrôler la profondeur et additionne la sortie obtenue avec les *feature map* correspondant sur le cycle *Top-down*.
- **Sortie prédictive:** Les *feature map* issues de l’étape *Top-down* sont soumises à une autre couches de convolution (filtre 3\*3) avant d’être exploité par les couches de détection et classification.

Une illustration de cette architecture est visible sur la Figure 82. *Feature Pyramid* n’est pas un réseau de détection d’entité mais un extracteur d’attributs exploitable par d’autres architectures, notamment par le modèle RetinaNet.

<sup>139</sup>Le *Upsampling* ne peut garantir une mise à l’échelle précise

#### 11.2.10.1 Amélioration de l'architecture du réseau central

#### 11.2.10.2 A faire

[58] <https://arxiv.org/pdf/1804.06215.pdf>

### 11.3 Object recognition: méthodes State-of-the-art

**Attention:** Cet aperçu de l'état de l'art n'est qu'indicatif et propose quelques approches présentant une efficacité reconnue ou une originalité novatrice (et intéressante à étudier). Il est certain qu'il doit exister d'autres méthodes aux performances au moins similaires.

Il existe deux grandes familles d'algorithmes d'*Object Detection: Region based object detectors* et *single shot object detectors*. Ces deux familles ont des algorithmes performants à l'état de l'art bien que leurs approches diffèrent légèrement.

En effet, *Region based object* est caractérisé par le fait qu'un même calcul sur une image puisse être réalisé plusieurs fois au sein d'une prédiction. Par exemple, une même *feature map* peut être exploitée (en partie) plusieurs fois au sein du réseau (lors de la création de la *bounding boxe* et de la classification de l'entité isolée par exemple). Cette tolérance à la redondance de calculs permet de favoriser une précision élevée du réseau mais porte préjudice à sa vitesse. De plus, cette famille d'algorithme est caractérisée par une approche en *bloc*. C'est-à-dire qu'elle accepte l'exploitation d'algorithmes extérieurs et ou de structures parallèles au sein d'un même réseau (par exemple, deux réseaux de neurones). La méthode finale est donc une combinaison de méthodes distinctes. Cette tolérance rend plus difficile l'apprentissage du réseau du fait de la nature différente de ses composants.

*Single shot object*, au contraire, cherche la simplicité et l'unicité. Elle n'exploite une donnée qu'une fois uniquement, ce qui permet une vitesse bien plus importante grâce à la non-redondance de calculs. L'objectif est donc d'unir le réseau de classification et de *Region Proposal* au sein du même réseau. Néanmoins, cette approche tend à rendre le modèle moins robuste bien que les modèles récents s'approchent grandement des performances des *Region based object*. Cette approche tend à donner des réseaux plus *simples* et modulables et semble gagner la bataille en terme de popularité grâce à cette simplicité qui laisse penser qu'elle possède une plus grande marge de progression.

Les méthodes *Region based object* sont principalement *R-CNN*, *Fast R-CNN*, *Faster R-CNN* et *R-FCN*. Les approches *Single shot object* sont *YOLO* et *SSD*.

### 11.3.1 R-CNN - Algorithme précurseur

R-CNN[21] est un algorithme d'*Object recognition* développé en 2014. Son fonctionnement repose sur des méthodes décrites dans la section précédente.

Le réseau se partage en 3 parties indépendantes:

- Création de *bounding boxes*: Cette étape est réalisée par une méthode de *Regional Proposal*. *Selective Search* a été choisi par les créateurs de R-CNN. Il y a une création d'environ 2000 *bounding boxes*.
- Extraction d'attributs: Les sous-images discriminées par les *bounding boxes*<sup>140</sup> sont analysées par un réseau convolutif déjà pré-entraîné. L'entrée de ce type de réseau est de taille fixe. Il est donc nécessaire de s'assurer que chaque image possède la même dimension. Pour redimensionner les images, une approche par *Warping* ou *Cropping* est utilisée.
- Prédiction et affinement: Cette étape réalise 2 actions: Prédire la classe de l'entité présente dans la *bounding boxes* observée<sup>141</sup> et un affinement de la dimension de la *bounding boxes*.

La classification est réalisée par un modèle indépendant. Dans l'architecture originale, un SVM (*Support Vector Machine*<sup>142</sup>) est utilisé. Le raffinement de la *bounding boxes* est réalisée par un réseau Full-Connected qui est chargé de prédire une nouvelle dimension pour la *bounding boxes*. Ce réseau réalise donc une "correction d'erreur" de la dimension par défaut obtenue par l'algorithme de *Region Proposal*. Un exemple illustratif de ce réseau est visible sur la Figure 83.

Un graphique récapitulatif de ce réseau est visible sur la Figure 84. Ce réseau possède des résultats remarquables en comparaison des autres méthodes du moment. Néanmoins, du fait des composants indépendants de son architecture, son apprentissage n'est pas aisé et ses prédictions longues à réaliser. Par conséquent, il ne peut pas être exploité dans le cadre du temps-réel.

### 11.3.2 Spatial Pyramid Pooling - Algorithme précurseur

R-CNN souffre de sa lenteur de prédiction. Cette lenteur est essentiellement due au réseau convolutif appliqué sur chaque *bounding boxes*. En effet, étant donné le recouvrement de nombreuses *bounding boxes*, il y a une redondance de calcul réalisé lors des créations de *feature map*<sup>143</sup>. La proposition de *Spatial Pyramid Pooling Network*[29](SPPnet) est de réaliser l'extraction des *feature map* **avant** l'application de l'algorithme de *Region Proposal*. Ainsi, l'algorithme de *Region Proposal* va s'appliquer sur la *feature map* de l'image et non l'image originale,

<sup>140</sup>Elles représentent une partie de l'image brute initiale

<sup>141</sup>Dans la majorité des cas, aucune classe n'est représentée car il n'y a pas d'entité présente!

<sup>142</sup>C'est un algorithme de Machine Learning utilisé pour des problèmes de Classification

<sup>143</sup>Le filtre de convolution va analyser les mêmes pixels plusieurs fois d'où la redondance

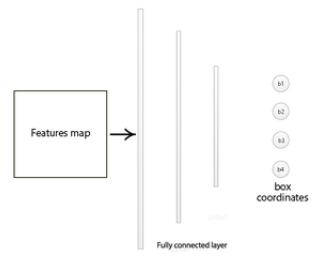


Figure 83: Illustration d'un réseau de prédiction de bounding boxes selon R-CNN

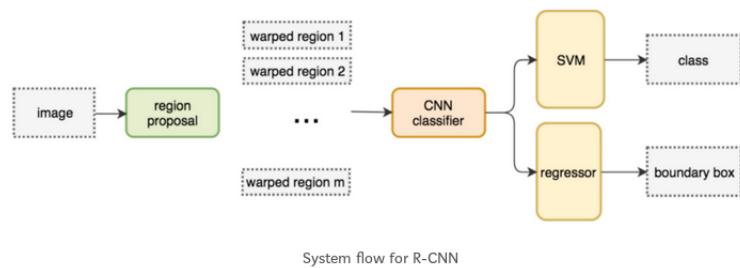


Figure 84: Illustration d'un réseau R-CNN

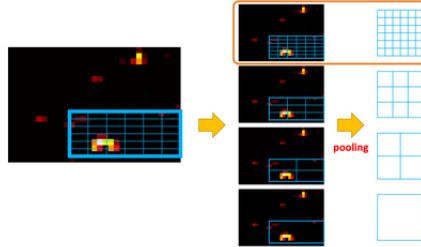


Figure 85: Illustration du Spatial Pyramid Pooling

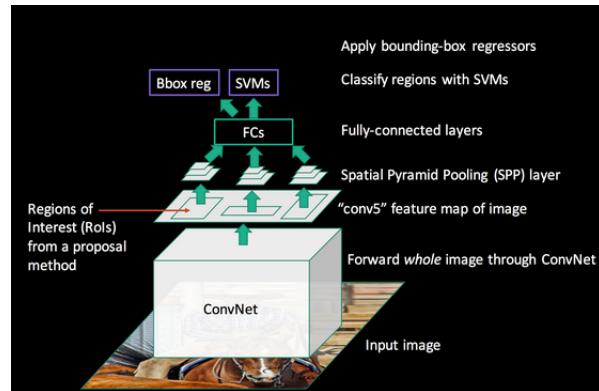


Figure 86: Illustration du Spatial Pyramid Pooling Network

permettant ainsi de réaliser l'extraction d'attributs une fois uniquement.

La problématique d'uniformisation des dimensions reste présente. Afin d'uniformiser les *feature map* issues de l'algorithme de *Region Proposal*, SPPnet introduit *Spatial Pyramid Pooling*. Cette méthode exploite une architecture pyramidale basée sur une action de Pooling où chaque étage de la pyramide réalise un Pooling de dimensions différentes sur la *feature map* initiale. Cette méthode permet ainsi d'avoir une sortie de dimension  $N * K_i$  avec  $N$ , nombre d'étages de la pyramide et  $K_i$ , nombre de valeurs obtenues par le Pooling. Cette valeur varie pour chaque étage, la dimension de sortie n'est donc pas uniforme selon l'axe de  $N$ . Un exemple est visible sur la Figure 85.

L'architecture de prédiction du réseau est similaire à l'architecure R-CNN. L'apport principal de ce réseau est le gain de temps important par rapport à R-CNN pour une qualité de prédiction similaire. Le réseau convolutif doit être appris en amont. L'architecture de SPPnet ne permet pas son apprentissage. La Figure 86 résume l'architure de SPPnet.

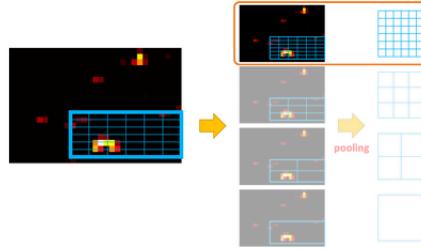


Figure 87: Analogie de RoI Pooling avec Spatial Pyramid Pooling

### 11.3.3 Fast R-CNN - Algorithme précurseur

Fast R-CNN[20], comme son nom l'indique, est une optimisation de R-CNN pour minimiser son temps d'exécution et faciliter son apprentissage. Ces deux améliorations reposent sur 2 spécificités.

Comme SPPnet, Fast R-CNN réalise l'extraction d'attributs par réseau convolutif avant l'exploitation d'un algorithme de *Region Proposal*. L'uniformisation des dimensions est réalisée par *RoI Pooling* (voir Section précédente). Cette uniformisation est comparable à un cas particulier de *Spatial Pyramid Pooling* où la pyramide possède qu'un étage uniquement (voir Figure 87).

La plus-value véritable comparée à SPPnet est le remplacement du SVM par un réseau Full-Connected qui réalisera la classification. Cette modification permet ainsi de réaliser une mise à jour intégrale du réseau (CNN+FC) par rétro-propagation du fait de l'architecture intégralement neuronale. Sans cette modification, le réseau convolutif doit être appris en amont indépendamment du réseau. Seules les couches FC pourraient être entraînées par le biais de la prédiction du correctif des *bounding boxes*. Ce système peut donc exploiter une méthode d'apprentissage unitaire et faciliter son exploitation en unifiant les entités du réseau. L'architecture du réseau et sa structure d'apprentissage sont visibles sur la Figure 88.

### 11.3.4 Faster R-CNN

Faster R-CNN[78] est une amélioration de Fast R-CNN. L'idée derrière cette architecture est de s'émanciper de l'algorithme de *Region Proposal* qui était indépendant du réseau jusque-là (utilisation de l'algorithme *Slinding Search*). Cette dépendance est très problématique car il est le facteur limitant à la vitesse du système. Faster R-CNN propose donc une approche qui intègre dans son réseau-même, une méthode de *Regional Proposal* nommée *Region proposal network*(RPN). Cette architecture repose sur l'approche par *Anchor Boxes* (voir Section précédente) et est intégralement convolutif. La sortie de RPN corre-

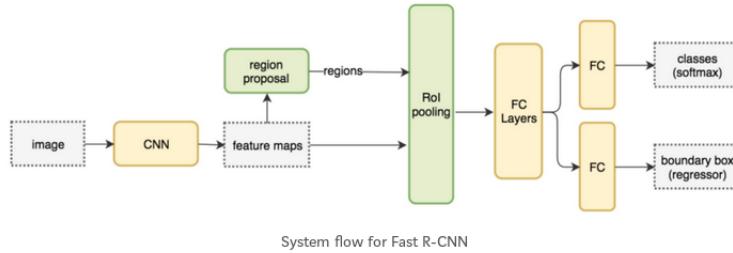


Figure 88: Illustration du Fast R-CNN

spond à la confiance en la présence d'une entité ou non sur l'*anchor* et des coordonnées corrigées de l'*anchors* utilisée. Si il y a 9 *anchors*, il y aura  $K*9*(N+M)$  valeurs de sortie où K, nombre de case du quadrillage, N nombre de variable pour représenter la confiance<sup>144</sup> et M, caractéristiques de la fenêtre<sup>145</sup>.

La partie prédictive du réseau est similaire à Fast R-CNN (RoI Pooling + classification par réseau Full-Connected). Une autre plus-value du système est de créer une méthode de *Region Proposal* propre à un jeu de données. Du fait que RPN apprend sur le jeu de données d'apprentissage de tout le réseau et non indépendamment, il sera parfaitement adapté et spécialisé à la nature des données observées. Il est intéressant de noter qu'il y a un double ajustement de la dimension des *bounding boxes*: lors de l'ajustement de l'*anchor* par le RPN et lors de la classification après l'uniformisation des dimensions par RoI.

Une illustration de ce réseau est visible sur la Figure 89. Il est un modèle avec l'une des meilleures précisions de l'état de l'art. Il reste néanmoins très lourd et lent, ce qui l'empêche d'être exploitable en temps-réel. Néanmoins, Faster R-CNN a été une amélioration remarquable de l'approche proposée par R-CNN. Un comparatif de performance est visible sur la Figure 90. D'autres approches sont préférées aujourd'hui car bien plus rapide et avec une qualité de résultat satisfaisante (telles que YOLO ou SSD par exemple). De plus, une faiblesse de ce type d'approche est la séparation des tâches. La séparation entre le réseau qui produit les *bounding boxes* et le réseau prédictif favorise un coût de calculs important lié à la redondance de calculs entre la phase de création des fenêtres et de la classification (les deux réseaux observent la même feature map) d'où les limites de vitesse de ce type de réseau.

### 11.3.5 Différence principale entre Single shot et Region based

Faster R-CNN, du fait de son architecture entièrement basée sur des réseaux de neurones, se rapprochent de l'architecture *Single shot*. Néanmoins, une dif-

<sup>144</sup>En général, il n'y a qu'une valeur en sortie

<sup>145</sup>En général, il y a 4 valeurs en sortie

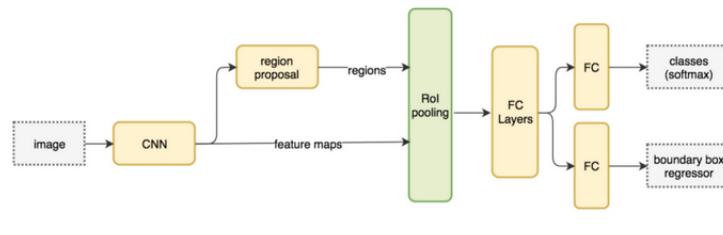


Figure 89: Illustration du Faster R-CNN

	<b>R-CNN</b>	<b>Fast R-CNN</b>	<b>Faster R-CNN</b>
Test time per image (with proposals)	50 seconds	2 seconds	<b>0.2 seconds</b>
(Speedup)	1x	25x	<b>250x</b>
mAP (VOC 2007)	66.0	<b>66.9</b>	<b>66.9</b>

Figure 90: Comparatif de performance entre R-CNN, Fast R-CNN et Faster R-CNN

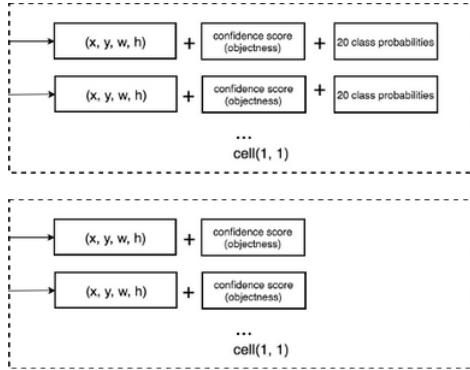


Figure 91: Différence de la sortie des couches convolutives entre les approches Single shot et Region based

férence majeure est encore présente et repose sur son approche de l’algorithme *Anchors Box*.

Supposons la création de 9 *anchors*, La sortie du RPN associée à Faster R-CNN sera de la forme  $K*9*(N+M)^{146}$ . N (souvent égal à 1) et M (souvent égal à 4) sont associés à la confiance en la présence d'une entité et à la dimension de la *bounding boxe*. La classification de l'entité est donc complètement ignorée, la sortie se limitant à dire sa confiance en la présence d'une entité sans discrimination. Cette sortie est donc envoyée à un autre "sous-réseau" pour la classification.

Au contraire, dans le cas d'un algorithme *Single shot*, la sortie serait de la forme  $K*9*(N+M+C)$  où  $C$ , probabilités de classification pour une classe donnée. Cette sortie est donc associée à la fin de la prédiction et donc du réseau prédictif. Il n'y a pas de transfert à un autre sous-réseau de classification. Supposons une *feature map* de dimension  $10*10$  où 20 classes sont à discriminer avec l'aide de 9 *anchors*. On obtiendra donc pour Faster R-CNN, une sortie de dimension  $10*10*9*(1+4)$  et pour un algorithme *Single shot*,  $10*10*9*(1+4+20)$ . Une illustration est visible sur la Figure 91.

Cette différence sur l'aspect prédictif implique une différence majeure: les modèles *Single shot* ont une valeur finie (et identique) de *bounding boxes* définies pour chaque image (en accord avec le quadrillage de l'image<sup>147</sup>) alors que les approches *Region based*, de part la séparation des sous-réseaux, permettent la création d'un nombre indéfini de *bounding boxes*. Cette spécificité des modèles *Region based* permet généralement d'obtenir un meilleur taux de détection car le *Rappel* est plus important. Néanmoins, ces modèles sont plus lourds, plus

---

<sup>146</sup>Voir section 11.3.4 pour plus de détails

<sup>147</sup>Cette particularité est expliquée dans les sections qui vont suivre

lents et plus sensibles aux problèmes de distributions fortement déséquilibrées entre les classes (notamment entre les classes associées à une entité et la classe générale qui traduit l'absence d'une entité).

### 11.3.6 You Only Look Once (YOLO)

Les méthodes précédentes séparent le réseau de *Region Proposal* du classifier. Une autre approche proposée par YOLO repose sur l'idée de pouvoir obtenir les *boundary boxes* et les classes directement à partir des *feature map* en un réseau unique. YOLO a été très populaire lors de sa sortie grâce à sa structure *simple* et souple d'utilisation. Aujourd'hui, deux versions ont été réalisées afin d'améliorer les performances de la version initiale. De plus, des versions dites *tiny* du réseau ont été créées pour diminuer l'impact mémoire du réseau et ses exigences matérielles pour réaliser les prédictions. Il s'agit donc de versions idéales pour des systèmes portables et embarqués.

#### 11.3.6.1 YOLOv1

La première version de YOLO[75] repose sur un réseau convolutif finalisé par deux couches Full-Connected. Le réseau va reproduire le comportement suivant:

- L'image d'entrée est divisé selon un quadrillage de dimension  $S \times S$ . Si le *centre* d'une entité de l'image (défini par le centre d'une *bounding boxes*) est présent dans une des cases du quadrillage, alors cette case est responsable de déterminer la classe de l'objet.
- Chaque case prédit  $K$  *bounding boxes*, i.e ses coordonnées spatiales (centre(x,y), hauteur et largeur) et son degrés de confiance en la présence d'un objet au sein de cette *bounding boxes*. Le degrés de confiance n'exprime que la probabilité qu'une entité quelconque soit présente. Il n'y a pas de discrimination de la nature de l'entité à ce niveau. Cette valeur, dans le cas idéal, devrait être exprimé selon:

Si aucun objet est présent:

$$conf(B_i) = 0$$

Si un objet est présent (même partiellement):

$$conf(B_i) = Pr(entite) * IoU_{Pred, True}$$

Avec  $Pr(entite)$ , probabilité qu'une entité soit présente et  $IoU_{Pred, True}$ , IoU entre la *bounding box* créée par le réseau et la vraie à obtenir. Ainsi, cette valeur permet d'évaluer la présence ou non d'un objet et la surface de l'objet isolé. En effet, une entité peut être détectée mais que partiellement ou au contraire, la surface peut être trop grande.

- Chaque case calcule C probabilités de classe  $Pr(Classe_i|entite)$ . Ainsi, toute *bounding boxes* dont le centre est délimité par cette case sera considérée comme représentative de la classe dont la probabilité est la plus élevée et ce, **indépendamment** du nombre de *bounding boxes*. Supposons 5 *bounding boxes* dont le centre est dans la case  $C_i$  représentant la classe i alors ces 5 *bounding boxes* seront classées comme de la classe i.
- La sortie du réseau est donc de la forme:  $S * S * (B * 5 + C)$  avec S\*S, nombre de cases du quadrillage, B, nombre de *bounding boxes* et C, nombre de classes discriminées. Dans la version originale, le quadrillage est de 7\*7 et chaque case est associée à 2 *bounding boxes* (B=2). 20 classes sont discriminées. La sortie du réseau sera donc de la forme 7\*7\*(2\*5+20).
- Les *bounding boxes* sont conservées si la valeur de confiance est supérieure à un seuil. Les *bounding boxes* maintenues sont ensuite filtrées par *Non-Max Suppression*. Une illustration récapitulative est visible sur la Figure 92.

**Important:** Les valeurs prédites du centre de la *bounding boxes* sont normalisées. Cette normalisation est **importante** et nécessaire au bon fonctionnement de YOLO. Le centre de la *bounding boxes* n'est pas prédite de manière *absolue* mais relative à la case à qui elle est associée. Ainsi, les valeurs (x,y) sont comprises entre 0 et 1 relatif à un point de référence défini sur le sommet supérieur gauche par convention. Par exemple, supposons la case (5,5) et supposons un centre idéalement positionné à (5.4, 5.1). La prédiction du réseau ne sera pas (5.4, 5.1) mais (0.4, 0.1). Cette spécificité est capitale pour conserver l'intégrité de l'approche YOLO. En effet, si on réalise une prédiction absolue, il est possible que le centre "quitte" la case associée (par exemple (5.5,6.1)). Il y a donc un conflit entre la nouvelle position du centre et la case responsable de la prédiction. Afin de réaliser cette normalisation, les prédictions suivent la relation suivante:

$$\begin{aligned}x &= \sigma(b_x) + c_x \\y &= \sigma(b_y) + c_y\end{aligned}$$

Avec  $c_x, c_y$  position du sommet supérieur gauche de la case considérée,  $\sigma$ , fonction sigmoïde<sup>148</sup>, (x,y) position réelle du centre et  $(b_x, b_y)$ , position relative du centre (valeurs prédites).

Le réseau de YOLO est un réseau convolutif inspiré d'une architecture de réseau *Inception* finalisée par deux couches Full-Connected (24+2 couches dans la version originale). Un récapitulatif du réseau est présent sur la Figure 94 mais pour une présentation détaillée du réseau, veuillez vous référer à l'article de recherche[75]. Nous n'approfondirons pas le réseau convolutif qui ne présente pas de particularité notable mais il est intéressant d'étudier les couches Full-Connected du réseau. Comme on peut le voir, la première couche possède 4096

---

<sup>148</sup>Elle donne une valeur entre 0 et 1

neurones. Or, la sortie finale doit être de dimension  $7*7*30$ , i.e 1470 valeurs<sup>149</sup>. Il est donc nécessaire de redimensionner la couche. Pour cela, la seconde couche de Full-Connected doit être de dimension 1470 et un redimensionnement de la sortie de cette couche devra être fait afin d'obtenir l'architecture  $7*7*30$ . Une illustration des couches Full-Connected est visible sur la Figure 93

Une des particularités de YOLOv1 est sa fonction de perte. Elle se présente sous la forme d'une combinaison linéaire de *Squared error* basée sur différentes données. Cette fonction considère la sortie du réseau prédictif. De ce fait, l'intégralité des *bounding boxes* seront considérées<sup>150</sup>. Plusieurs spécificités sont à considérer:

- Afin d'augmenter la précision des *bounding boxes*, il est utile de pondérer l'erreur associée à la prédictions de la *bounding boxes* par rapport à la classification de l'entité associée. On va donc créer un coefficient  $\lambda_{coord}$  (supposons  $\lambda_{coord}=5$ ) qui va pondérer l'erreur de dimension par rapport à l'erreur de classification laissée à 1.
- Du fait du nombre important de *bounding boxes* qui ne contiennent pas d'objet, il est important de minorer leurs impacts sur la fonction de coût au risque de voir l'importance des prédictions de *bounding boxes* contenant un objet fortement minorée. En effet, Si l'essentiel des *bounding boxes* sans objet sont bien prédites (c'est souvent plus facile de à réaliser car la discrimination est plus faible pour les *bounding boxes* sans objets) alors une erreur sur une *bounding boxes* contenant un objet sera moins préjudiciable sur le résultat de la fonction de coût car "compensée", ce qui est très problématique. Pour cela, un coefficient noté  $\lambda_{noobj}$  est crée et minorera l'erreur correspondant aux *bounding boxes* sans objet. Supposons  $\lambda_{noobj}$  égal à 0.5.
- Etant donné qu'il n'y a pas de filtrage de *bounding boxes* pour le calcul de l'erreur, il est nécessaire d'isoler la *bounding boxes* qui servira de "référence" pour la prédiction. Afin de la déterminer, on gardera, pour chaque objet identifiée sur l'image d'apprentissage, la *bounding box* possédant le meilleur IoU avec la fenêtre de référence définie sur l'image d'apprentissage. Ainsi, pour 3 entités présentes sur une image, seulement 3 *bounding boxes* seront considérées comme responsable de la prédiction et de ce fait, délimiteur d'un objet. Toutes les autres seront considérées comme négatives. Il y a, bien sûr, un risque important de faux positifs et faux négatifs durant les premières itérations d'apprentissage du réseau.
- La classification d'un objet est pertinent si une *bounding box* isole un objet. Si une *bounding box* n'est pas "responsable" de l'isolation d'un objet, la classe prédite par cette *bounding box* n'a pas d'influence sur la fonction

---

<sup>149</sup>En effet, la notion de quadrillage est représentée par la dimension de la sortie

<sup>150</sup>L'étape de sélection des *bounding boxes* et l'application de Non-Max Suppression ne sont pas considérées

de perte. Il en va de même pour la dimension de la fenêtre définie. Ainsi, pour une *bounding box* non responsable de la détection d'une image, seule la valeur de confiance en la présence d'un objet est considérée alors que pour une *bounding box* qui est responsable de la détection d'un objet, la dimension de la fenêtre et la classification sont considérées.

- *Square Error* est convexe et présente donc une caractéristique intéressante dans le cadre d'une optimisation. Néanmoins, elle favorise la correction des grandes erreurs et non des petites du fait de la puissance. Cet aspect est problématique car elle favorisera la correction des erreurs associées à de grandes *bounding boxes* et tolérera plus facilement les erreurs sur des petites car l'échelle de la taille des erreurs sera "différente". Cet aspect est partiellement corrigé en considérant la racine carré de la valeur à comparer (hauteur et largeur) et non la valeur réelle. Cette imperfection participe à la faiblesse de YOLOv1 à la détection d'entité de petite dimension.

En considérant ces caractéristiques, la fonction de coût est donc de la forme<sup>151</sup>:

- ***bounding boxes responsables*** de la détection d'un objet:

– **Dimension:** Centre et Hauteur/Largeur:

Pour  $\mathbb{1}_{ij}^{obj} = 1$  si la *bounding box* est **responsable** de la détection d'un objet (0 sinon):

$$\begin{aligned}\mathcal{L}_1 &= \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{obj} [(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2] \\ \mathcal{L}_2 &= \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{obj} [(\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2]\end{aligned}$$

– **Classification:** Confiance + probabilités de classe:

$$\begin{aligned}\mathcal{L}_3 &= \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{obj} (C_i - \hat{C}_i)^2 \\ \mathcal{L}_4 &= \sum_{i=0}^{S^2} \mathbb{1}_{ij}^{obj} \sum_{c \in classes} (p_i(c) - \hat{p}_i(c))^2\end{aligned}$$

- ***bounding boxes non responsables*** de la détection d'un objet:

– **Classification:** Confiance:

Pour  $\mathbb{1}_{ij}^{noobj} = 1$  si la *bounding box* n'est **responsable** de la détection d'un objet (0 sinon):

$$\mathcal{L}_5 = \lambda_{noobj} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{noobj} (C_i - \hat{C}_i)^2$$

---

<sup>151</sup>La forme peut impressionner mais elle est facilement compréhensible. Seules les notations sont lourdes !

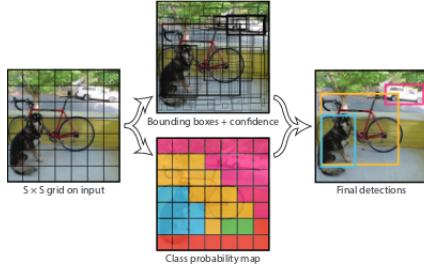


Figure 92: Fonctionnement général de l'algorithme YOLOv1

- **Fonction de perte totale:**

$$\mathcal{L}_{total} = \mathcal{L}_1 + \mathcal{L}_2 + \mathcal{L}_3 + \mathcal{L}_4 + \mathcal{L}_5$$

Cette approche présente trois défauts significatifs. Une *bounding boxes* dont le centre est dans une case sera associée à la classe prédictive par cette case. Cette contrainte spatiale forte est très préjudiciable en cas d'objets de différentes natures proches. En effet, si deux objets distincts sont proches et donc, que le centre des *bounding boxes* associées sont au sein d'une même case, la prédiction de la classe ne sera pas capable de prédire deux classes distinctes, ce qui produira une erreur de prédiction. De même, chaque case définit B *bounding boxes*. De ce fait, seulement B *bounding boxes* peuvent avoir un centre dans une case, ce qui provoque une limite spatiale au nombre d'entités détectables ( $S^*S^*B$ ). Ce modèle est donc peu robuste pour détecter les entités (de même nature) proches et en masse telles que les nuées d'oiseaux ou une foule d'individus par exemple. Pour finir, cet algorithme est très sensible aux dimensions de *bounding boxes* exotiques<sup>152</sup>. Il est très dépendant des *bounding boxes* du jeu d'apprentissage, ce qui peut être dangereux avec un jeu de données faiblement représentatif. Ses capacités de discrimination, du fait de l'architecture du réseau choisi (beaucoup de *downsampling*), tend à être grossières. Un risque de faible généralisation est donc probable, de même que des approximations de calculs de dimensions des *bounding boxes*. Une dépendance au jeu de données d'apprentissage semble donc très (trop) importante malgré une efficacité très correcte de l'algorithme.

### 11.3.6.2 YOLOv2 - YOLO9000

Afin d'améliorer le modèle initial de YOLO, une version 2 a été créée et appelée (sobrement) YOLO9000[76]. Cette amélioration rend le modèle plus rapide, plus précis et généralise la détection à 9000 classes au lieu de 20.

---

<sup>152</sup>YOLO possède de grandes difficultés à détecter des entités de petites tailles

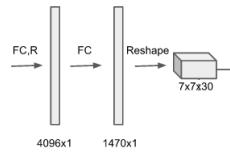


Figure 93: Architecture des couches Full-Connected de l'algorithme YOLOv1

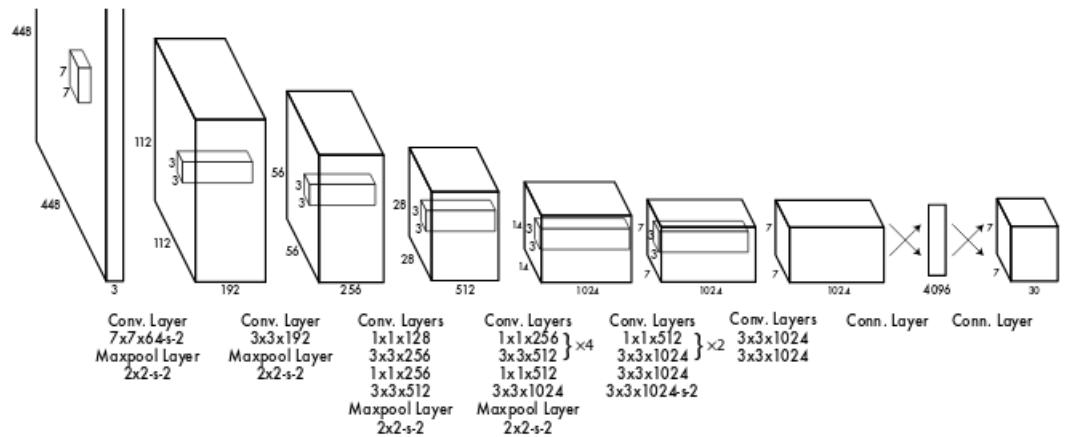


Figure 94: Architecture de l'algorithme YOLOv1

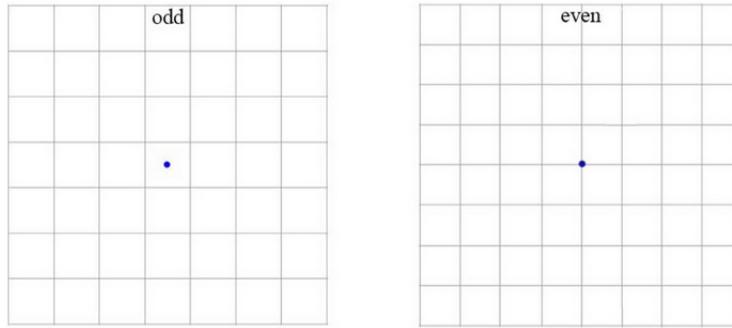


Figure 95: Différence de centre entre un quadrillage pair et impair

YOLOv1 réalise un nombre significatif d'erreur de localisation durant ses prédictions. De même, son *Rappel* (Recall) est trop faible comparé aux méthodes de *Region Proposal* classiques<sup>153</sup>. Afin d'améliorer le modèle, il a donc été préféré d'améliorer la localisation et la détection d'entité au détriment de la classification.

Afin d'améliorer ces caractéristiques, différents ajouts ont été faits. YOLO9000 utilise *Batch Normalization* après chaque couche de convolution. Afin d'être plus robuste dans la détection, un pré-apprentissage<sup>154</sup> à de multiples résolutions est réalisé (224\*224 puis 448\*448). Ceci permet d'améliorer le modèle sur la gestion d'images à "haute" résolution. En effet, YOLOv1 réalisait un apprentissage sur des images 224\*224 avant de se mettre à l'échelle soit 448\*448 mais cette approche ne permettait pas une évolution efficace des extracteurs d'informations des couches convolutives. Un double apprentissage avec YOLO9000 compense cette faiblesse. De plus, la dimension du quadrillage passe de 7\*7 à 13\*13, ce qui permet une bien meilleure discrimination des entités, notamment groupées. Une approche un peu *tricky* a été faite. En effet, 13 étant impair, le quadrillage possède une case centrale. En générale, une image a tendance à posséder une entité en son centre (ou proche). Il est donc intéressant de pouvoir la détecter sans courir le risque d'un décalage associé à un problème de centre d'*anchor* légèrement décalé. une illustration présente le problème sur la Figure 95.

Pour compenser le problème de faible *Rappel*, YOLO9000 va exploiter l'approche par *Anchor boxes* proposée par Faster R-CNN. Cette approche est très bénéfique car elle permet une classification d'entités variées au sein d'une même case du quadrillage. En effet, chaque *anchor* est indépendante et peut prédire une classe distincte. Ainsi, si 9 *anchors* sont présentes sur une même case, alors 9 entités peuvent avoir leur centre localisé sur une même case et être classées

<sup>153</sup>Notamment utilisées par Faster R-CNN

<sup>154</sup>Il s'agit de l'entraînement des couches convolutives d'extraction d'informations avant de faire l'apprentissage spécifique de détection

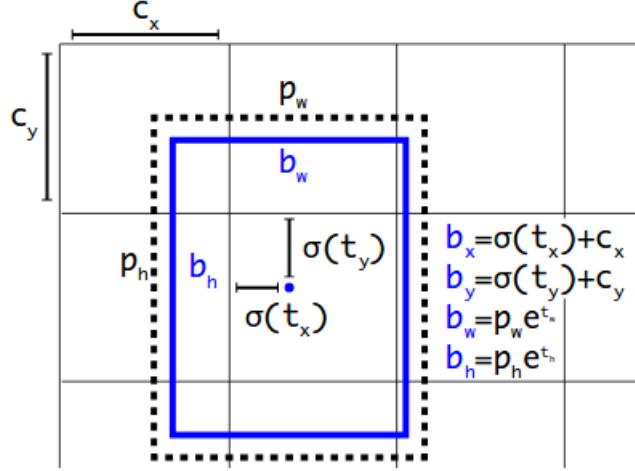


Figure 96: Conversion des prédictions du réseau YOLOv2 selon l'anchor associée

selon des classes différentes. Ainsi, il n'y a plus de prédictions "directes" des dimensions de la *bounding box* mais des prédictions d'*offset* d'une dimension pré-établie. Ce travail est plus facile à apprendre pour le réseau de neurones mais diminue la précision du modèle. La diminution est négligeable alors que le *Rappel* augmente significativement. Dans le cadre de YOLOv2, il y a 5 *anchors* par case.

Tout comme pour YOLOv1, le centre est relatif à la case associée. Par contre, la dimension de la *bounding box* sera relative à la dimension de l'*anchor*. On obtient donc:

$$\begin{aligned}
 x &= \sigma(b_x) + c_x \\
 y &= \sigma(b_y) + c_y \\
 h &= p_h e^{t_h} \\
 l &= p_l e^{t_l} \\
 Pr(\text{entité}) * IoU(\text{box}, \text{entité}) &= \sigma(t_0)
 \end{aligned}$$

Avec  $c_x, c_y$  position du sommet supérieur gauche de la case considérée,  $\sigma$ , fonction sigmoïde<sup>155</sup>,  $(x,y)$  position réelle du centre et  $(b_x, b_y)$ , position relative du centre (valeurs prédictées).  $p_h$  et  $p_l$  correspondent respectivement à la hauteur et à la largeur de l'*anchor*.  $t_h$  et  $t_l$  sont les prédictions associées aux offset des dimensions de l'*anchor*.  $t_0$  est la valeur prédictée de la confiance d'une *anchor* en la présence d'une entité. Un exemple est visible sur la Figure 96.

<sup>155</sup>Elle donne une valeur entre 0 et 1

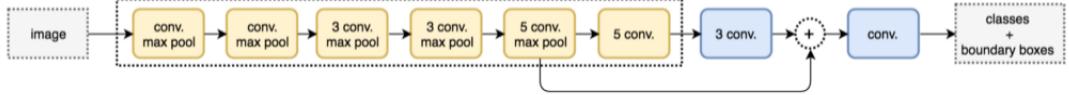


Figure 97: Architecture du réseau convolutif YOLOv2

Les *anchors* sont, à la base, prédéfinies manuellement. Il peut être difficile de définir des dimensions capables de discriminer l'essentiel des entités qu'on cherche à détecter. Bien que des dimensions préférées aient déjà fait leurs preuves dans divers contextes, YOLOv2 fait le choix de définir des dimensions en adéquation avec le jeu de données d'apprentissage. Pour cela, il va utiliser l'algorithme *K-means*[45]. Pour éviter la problématique de tolérance associée aux faibles dimensions, la métrique euclidienne n'est pas exploitabile. Une autre a donc été créée pour corriger ce défaut et repose sur IoU. Elle est définie par:

$$d(box, centroid) = 1 - IoU(box, centroid)$$

Afin de considérer la possibilité de présence d'entités de tailles différentes, YOLOv2 va *additionner* des *feature map* à différents niveaux du réseau afin de considérer différentes échelles de dimensions. Ainsi, un comportement similaire à l'addition à la fin d'un block *ResNet* est réalisée. Un exemple illustratif de cette approche est visible sur la Figure 97. Il est **important** de constater que les couches Full-Connected en fin de réseau ont été remplacées par des couches convolutives, ce qui favorise la vélocité du système.

Par ailleurs, YOLO9000 propose une nouvelle approche afin d'unir différents jeux de données. Dans le cadre de YOLO9000, il s'agit d'ImageNet et COCO. Nous ne détaillerons pas cette méthode qui sort du cadre de ce cours. Pour plus d'informations sur cette méthode, référez-vous à l'article associé [76]. YOLOv2 et YOLO9000 sont strictement identiques si ce n'est la différence associée à la méthode d'union des deux jeux de données et de l'apprentissage associé qui s'en voit légèrement modifié.

#### 11.3.6.3 YOLOv3

La version la plus récente est YOLOv3[74]. Cette amélioration vise à améliorer les performances de prédiction du réseau mais la rapidité de prédiction est diminuée à cause de la complexité grandissante du réseau<sup>156</sup>.

Bien que YOLOv2 ait des résultats meilleurs sur cette thématique, la détection des entités de petite taille reste problématique. Cela est due aux *Downsampling* du réseau qui extrait la sémantique de l'image mais nuit à la "résolution" de l'image, ce qui accentue la perte des informations "discrètes" associées à des

---

<sup>156</sup>La logique de simplicité et d'autonomie initiale du réseau semble ne plus être un critère décisif des chercheurs

entités de petites tailles.

Pour corriger ce défaut, plusieurs modifications ont été faites:

- **Architecture interne du réseau approfondie:** Le réseau YOLO était *primaire*. Relativement peu profond, il n'utilisait pas de composantes caractéristiques des réseaux de l'état de l'art. Avec YOLOv3, cette faiblesse n'est plus d'actualité: ajout de *résidus* par addition et concaténation (*skip-layer*), ajout de *Upsampling*, ajout de profondeur (passage de 53 à 106 couches<sup>157</sup>). Ces particularités apportent différentes plus-values:
  - **Profondeur du réseau:** L'ajout de couches de convolution permet d'améliorer l'extraction d'attributs des images et, de ce fait, la qualité de la prédiction. Néanmoins, l'impact sur le temps de prédiction est à considérer<sup>158</sup>.
  - **Skip-layer et résidus:** Dans les premières couches de convolutions, des *résidus* sont transférés afin de maintenir la bonne cohérence des données dans les différentes couches successives<sup>159</sup>. De même, la présence de *Skip-layer* permet de conserver un bon équilibre entre l'information sémantique obtenue par *Downsampling* et l'information détaillée des *feature map* issues des couches précédentes.
  - **Upsampling:** Le réseau exploite des *feature map* de différentes dimensions. L'utilisation de *résidus* et de *skip-layer* peut être difficile dans les cas d'opérations sur des *feature map* de dimensions différentes. Afin de les mettre à la même échelle, les *feature map* reçoivent un *Upsampling*. La méthode de *Upsampling* n'est pas explicitée<sup>160</sup> dans l'article de recherche. Il est donc nécessaire de la choisir selon votre sensibilité personnelle.
- **Prédiction sur 3 échelles:** Alors que YOLOv1/2 réalisait la détection sur un set unique de *feature map*, YOLOv3 le fait sur 3 sets distincts à des échelles différentes (quadrillage appliqué à l'image de différentes dimensions). Plus le *Downsampling* est réalisé au cours du réseau, plus la sémantique de l'image est extraite. Cependant, on perd les détails de l'image et de ce fait sa capacité de détection des petites entités. Afin de corriger cela, une extraction de *feature map* sur 3 sections différentes du réseau permet de réaliser une détection à 3 échelles différentes et ainsi détecter (en simplifiant grossièrement) les entités de petites, moyennes et grandes tailles. Le fonctionnement suit l'idée apportée par les *Feature Pyramid Networks*<sup>161</sup> (FPN).

---

<sup>157</sup>D'où la plus grande lenteur du réseau comparé à YOLOv2

<sup>158</sup>Tout n'est que jeu de compromis...

<sup>159</sup>Voir ResNet pour plus de détails sur la notion de résidus

<sup>160</sup>A confirmer par une lecture très minutieuse

<sup>161</sup>Pour plus d'informations, voir Section 11.2.10

- **Anchor boxes:** 9 *anchors* sont utilisées par YOLOv3. Ces *anchors* sont réparties sur les 3 sorties prédictives. Ainsi, pour chaque sortie, 3 *anchors* sont réparties et appliquées respectivement<sup>162</sup><sup>163</sup>. Malgré le fait qu'il n'y ait que 3 *anchors* (5 pour YOLOv2), la présence de 3 sorties prédictives à des échelles bien plus précises (dimension du quadrillage supérieure à  $13 \times 13$ ) provoque la création d'un nombre bien plus important de *bounding boxes*. YOLOv3 prédit environ 10x le nombre de *bounding boxes* prédit par YOLOv2.
- **Multi-label:** Il est possible qu'une entité ait plusieurs labels. Par exemple, une femme peut être labellisée *Individu* et *femme*. Avec YOLOv1 et YOLOv2, la classe d'une entité est unique, ce qui est très limitant. Cette limite est due à l'utilisation de la fonction d'activation *Softmax* en sortie prédictive, ce qui produit une prédiction unique (prédiction exclusive). Pour corriger ce défaut, chaque classe est prédite selon une régression logistique<sup>164</sup>, ce qui permet un multi-label pour la classification.
- **Fonction de coût:** La fonction de coût est modifiée pour ses composantes probabilités (confiance et probabilité conditionnelle de classe). L'approche par *Squared error* n'est pas très performante dans le cadre de comparaison de probabilités. Pour corriger ce défaut, l'utilisation de la *Cross-Entropy* permet d'améliorer les performances de cette fonction.

Une illustration du réseau YOLOv3 est visible sur la Figure 98. En considérant la métrique de validation à un IoU de 0.5<sup>165</sup>, YOLOv3 a des résultats qui dépassent SSD et qui rivalisent avec RetinaNet. Cependant, si le seuil du critère de l'IoU augmente, YOLOv3 perd grandement en performances, ce qui traduit que ce réseau manque de précision dans la prédiction de ses *bounding boxes*. De plus, dans les versions initiales, YOLO avait du mal avec les entités de petites tailles. Avec YOLOv3, les performances pour les entités de petites tailles a augmenté mais la prédiction des entités de tailles intermédiaires et grandes a significativement diminué. Un travail important reste à faire sur la précision de la prédiction. Néanmoins, le réseau reste remarquablement rapide comparé à ses concurrents, ce qui fait de YOLOv3, une alternative parfaitement viable et exploitable<sup>166</sup> pour les situations nécessitant une vitesse de prédiction élevée comme le *Tracking*.

---

<sup>162</sup>Les 9 anchors ne sont donc pas appliquées sur toutes les feature map

<sup>163</sup>On peut considérer qu'il y a 3 anchors fondatrices qui sont mises à l'échelle de différentes manières

<sup>164</sup>Comparable à la sigmoïde

<sup>165</sup>Si l'Iou est supérieur ou égal à 0.5, la prédiction est jugée bonne. C'est une convention pour les compétition d'analyse d'images

<sup>166</sup>D'un point de vue métier, les performances sont à la hauteur des attentes. Il est important de savoir que le référentiel n'est pas la performance applicative mais la performance absolue du modèle contre les autres existants, ce qui peut être contre-productif dans le cadre d'une production du modèle au niveau industriel. En effet, un modèle n'a pas à être forcément le "meilleur" au niveau des benchmarks pour être pertinent. De nombreux autres paramètres sont à considérer.

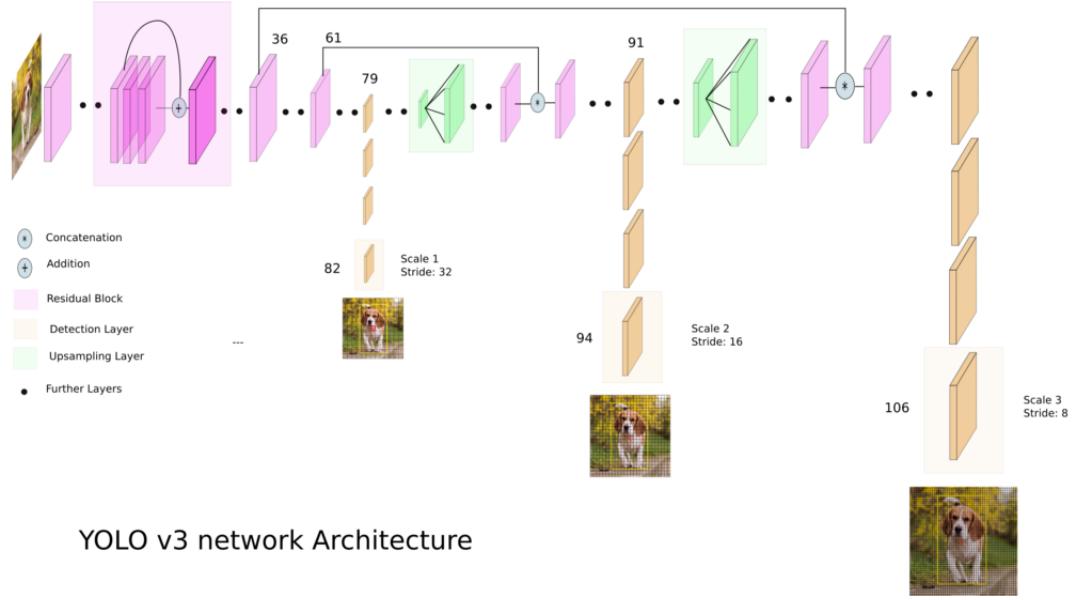


Figure 98: Architecture du réseau convolutif YOLOv3

### 11.3.7 Single Shot MultiBox Detector (SSD)

SSD est un concurrent de YOLO. Longtemps considéré comme plus performant, les dernières versions de YOLO tendent à relancer la concurrence entre les deux approches tant les performances sont du même ordre de grandeur.

#### 11.3.7.1 SSD

Le nom complet du réseau SSD[64] traduit les spécificités de son architecture. En effet, SSD est un algorithme *Single Shot*, i.e il réalise la tâche de classification et de localisation sur une même étape *Forward*. La prédiction des *bounding boxes* repose sur l'approche *MultiBox*[93] et il réalise une tâche d'*Object Detection* d'où *Detector*.

La version original de SSD repose sur l'architecture VGG-16 où les couches Full-Connected ont été supprimé. En effet, nous n'exploitons que les capacités d'extraction d'attributs de ce réseau et non sa capacité prédictive. SSD se démarque de YOLO par son approche de *détection multi-échelle*. En effet, au lieu de réaliser une prédiction sur un seul set de *feature map*, SSD va extraire des *feature map* à différentes échelles et réaliser une prédiction distincte pour chacun de ces sets. L'architecture du modèle prédictif (convolutif) est différente pour chaque set de *feature map*, ce qui contourne la problématique de dimension des *feature map*. Une illustration du réseau SSD est visible sur la Figure 99.

La prédiction des *bounding boxes* suit l'architecture *MultiBox*. Contrairement aux approches standards, *MultiBox* propose une architecture par flux parallèles comparable à l'approche *Inception*. Ainsi, à partir d'un même set de *feature map*, plusieurs prédictions sont réalisées selon des méthodes d'extraction d'attributs variables (couches de convolutions différentes et distinctes). Cette méthode est ainsi plus robuste car elle permet une analyse plus fine des *feature map* et de ce fait, une meilleure prédiction. Un exemple illustratif de *MultiBox* est visible sur la Figure 100.

SSD repose sur l'approche *Anchor box*<sup>167</sup>. Ainsi, chaque prédiction est composée de l'offset de la *bounding box* de référence (*l'anchor*) et des probabilités de classes soit  $(c+4)*k$  avec  $c$ , nombre de classes (contenant la classe *background* qui correspond à aucune classe représentée) et  $k$ , nombre d'*anchor*. SSD utilise 6 *anchors* par case.

La fonction de perte de SSD est de la forme:

$$L(x, c, h, l) = \frac{1}{N} (L_{conf}(x, c) + \alpha L_{loc}(x, h, l))$$

Avec  $L_{conf}$ , fonction d'évaluation de la classification,  $L_{loc}$ , fonction d'évaluation de la localisation,  $N$  nombre de *bounding boxes* conservées et  $\alpha$ , coefficient de pondération entre la classification et la localisation. C'est un hyperparamètre.  $L_{conf}$  repose sur la fonction *Cross-Entropy* pour évaluer la classification et  $L_{loc}$  sur *Smooth-L1* pour la localisation. Pour rappel, *Smooth-L1* est définie par:

$$smooth_{L1}(x) = \begin{cases} 0.5x^2 & \text{if } |x| < 1 \\ |x| - 0.5 & \text{otherwise} \end{cases}$$

L'utilisation de *Smooth-L1* permet une plus grande souplesse d'apprentissage. En effet, il n'est pas nécessaire d'avoir une précision au pixel-près, ce que favorise la norme L2. Cette contrainte peut être trop forte et limiter l'apprentissage sans réel bénéfice pour l'oeil humain et l'exploitation métier. Pour plus de détails sur la fonction de perte de SSD, référez-vous à l'article de recherche [64]. Étant donné le nombre significativement plus grand du nombre de *bounding boxes* négatives, il est nécessaires de limiter leurs utilisations durant l'apprentissage pour conserver un équilibre entre *bounding boxes* positives et négatives. Pour corriger ce défaut, SSD trie les valeurs de la fonction de confiance ( $L_{conf}$ ) pour chaque *anchor* par ordre décroissant et sélectionne ces valeurs de manière à conserver une ration 3:1 entre positifs et négatifs pour l'apprentissage. Cette approche est appelée *Hard Négative Mining*.

Les créateurs de SSD conseille l'utilisation de *Data Augmentation* pour renforcer l'apprentissage et rendre le modèle plus robuste. Leur algorithme de génération

---

<sup>167</sup>Cette approche est récurrente. Nous ne la détaillerons pas ici. Elle est similaire à celle exploitée par Faster R-CNN

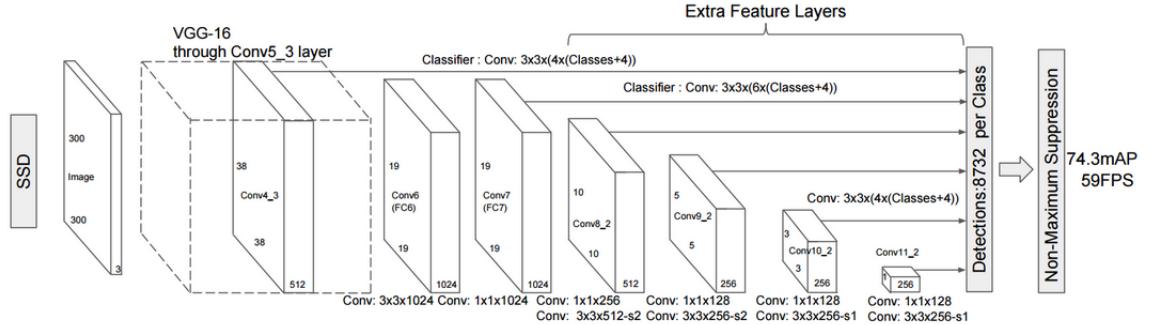


Figure 99: Architecture du réseau SSD

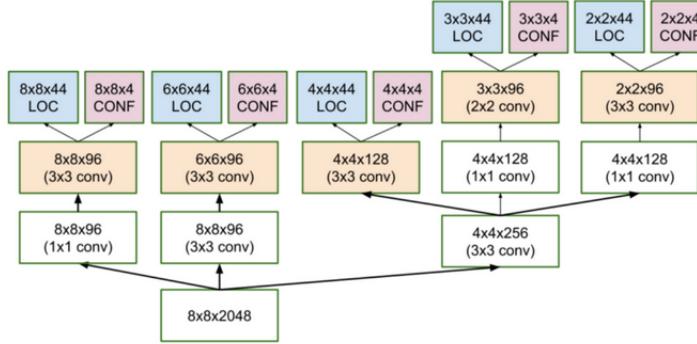


Figure 100: Architecture de Multibox

de données repose sur l'exploitation de la métrique IoU pour créer différentes sous-images conservant une valeur minimum d'IoU. Cette méthode peut être pratique pour apprendre à détecter des entités partielles, détériorées ou de caractéristiques légèrement différentes de celles présentes dans le jeu de données d'apprentissage.

#### 11.3.7.2 Tiny SSD

Les modèles de Deep Learning sont lourds et exigeants en terme de matériels. Cette contrainte est une problématique récurrente pour implémenter ce type de modèle sur des supports de faibles capacités tels que l'embarqué. Pour répondre à cette contrainte, une version plus légère de SSD a été conçue: Tiny SSD: [102].

Ce modèle repose sur l'exploitation de l'architecture *SqueezeNet* et de ses *Fire*

*Module*<sup>168</sup>. Hormis cette particularité, il n'y a pas de nouveauté significative. Pour plus de détails sur l'architecture du modèle, veuillez consulter l'article associée. L'architecture du réseau est le fruit d'une optimisation soutenue entre exploitation des *Fire Module* et des couches de convolution classiques. Il est donc pertinent de suivre à la lettre, le modèle décrit pour une future implémentation.

Ce modèle est remarquable pour l'embarqué du fait de ses performances et de sa taille. En effet, son concurrent direct (Tiny YOLO) est environ 30x plus lourd (60.5Mo contre 2.3Mo) tout en ayant une performance moins élevée (57.1%<sup>169</sup> contre 61.3%).

#### 11.3.7.3 Deconvolutional Single Shot Detector (DSSD)

*Deconvolutional Single Shot Detector*[19] est une amélioration importante de SSD. Ses deux principales innovations sont le remplacement du modèle VGG par un ResNet (Residual-101) et l'utilisation de *déconvolutions* avec application de *skip-layer*.

Le réseau initial de DSSD est comparable au modèle SSD. Sur la dernière couche de convolution du modèle SSD, le *Downsampling* a permis de produire une image avec une forte explication sémantique mais une faible résolution de l'image. Comme décris précédemment, SSD réalise une extraction de *feature map* sur les dernières couches du réseau afin d'obtenir différentes échelles et ainsi améliorer la prédiction. Bien qu'efficace, cette méthode n'exploite pas les capacités explicatives obtenues par les dernières couches de convolution. En effet, les *feature map* sont extraites **avant** la (ou les) dernière couche de convolution du réseau. L'idée de DSSD est de ne pas perdre la capacité explicative obtenue lors des dernières couches de convolution. Il ne va donc pas extraire les *feature map* avant ces couches mais **après**. Pour cela, il est nécessaire d'appliquer une méthode pour redimensionner les *feature map* (*Upsampling*) tout en leur redonnant l'information associée à la résolution de l'image perdue durant l'extraction de la semantique de l'image. Pour cela, DSSD exploite donc la *Déconvolution*<sup>170</sup> pour remettre à l'échelle et des *skip-layer* pour réintroduire l'information de résolution de l'image.

L'action des *skip-layers* est d'additionner les *feature maps* obtenues par *Déconvolution* (porteur de la sémantique) avec les *feature maps* issues de couches précédentes avant *Downsampling* (porteur de la résolution de l'image). Les *feature maps* obtenues possèdent donc les deux informations au lieu d'un compromis que réalisait SSD selon la couche extraite. Une illustration de l'architecture générale est visible sur la Figure 101 et la déconvolution réalisée est détaillée

---

<sup>168</sup>La description détaillée de cette architecture est disponible dans la Section 5.6.10.1

<sup>169</sup>Métrique mAP sur VOC 2007

<sup>170</sup>Ne pas oublier que ce nom est un abus de langage !

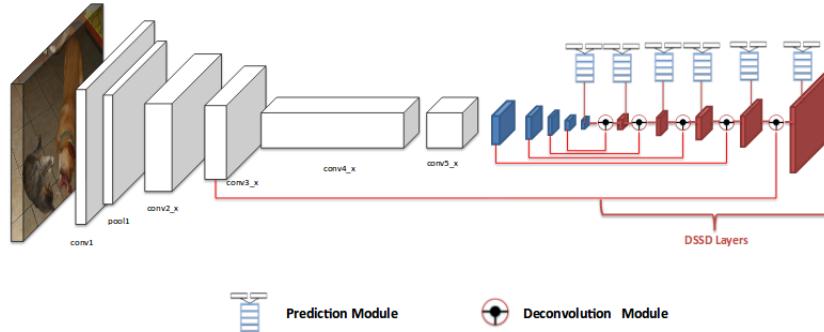


Figure 101: Architecture de Deconvolutional Single Shot Detector (DSSD)

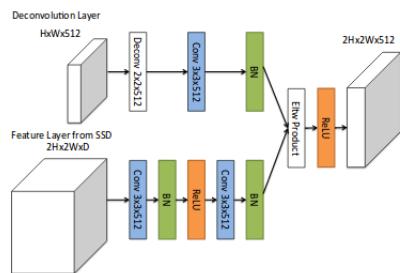


Figure 102: Architecture du Deconvolution module de DSSD

sur la Figure 102.

Les prédictions des *bounding boxes* possèdent toujours les mêmes spécificités que les autres modèles, i.e un problème de régression basé sur la prédiction de probabilités de classes et de dimensions (plus précisément offset) de la *bounding boxes*. La spécificité ajoutée par DSSD est d’exploiter l’architecture *résiduelle* dans son module prédictif. Plusieurs variantes sont proposées par les créateurs de DSSD et sont visibles sur la Figure 103.

Afin de supprimer les *feature map* redondantes, DSSD exploite l’algorithme *Non-Max Suppression* pour réaliser une sélection des *feature map* à conserver. De même, son approche pour la fonction de perte est comparable à DSD et réalise toujours un tri sur les *feature map* à exploiter pour l’apprentissage afin de garantir un ratio 3:1 entre positif et négatif (approche appelée *Hard example mining*).

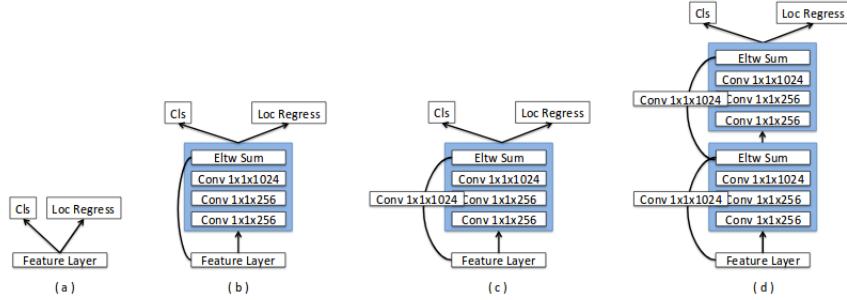


Figure 103: Architecture du module prédictif de DSSD

### 11.3.8 RetinaNet

Une des contraintes principales des modèles d'*Object Detection* est la différence de distribution entre les classifications positives et négatives du fait du nombre très importants de *bounding boxes* proposées pour chaque image. Deux approches sont exploitées aujourd’hui: exploiter une sous-parties des données prédites afin d’exploiter une distribution avantageuses (*Hard Negative Mining* par exemple) ou appliquer une pondération spécifique dans la fonction de coût. Le modèle *RetinaNet*[62] exploite la seconde approche en proposant une nouvelle fonction de perte: *Focal Loss*.

#### 11.3.8.1 Focal Loss

*Focal Loss* est une nouvelle fonction de perte basée sur *Cross-Entropy*. Son approche propose de pondérer la valeur de l’erreur produite en fonction des probabilités issues des prédictions afin de favoriser l’expression des erreurs de prédiction. En effet, *Cross-Entropy* impose une erreur (même faible) pour toute prédiction non parfaite, i.e différente de la valeur de la données d’apprentissage (souvent 1 pour la classe correspondante). Cette imperfection du modèle n’a pas d’influence sur la performance du modèle. En effet, la prédiction reste inchangée que la probabilité soit de 1 ou de 0.7 par exemple. Cependant, pour la fonction de perte, cette différence est majeure car les vraies erreurs sont noyées par l’effet de bord produit par les bonnes prédictions "imparfaites". Afin de pondérer les vraies erreurs du modèle, *Focal Loss* propose une pondération de l’erreur selon sa probabilité prédite. Elle est définie par:

$$p_t = \begin{cases} p & \text{if } y_{\text{classe}, \text{ref}} = 1 \\ 1 - p & \text{otherwise} \end{cases}$$

$$FL(p_t) = -\alpha(1 - p_t)^\gamma \log(p_t)$$

Avec  $\alpha$  et  $\gamma$ , hyperparamètres du modèle. En supposant  $\gamma = 2$  et  $\alpha = 1$ , si  $p_t = 0.9$  alors son erreur sera 100x plus faible que l’erreur produite par *Cross-*

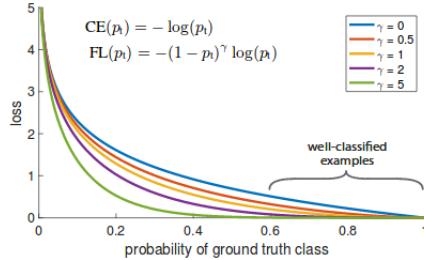


Figure 104: Erreur obtenue par Focal Loss selon  $\gamma$

*Entropy* par exemple. Cette réduction permet de s'émanciper du déséquilibre entre les distributions de classes et d'éviter l'étape de *sampling* consistant à ramener le minibatch d'apprentissage à un ratio 3:1. Le comportement de *Focal Loss* est visible sur la Figure 104. Pour  $\gamma = 0$ , *Focal Loss* est identique à *Cross-Entropy*. Expérimentalement, les créateurs du modèle ont observé que les meilleurs résultats expérimentaux sont obtenus avec  $\gamma = 2$ . Avec cette configuration, on peut observer que l'erreur pour les probabilités supérieures à 0.6 est quasi-nulle. Néanmoins, cette approche pénalise la confiance du modèle pour ne considérer que la prédiction finale. Ainsi, d'un point de vue métier, le modèle favorise les bonnes prédictions au détriment de la confiance en ses prédictions. Par conséquent, bien qu'il y ait une amélioration de prédiction empiriquement, les résultats sont moins fiables. Il peut donc être délicat d'employer ce genre de métrique lorsque l'on recherche un résultat avec une forte probabilité de certitude.

#### 11.3.8.2 Le modèle RetinaNet

L'architecture de *RetinaNet* ne présente pas de nouveautés notables et repose sur des architectures connues. Ainsi, *RetinaNet* exploite une architecture *Resnet* pour l'extraction d'attributs associée à un *Feature Pyramid Network*<sup>171</sup> (FPN) pour extraire les *feature map* de prédiction. L'approche *Anchor boxes* est utilisée pour définir les *bounding boxes*. La classification et la détermination des coordonnées de la *bounding boxes* sont définies par deux réseaux *Full Convolutional Network* indépendants, i.e ils ne partagent pas les mêmes paramètres. Une illustration de l'architecture de *RetinaNet* est visible sur la Figure 105.

L'utilisation de procédés ayant démontré leurs efficacités associée à une fonction de perte plus performante que les modèles précédents font de *RetinaNet*, l'un voire le modèle le plus performant actuellement. Seule sa vitesse de prédiction est préjudiciable, faisant de YOLO la seule alternative véritablement convaincante selon les circonstances.

<sup>171</sup>Voir Section 11.2.10 pour plus de détails

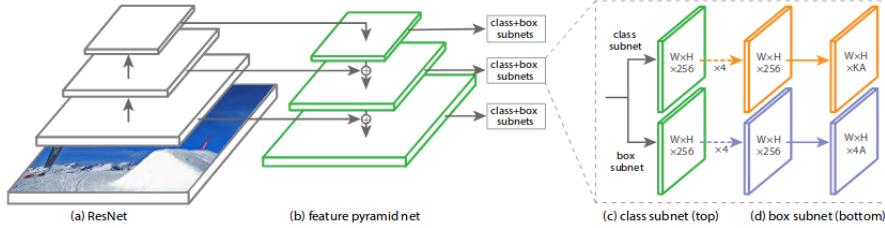


Figure 105: Architecture du réseau RetinaNet

### 11.3.9 Comparatif des modèles récents

Sur la Figure 106, nous pouvons observer un comparatif sur le jeu de données COCO selon la métrique mAP. Comment on peut le constater, les modèles *Retina* sont les plus performants que ce soit en temps de prédiction ou de précision. Seul YOLO est significativement plus rapide en contrepartie d'une précision moindre. Il est important de comprendre que ce comparatif est réalisé selon un mAP strict. Il n'est pas toujours pertinent d'avoir une prédiction avec une précision très importante. En effet, d'un point de vue métier, il n'est pas forcément utile d'être au pixel-près.

Sur la Figure 107, le même comparatif est réalisé avec un mAP placé à 0.5<sup>172</sup>. L'exigence de précision est donc plus faible mais reste suffisamment stricte pour garantir une exploitation métier satisfaisante. On observe que YOLov3 devient significativement meilleur que précédemment. Sa vitesse d'exécution est significativement plus rapide que les autres modèles et ce, pour une qualité de prédiction meilleure ou équivalente. Seule FPN FRCN est plus précis en contrepartie d'un temps de prédiction nettement plus important.

On peut donc conclure que *Retina* est le choix de référence pour la précision du modèle lorsqu'une forte exigence est requise. Au contraire, lorsque l'exigence est moindre, YOLov3 offre un compromis vitesse/précision remarquable et FPN FRCN, la meilleure précision malgré un temps de calculs élevé.

## 12 Application au traitement du langage écrit

### 12.1 Introduction

#### 12.1.1 La loi de Zipf et Mandelbrot

Au 20<sup>ième</sup> siècle, Zipf proposa une loi, nommée *Loi de Zipf* pour expliquer la fréquence d'apparition d'un mot au sein d'un texte (volumineux). Selon cette loi, si *mot<sup>n</sup>* est le *n<sup>ième</sup>* mot le plus fréquent, alors sa fréquence d'apparition est

---

<sup>172</sup>C'est une métrique standard pour les compétitions de modèles

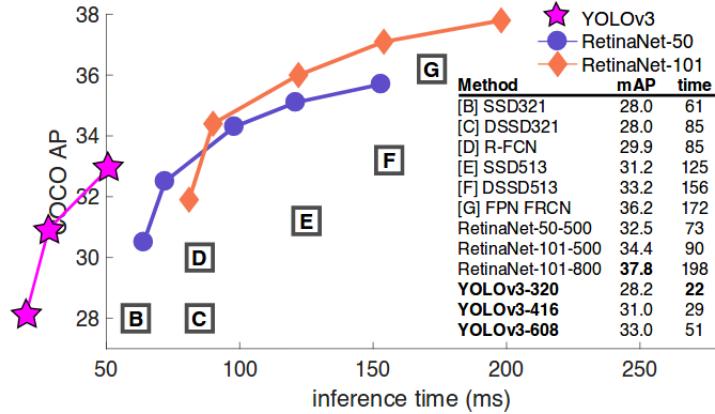


Figure 106: Comparatif des modèles d'Object Detection

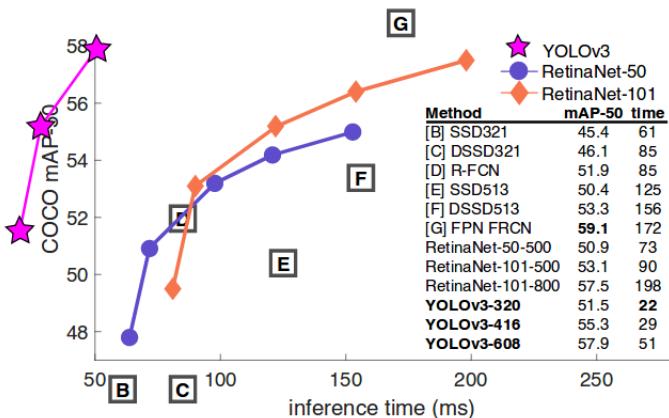


Figure 107: Comparatif des modèles d'Object Detection pour un mAP à 0.5 IoU

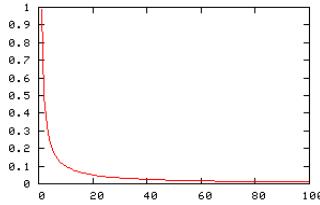


Figure 108: Loi de Zipf pour K=1

définie par  $Z(n) = \frac{K}{n}$  avec K, une constante. Supposons K=1, alors la fréquence de ce mot est de  $\frac{1}{n}$ .

Pour rappel, la Figure 108 illustre son comportement pour K=1. Nous pouvons observer que la répartition statistique des mots est très fortement déséquilibrée. Des mots sont omniprésents alors que d'autres sont quasi-absents. Ce comportement justifie des difficultés historiques du traitement du langage naturel car les modèles reposant sur des fondations statistiques, cette particularité constitue une contrainte importante. Une seconde interrogation se porte sur la nature des mots fréquents et peu fréquents. Il a été montré que les mots récurrents sont des mots de liaisons grammaticales (adverbes, pronoms, déterminants...) alors que les mots illustratifs (noms, adjetifs...) sont plus rares. Cette particularité accentue la difficulté d'analyse car l'information discriminante est *rare*.

Dans les faits, la loi de Zipf est incomplète et a été généralisée par Mandelbrot. En effet, Zipf a été vivement critiqué pour avoir exploité des mots non *lemmatisés*<sup>173</sup>. La loi de Mandelbrot est ainsi définie par  $M(n) = \frac{K}{(a+bn)^c}$  avec a,b,c et K des constantes.

Cette loi illustre des problématiques récurrentes en traitement du langage. Le comportement statistiques des données soulève des contraintes mathématiques notamment des biais pour manque de représentativité<sup>174</sup>. La faible proportion de mots explicatifs nuit aux capacités de discrimination des modèles et les rend plus sensible au bruit des données d'apprentissage. Le traitement du langage naturel (*Natural Language Processing*) est, aujourd'hui encore, un problème partiellement résolu et la recherche toujours fortement active.

### 12.1.2 Pré-traitement d'un texte

Afin d'exploiter une donnée textuelle, deux catégories de pré-traitement sont nécessaires. La première consiste à *normaliser* le texte afin de considérer une forme qui facilite l'apprentissage et limite le bruit intrinsèque lié à une distribution de données peu exploitable. La seconde permet de représenter un

<sup>173</sup>Cette notion est explicitée dans la suite de cette introduction

<sup>174</sup>Des mots peuvent ne pas être représentés dans les données d'apprentissage par exemple, ce qui est très délicat à exploiter pour les approches probabilistes

texte dans un format compréhensible par un réseau neuronal. En effet, seules les valeurs numériques sont exploitables. Il est donc nécessaire de trouver une approche permettant de représenter une valeur textuelle (donc catégorielle) en valeur numérique.

#### 12.1.2.1 Tokenization

Un réseau de neurones exploite des séquences de valeurs (série de pixels dans le cadre d'une image par exemple). Afin d'exploiter un texte, il est nécessaire de rendre son architecture sérielle, i.e transformer un texte en une suite d'entités de même nature. La Tokenization est le procédé qui permet de réaliser cette tâche. Lors de la Tokenization, une entité unitaire est appelée *Token*.

**Important:** Dans la suite de cette introduction, nous considérerons l'alphabet latin. Bien que le raisonnement soit identique pour toutes les langues, les conclusions peuvent variées selon les spécificités de chacune.

Considérons un texte quelconque. Quelles sont les entités qui le constituent ?

Source	Entité sous-jacente
Texte intégral	Phrases
Phrases	Mots
Mots	Syllabes
Syllabes	Lettres

Nous observons différentes catégories d'entités fondamentales. Dans les faits, un consensus a choisi le mots comme type de Token. Néanmoins, dans le cadre de certaines problématiques, plusieurs types de Tokenization peuvent être exploités<sup>175</sup>.

Bien qu'en apparence simple, la Tokenization présente des difficultés notables dont la solution relève essentiellement d'une sensibilité personnelle plus que d'une approche scientifiquement démontrable. Parmi ces difficultés, nous pouvons citer l'exploitation de la ponctuation et des symboles (y compris les jeux graphiques tels que les *émoticônes* par exemple), l'interprétation des mots composées et/ou à particules ou encore les ambiguïtés syntaxiques (notamment liées à l'apostrophe). De même, pour l'anglais, l'interprétation des contractions pose des difficultés. Comment traiter *aren't* ? Par *aren't*, *arent*, *aren t*, *are n't* ?

#### 12.1.2.2 Lemmatisation et racinisation

En cours

---

<sup>175</sup>Ceci sera développé dans la suite de cette introduction.

### 12.1.2.3 Stopwords

Lors d'une tâche de classification (Analyse sentimentale, catégorisation de documents par exemple), il est nécessaire d'extraire l'information discriminante d'un texte afin de pouvoir le classifier. Pour cela, il est nécessaire de mettre en valeur, les mots porteurs d'informations.

Les mots les plus fréquents dans un texte sont des entités de liaisons grammaticales. Par exemple, nous pouvons citer les déterminants, pronoms ou encore certains adverbes. Ces mots n'ont pas de pouvoir de discrimination car leur expression est neutre et/ou sont trop représentés statistiquement pour être discriminant. Par exemple, le mot le n'a aucun pouvoir de discrimination alors que l'auxiliaire être est trop représentés pour être discriminant.

Ces mots sont définis comme *Stop Words*, i.e des mots au pouvoir de discrimination nul. Ils sont donc inutiles pour la tâche de classification qui repose sur des entités au fort pouvoir explicatif. Une approche classique de pré-traitement de texte est donc de supprimer ces mots des textes à analyser pour ne conserver que des mots pertinents.

Ainsi, par exemple, supposons les tokens suivants: [Cet, article, est, une, arnaque, car, la, qualité, semble, mauvaise !]. Après suppression des *Stop Words*, nous obtenons : [article, arnaque, qualité, semble<sup>176</sup>, mauvaise].

Cette approche supprime les mots sans incidence sur la décision du réseau. La prédiction est donc plus rapide car le volume de données traitées est grandement diminué<sup>177</sup> et l'apprentissage plus performant grâce à la plus grande spécialisation des données d'apprentissage. Néanmoins, cette méthode ne conserve pas la sémantique du texte ni sa structure syntaxique.

### 12.1.2.4 Problématique et pré-traitement

Toutes les modifications précédentes (sauf Tokenization) détériorent l'intégrité sémantique et syntaxique d'un texte. Pour certaines tâches, cette détérioration est sans importance. Dans le cas d'une classification par exemple, cette perte a souvent des conséquences négligeables sur la qualité du résultat. Au contraire, pour les problématiques qui demandent de conserver une bonne structure sémantique et syntaxique, appliquer cette approche fera échouer l'apprentissage de manière critique. Il est donc important d'étudier les caractéristiques du problème à résoudre avant d'appliquer un pré-traitement sur les données.

Généralement, pour les tâches qui ne demandent pas de création de texte comme valeur de sortie, l'application de pré-traitement est envisageable. Nous pouvons citer l'analyse sentimentale, la Recherche d'informations, la désambiguïsation

---

<sup>176</sup>Il est possible de discuter sur la pertinence ou non de ce mot.

<sup>177</sup>Dans le cadre d'analyse intégrale d'oeuvres ou de documents

par exemple. Au contraire, lorsqu'un texte syntaxiquement cohérent doit être généré, utiliser un pré-traitement est dangereux. La traduction automatique, les systèmes Question-Réponse, les générateurs de texte ou encore l'Image Captioning sont des exemples classiques.

Néanmoins, les approches récentes de classification, notamment basées sur les RNN et les principes d'*Attention* imposent la conservation de la structure sémantique et syntaxique du texte. De ce fait, le pré-traitement des données tend à devenir de plus en plus "néfaste" même sur des tâches qui s'y prêtent traditionnellement.

## 12.2 Représentation vectorielle

### 12.2.1 Projection Word-Based

En cours

### 12.2.2 Projection Character-Based

En cours

## 12.3 Classification

La classification de texte est une thématique qui présente un fort intérêt d'un point de vue métier. Son objectif est, comme dans le cadre de l'image, de classer le texte dans une (ou plusieurs) catégorie qui correspond à ses spécificités. La classification des Tweets selon leurs "toxicités" (neutre, harcèlement, discrimination...) est un exemple classique d'application.

Cette thématique présente une grande diversité d'architecture exploitable. En effet, il est possible d'utiliser les réseaux convolutifs, les réseaux récurrents ou encore l'union de ces deux types de réseau. Nous verrons, dans cette partie, quelques unes des solutions standards de l'état de l'art et les spécificités théoriques observées dans le cadre de la classification de texte.

Aujourd'hui, il n'y a pas de consensus sur la meilleure famille d'architecture pour réaliser de la classification de texte. Néanmoins, les réseaux convolutifs sont plus rapides à apprendre et à réaliser des prédictions. Par contre, les réseaux récurrents ont une plus grande capacité de compréhension de la sémantique du texte, ce qui, dans le cas de texte difficile ou ambiguë, peut permettre de meilleurs résultats que les réseaux convolutifs.

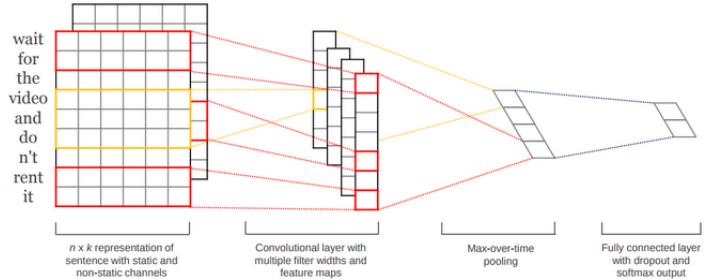


Figure 109: Architecture de CNN pour la Classification de Sentences

### 12.3.1 Classification par CNN

#### 12.3.1.1 CNN pour la Classification de Sentences

*CNN for Sentence Classification*[47] est un modèle précurseur créé en 2014. Ce modèle considère le texte comme une séquence de mots et analyse les N-Grams obtenus selon les spécificités du filtre de convolution choisi. Cette approche néglige le contexte sémantique global du texte pour se focaliser sur le comportement local des groupes de mots.

Le réseau utilisé présente une architecture simple. Les Words Embeddings correspondant aux mots du texte sont générés aléatoirement ou issus d'une bibliothèque de vecteurs pré-entraînés. Dans le cas où les vecteurs initialement sont pré-entraînés, la valeur des vecteurs est figée durant l'apprentissage. Il est possible de cumuler plusieurs Words Embeddings pour un même mot en concaténant les vecteurs. L'architecture du réseau applique une approche *Inception*. Chaque flux exploite un kernel de dimension différente afin de produire différents N-Grams (le nombre de filtre est un hyper-paramètre). Il n'y a qu'une couche de convolution par flux. Le réseau est donc très peu profond mais large. Une couche de MaxPooling est appliquée sur chaque flux pour extraire les N-Grams les plus pertinents. Ces valeurs sont ensuite envoyées à une couche Full-Connected pour réaliser la prédiction. Le modèle est visible sur la Figure 109.

Ce modèle présente une efficacité convenable sur des textes simples et de petites tailles mais est insuffisant dans le cas contraire. Néanmoins, il met en avant les capacités des réseaux convolutifs pour la classification d'un texte alors "dominé" théoriquement par les réseaux récurrents.

#### 12.3.1.2 Améliorations notables

De nombreuses améliorations de *CNN for Sentence Classification* ont été proposées. Néanmoins, l'architecture générale reste très similaire.

- **Dynamic Convolutional Neural Network (DCNN)**[44]: Ce modèle

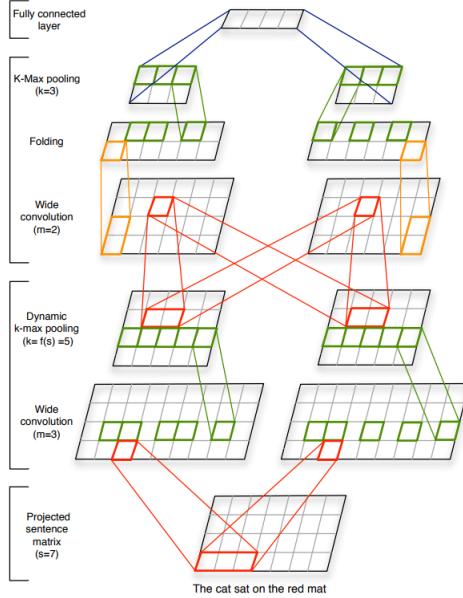


Figure 110: Architecture de Dynamic Convolutional Neural Network

exploite K-max Pooling pour capturer les relations courtes et longues distances dans le texte. Les kernels des couches de convolutions sont aussi plus larges et diminuent progressivement, de même que la dimension de sortie des couches de K-max Pooling. Son architecture est visible sur la Figure 110.

- **Multi Channel Variable size CNN (MV-CNN)**[106]: Ce modèle est très similaire à *Dynamic Convolutional Neural Network* mais exploite l'idée des Embeddings multi-sources. Ainsi, le modèle applique la même extraction d'attribut indépendamment pour chaque matrice initialisées par chacune des librairies d'Embeddings et réalise la prédiction à partir de la concaténation des résultats obtenus pour chaque flux<sup>178</sup>. L'intuition repose sur le faits que les Words Embeddings sont définies selon des méthodes différentes mais surtout des jeux de données différents. Selon la source d'apprentissage, le contexte global peut changer et de ce fait, des mots rares ou absents dans une librairie peuvent être significatifs dans une autres. Son architecture est visible sur la Figure 111. Une variante très proche, intitulé **Multi Group Norm Constraint CNN (MG(NC)-CNN)**[115], repose sur une architecture similaire mais simplifiée en plus d'un ajout de contraintes de régularisation pour compenser la simplicité du modèle.

<sup>178</sup>Si 2 librairies sont utilisées, il y aura deux flux.

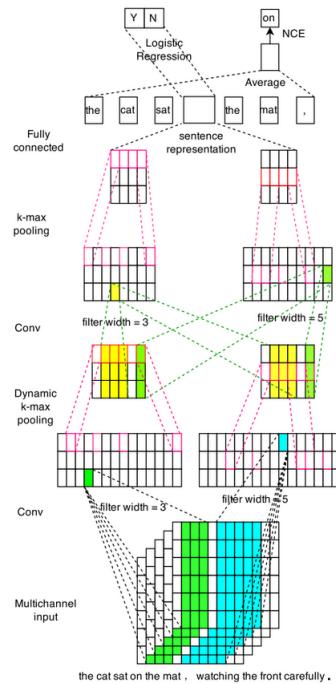


Figure 111: Architecture de Multi Channel Variable size CNN

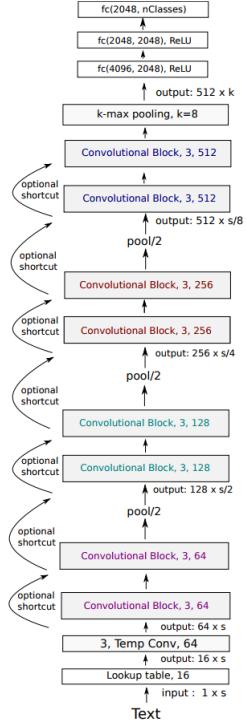


Figure 112: Architecture de Very Deep CNN

### 12.3.1.3 Very Deep CNN

Les modèles précédents se basent sur le mot comme unité de référence et exploitent un réseau peu profond mais large. Au contraire, *Very Deep CNN* (VDNN)[13] exploite la lettre comme unité de référence et repose sur une architecture très profonde mais peu large. La ponctuation est les symboles sont considérés comme des lettres et utilisés par le réseau. Des projections vectorielles spécifiques aux lettres sont utilisées pour représenter chaque lettre. Afin de considérer la spécificité de l'unité choisie, le nombre de filtres est significativement augmenté mais la dimension du kernel reste dans le même ordre de grandeur (<5). Afin de progressivement diminuer la dimension des 112.

Du fait de la profondeur du réseau et du volume de données à analyser, ce modèle est significativement plus lent à apprendre et à réaliser ses prédictions.

### 12.3.1.4 Comparatif expérimental sur les architectures CNN

Deux études expérimentales ont été réalisées afin d'étudier les différentes approches et définir laquelle semble être la plus performante.

La première, réalisée par Hoa T. Le[56], a montré que dans le contexte de la classification de texte:

- Les modèles larges et peu profonds sont meilleurs que les modèles très profonds. De plus, du fait de la complexité des modèles profonds, l'apprentissage de ce type de réseau est plus difficile et gourmand en temps et en ressources matérielles.
- L'utilisation de *Global Max Pooling* avec un modèle peu profond mais large semble aussi performant que l'utilisation du *Max Pooling* avec un modèle profond. La capacité très généralisante semble suffisamment discriminante dans le contexte de la classification.
- Les modèles exploitant le mot sont plus performants que les modèles basés sur la lettre. De plus, les modèles *character-based* imposent l'exploitation d'un réseau très profond et de ce fait, s'associe aux problématiques de l'apprentissage et de ressources.

Ainsi, les réseaux de classification de textes s'opposent aux dogmes des réseaux de classification d'images qui reposent sur l'idée que le réseau doit être très profond pour être efficace.

La seconde étude, réalisée par Ye Zhang[116], apporte une aide sur le paramétrage des hyperparamètres des modèles de classification de textes. Ainsi, plusieurs conseils ont été proposés:

- Durant la première approche, exploitez des Word Embeddings non statiques<sup>179</sup> au lieu de *One-Hot vector*. Concaténer plusieurs Word Embeddings ne semble pas améliorer significativement les performances de la classification.
- Exploitez une dimension de kernel entre 1 et 10. Réalisez éventuellement un *GridSearch* autour de la meilleure valeur de kernel obtenue.
- Variez le nombre de filtres de 100 à 600. Durant la recherche, utilisez un DropOut ( $< 0.5$ ) et une Max-Norm contrainte importante. Gardez en tête qu'il y a un compromis à faire entre nombre de filtres et temps d'apprentissage.
- Plus le nombre de *feature map* au sein du réseau augmente, plus les contraintes de régularisation doivent être fortes.
- Ne pas négliger la Cross-Validation pour la recherche d'hyperparamètres pour éviter les biais d'apprentissage. Néanmoins, cette méthode est gourmande en temps de calculs.

---

<sup>179</sup>Variables durant l'apprentissage et non figés.

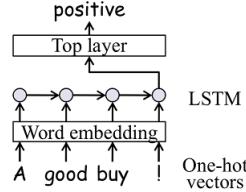


Figure 113: Architecture de LSTM Deep Sentence Embedding

### 12.3.2 Classification par RNN

Il existe des modèles de classification de textes basés sur les réseaux récurrents (spécialement avec des cellules LSTM ou GRU). Nous allons étudier plusieurs modèles de l'état de l'art reposant sur cette architecture.

#### 12.3.2.1 LSTM Deep Sentence Embedding

*LSTM Deep Sentence Embedding*[70] est un réseau simple à réaliser et présente des performances satisfaisantes. Il est composé d'un réseau LSTM-RNN et d'une couche Full-Connected qui exploite le vecteur d'états cachés de la dernière cellule LSTM. Ce vecteur résume l'information utile du texte analysé et de ce fait, représente un Embedding du document. Il utilise le mot comme entité-unité. La Figure 113 illustre l'architecture de ce modèle.

La faible dimension du vecteur caché permet de s'émanciper des couches de Pooling (ou autres méthodes de *Downsampling*). Néanmoins, l'espace de représentation est de faible dimension et de ce fait, favorise la perte d'informations. De plus, bien que le LSTM corrige grandement le problème de mémoire à long terme des réseaux récurrents, la perte d'informations dans le cas d'analyse de textes volumineux n'est pas à négliger.

#### 12.3.2.2 Discriminative RNN

*Discriminative RNN*[107] est très similaire à *LSTM Deep Sentence Embedding* mais propose une solution à la perte d'informations liée à la sélection du vecteur d'états cachés. Pour cela, au lieu d'exploiter le vecteur de la dernière cellule, l'Embedding du document sera égal à la moyenne des différents vecteurs d'états cachés du LSTM-RNN. Cette approche permet d'être moins sensible à la perte de mémoire liée à ce type de réseau. Son architecture est visible sur la Figure 114.

#### 12.3.2.3 Hierarchical Attention Networks

*Hierarchical Attention Networks*[105] pose le postulat qu'un mot et qu'une phrase ont deux pouvoirs explicatifs dissociés. Ainsi, afin de considérer ces deux sources d'informations, le réseau présente une architecture à deux niveaux

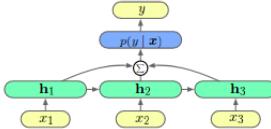


Figure 114: Architecture de Discriminative RNN

composée de deux réseaux Bi-LSTM imbriqués.

Le premier niveau analyse le texte en isolant chacune des phrases et en analysant l'importance des mots qui constituent la phrase. Le second niveau analyse le texte en fonction des phrases dont l'information est calculée à travers le premier réseau. La sortie de ce réseau sera exploitée par la couche prédictive composée de couches Full-Connected.

Supposons un texte composé de  $N$  phrases et chacune des phrases possède  $M$  mots. Le premier réseau considérera une entrée composée de  $M$  éléments successifs et réalisera  $N$  cycles prédictifs. Il produira donc  $N*M$  vecteurs d'états cachés (et  $N$  vecteurs d'états cachés finaux). Le second réseau considérera une entrée composée de  $N$  éléments successifs (chacun représentant l'information d'une phrase) et produira un cycle uniquement. Chaque entrée du second réseau correspond à la sortie obtenue à la fin d'un cycle du premier.

Cette architecture exploite aussi la notion d'*Attention* (Voir Section 10.2). Ainsi, chaque entrée du second réseau correspond à une moyenne pondérée des vecteurs de contexte du premier réseau liés à cette entrée. La pondération est dynamique et apprise durant l'apprentissage. Cette méthode est aussi appliquée à la sortie du second niveau. Ce réseau est illustré par la Figure 115.

### 12.3.3 Classification par CNN-RNN

Des architectures ont été développées en utilisant à la fois l'architecture CNN et RNN en se basant sur l'hypothèse que l'utilisation des deux architectures permettrait d'unir leurs points forts (compréhension sémantique du RNN et la détection de patterns locaux du CNN).

#### 12.3.3.1 CNN-RNN

Le modèle *CNN-RNN*[9] est composé de deux parties:

- **Extraction du vecteur de contexte:** Cette action est réalisée par un réseau convolutif (CNN). Ce réseau prend une séquence de *Word Embeddings* en entrée et produit un vecteur représenté par la sortie d'une couche

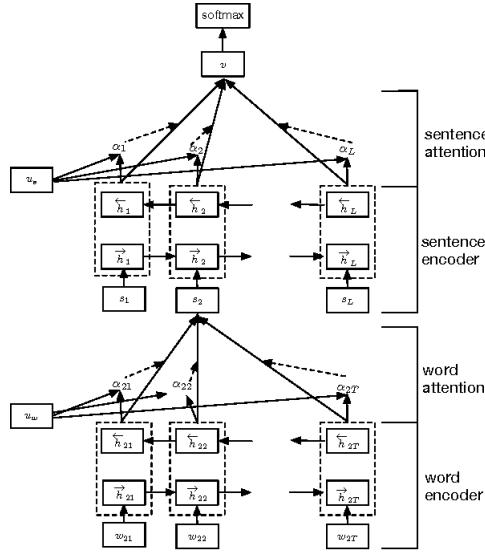


Figure 115: Architecture de Hierarchical Attention Networks

Full-Connected<sup>180</sup>. Le réseau CNN présente une architecture peu profonde mais large comparable au réseau *NN for Sentence Classification*.

- **Prédiction du label:** Cette action est réalisée par un réseau récurrent (RNN). Ce réseau exploite le vecteur de contexte pour initier son état caché initial en plus d'être additionné aux autres entrées des cellules RNN. L'addition suit la relation suivante:

$$h^{(t)} = f_{activation}(W_h x_t + U_h h_{t-1} + W_T T)$$

Avec  $T$ , vecteur de contexte et  $W_T \in R^{q*t}$  avec  $q$ , dimension de l'état caché de la cellule RNN et  $t$ , dimension du vecteur de contexte.

Une couche *softmax* est appliquée en amont de chaque sortie du RNN afin de prédire le label le plus probable selon la sortie de la cellule RNN ciblée.

La cellule RNN peut être remplacée par une cellule LSTM ou GRU afin de limiter l'impact de la perte de mémoire. Dans ce cas, le vecteur de contexte est ajouté à chacune des différentes *Gates* de la cellule. La cellule  $n+1$  utilise la prédiction faite par la cellule  $n-1$ , i.e son tag, afin d'orienter sa prédiction. Le réseau n'a connaissance du texte qu'à travers le vecteur de contexte proposé par le CNN. Il est donc important de s'assurer que

---

<sup>180</sup>La fonction d'activation de la couche est linéaire, i.e aucune fonction d'activation particulière n'est exploitée.

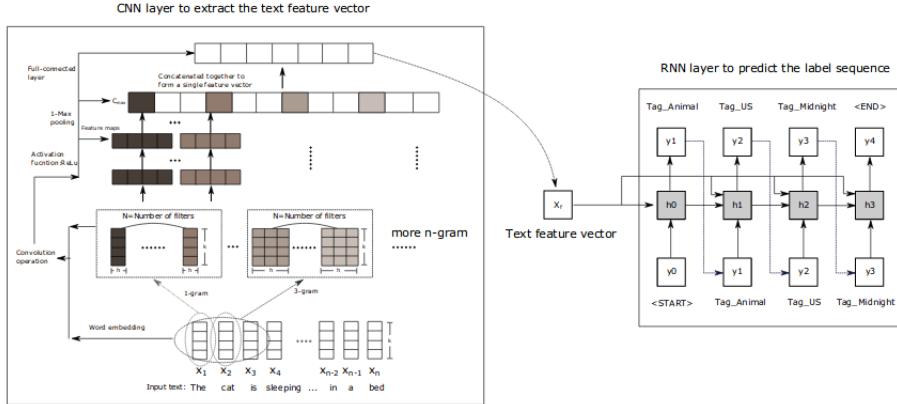


Figure 116: Architecture de CNN-RNN

la dimension de ce vecteur est suffisante pour représenter la séquence à prédire<sup>181</sup>. La Figure 116 illustre cette architecture.

### 12.3.3.2 C-LSTM

Le modèle *C-LSTM*[117] est composé de deux parties:

- **Extraction des N-Grams:** Le réseau convolutif applique n filtres (kernel identique) sur la séquence à prédire. Les n *feature map* obtenus sont concaténés selon l'axe des colonnes. Le réseau présente deux particularités:
  - Aucune méthode de Downsampling est utilisée (stride, Pooling...) afin de conserver la cohérence sémantique de la séquence<sup>182</sup>. Une alternative proposée par [99] reprend l'architecture du C-LSTM mais applique du Pooling afin de réduire la dimension des *feature map* en sortie du réseau convolutif et ainsi, augmenter la vitesse de prédiction et d'apprentissage du réseau (voir Figure 118).
  - Un seul kernel est exploité afin de permettre une bonne concaténation et la cohérence des données concaténées. Néanmoins, exploiter différents kernels est possible en s'assurant du respect des dimensions des *feature map* notamment via l'usage de *padding*. **Cette spécificité est incertaine et relève de mon interprétation du papier de recherche. Veuillez vous référer au papier associé afin de confirmer cette affirmation.**

<sup>181</sup>Plus le texte est complexe et volumineux, plus la dimension du texte doit être grande. Néanmoins, l'utilisation de l'*Attention* peut aider à éviter cette proportionnalité dimensionnelle.

<sup>182</sup>Extraire des sous-séquences détruit l'intégrité de la continuité sémantique

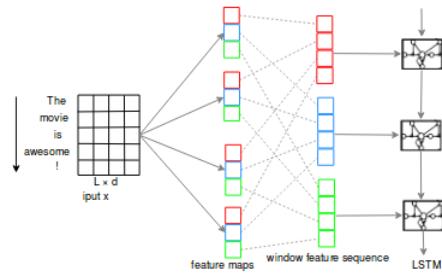


Figure 117: Architecture de C-LSTM

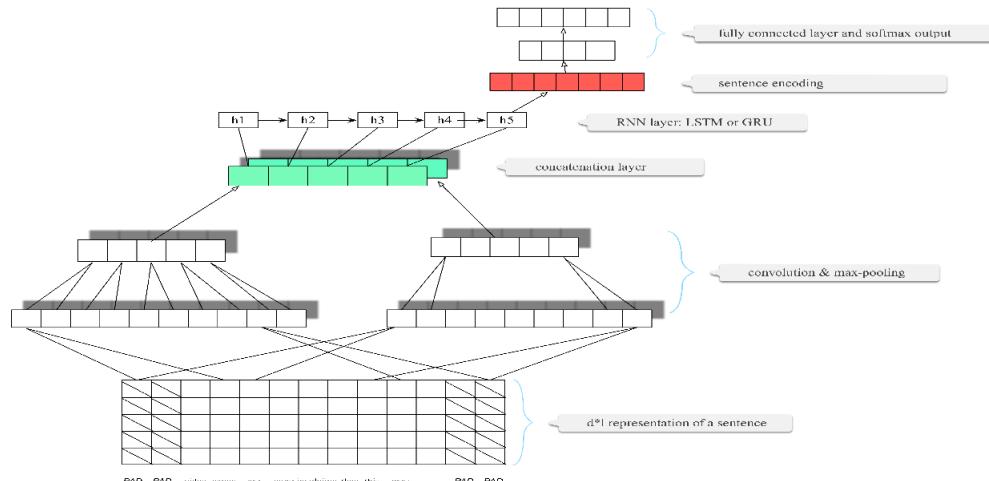


Figure 118: Architecture de C-LSTM avec Pooling

- **Extraction d'un vecteur de contexte:** Le réseau RNN (LSTM) est utilisé afin d'interpréter le comportement sémantique du texte à partir des N-Grams obtenus par le réseau convolutif. Ainsi, l'entrée séquentielle  $i$  du RNN correspond aux N-Grams concaténés à la position  $i$  des différentes *feature map*<sup>183</sup>.

Une couche *softmax* prend en entrée le **dernier état caché** du LSTM afin de réaliser sa prédiction. Il n'y a donc qu'une prédiction réalisée (à partir du dernier état caché) en fin de réseau alors qu'avec le modèle *CNN-RNN*, la prédiction se fait *en continu* à partir des différents états cachés du RNN.

---

<sup>183</sup>Dans les faits, les *feature map* sont représentées par une matrice unique de dimension  $i*j$  avec  $i$  nombre de N-grams et  $j$ , nombre de filtres. Les entrées du RNN correspondent donc aux lignes de la matrice.

### 12.3.3.3 AC-BLSTM

Le modèle *symmetric Convolutional Bidirectional LSTM Networks* (AC-BLSTM) [59] propose une modification des couches convolutives nommée *Asymmetric Convolution* afin d'extraire l'information de la séquence à prédire.

Supposons un séquence S de dimension L donc le j-ième mot  $x_j$  est représenté par un *Word Embeddings* tel que  $x_j \in R^d$ . Nous avons donc:

$$S = [x_1, \dots, x_L]$$

$$x_i = [x_{i,1}, \dots, x_{i,d}]$$

Supposons k dimension du kernel du filtre d'une couche de convolution. Une convolution standard (sur une dimension) réalise donc une opération de dimension  $k*d$ , i.e analyse k mots successifs représentés par un vecteur de dimension d.

Une convolution asymétrique réalise la convolution en deux étapes convolutives:

- La première étape est représentée par un kernel de dimension  $1*d$ . Ainsi, l'action de cette opération correspond à une projection vectorielle définie par  $R^d \rightarrow R^1$ . En d'autres mots, le *Word Embeddings* est représenté par une valeur uniquement à la fin de cette étape.
- La seconde étape est représentée par un kernel de dimension  $k*1$  où k correspond à la taille du filtre choisi initialement. Le fonctionnement de cette étape est comparable à une convolution standard sur des données à 1 dimension.

Expérimentalement, la convolution asymétrique a présenté des résultats qualitatifs. De même, on peut observer une ressemblance avec l'approche *Depthwise-Pointwise* très utilisée pour l'analyse d'image et les réseaux faibles consommation.

La concaténation des *feature map* et l'exploitation du réseau récurrent de AC-BLSTM est comparable à celle de C-LSTM. Deux différences importantes sont à noter pour cette dernière.

- Le réseau AC-BLSTM exploite un réseau *Bi-directionnel* afin d'améliorer les capacités prédictives du réseau LSTM.
- La prédiction du label est réalisée à partir d'une couche softmax qui prend en entrée l'intégralité des états internes du BLSTM concaténés. Ainsi, supposons un réseau BLSTM composé de m cellules de dimension n. L'entrée de la couche softmax sera alors de dimension  $m * (2 * n)$ .

Une illustration de l'architecture du réseau est visible sur la Figure 119. Néanmoins, une particularité importante est à remarquer. Contrairement au réseau C-LSTM, AC-BLSTM accepte les kernels de tailles variables. Sans l'exploitation

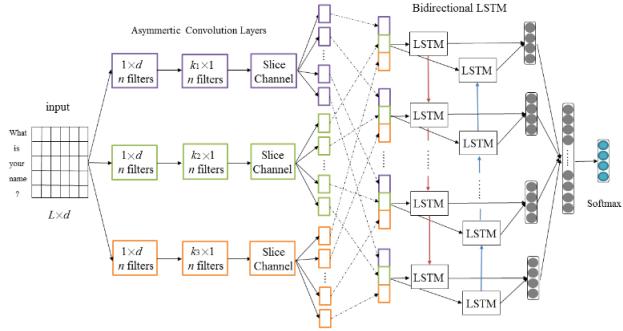


Figure 119: Architecture de AC-BLSTM

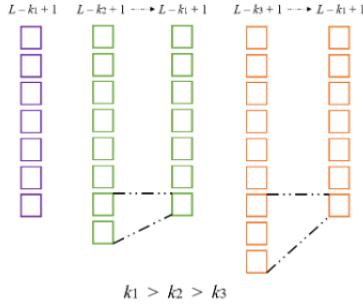


Figure 120: Harmonisation des dimensions des feature map selon l'approche de AC-BLSTM

de padding, les *feature map* sont de dimensions variables et ne peuvent être concaténées. Pour résoudre cette problématique, une méthode a été proposée.

Supposons un filtre de dimension  $k_i$  appliqué sur une séquence de taille  $L$ , une convolution sans padding produira une *feature map* de dimension  $D_i = L - k_i + 1$ . Supposons maintenant un filtre de dimension  $k_j$  tel que  $k_j > k_i$  alors, la dimension de la *feature map* sera de  $D_j = L - k_j + 1$  avec  $D_i > D_j$ . L'approche standard consiste à **supprimer** les dimensions supérieures à  $\min(\text{Card}(D_{k, k \in \text{filters}})) = \hat{D} = D_j$ . Cependant, elle est très destructrice.

Supposons  $c_i$ , *feature map* du filtre i. La méthode proposée consiste à extraire pour chaque *feature map* c, les valeurs  $c_i^t$  avec  $t \geq L - \hat{D} + 1$  et d'appliquer une couche Full-Connected afin de produire un nouvel attribut qui remplacera les valeurs placées en entrée de la couche FC. Les dimensions de différentes *feature map* sont ainsi harmonisées. Un exemple illustratif de l'action réalisée est visible sur la Figure 120.

## 13 Apprentissage par Renforcement et Deep Learning

**Important:** Ceci ne constitue pas un cours expliquant l'apprentissage par Renforcement. Bien que des notions soient introduites, de nombreux points importants seront ignorés car s'éloignant de la thématique du Deep Learning, de même que la rigueur mathématique<sup>184</sup> qui sera limitée pour faciliter la compréhension générale des idées explicitées.

Les notations peuvent sembler lourdes et les détails mathématiques difficiles à assimiler. Néanmoins, les mathématiques abordées relèvent essentiellement du jeu d'écriture et d'*astuces* à connaître.

### 13.1 Approche théorique

#### 13.1.1 Généralités

L'*Apprentissage par Renforcement* est l'une des grandes familles d'apprentissage automatique au même titre que l'apprentissage supervisé et non supervisé par exemple. Cette approche se veut très proche de la méthodologie de l'apprentissage de l'homme basée sur l'expérience et l'analyse des conséquences suite à une action réalisée<sup>185</sup>. Bien que peu répandu en dehors de la Recherche, l'*apprentissage par renforcement* commence à se révéler grâce à sa capacité d'adaptation, sa faible dépendance à des données d'apprentissage pré-fournies et à ses performances. Il est très présent dans le cadre des systèmes autonomes (notamment les véhicules) et de la résolution de jeux (jeux de plateau tels que le Go ou les échecs, jeux vidéo).

Les modèles d'*apprentissage par renforcement* reposent sur les interactions avec l'environnement et la réponse de cet environnement vis-à-vis de l'action choisie. Ainsi, la problématique peut être formalisée par un agent A, qui à l'instant  $t$ , au sein d'un environnement, doit choisir une action à réaliser, notée  $a_t$  selon l'état  $s_t$  dans lequel il se trouve. Après son action, l'environnement fournit une récompense  $r_t$  correspondant à la conséquence de cette action sur l'environnement. La récompense peut être positive ou négative selon si la réponse de l'environnement est bénéfique ou non. L'agent se trouve alors dans l'état  $s_{t+1}$  et choisit une nouvelle action à réaliser (ou s'arrête selon le cas). La Figure 121 résume cette problématique. L'objectif de cet apprentissage est donc de définir un modèle capable de maximiser les gains de récompenses.

---

<sup>184</sup>Nous ne traiterons pas les preuves de convergence des modèles par exemple alors que ça représente un aspect très important des algorithmes d'apprentissage !

<sup>185</sup>Un enfant apprend de ses chutes jusqu'à réussir à marcher

### 13.1.1.1 L'environnement

L'environnement représente le milieu où évalue l'agent. Il peut présenter différentes spécificités:

- **Déterministe/Stochastique:** Lors d'une action  $a_t$  à partir d'un état  $s_t$ , l'agent arrive dans l'état  $s_{t+1}$ . L'environnement est défini par un espace d'états  $S$  fini ou non et d'un espace d'actions possibles  $A$ . Il est *déterministe* si:

$$\forall a_t, s_t \in (A_{s_t}, S), \exists s_{t+1} \in S, P(s_{t+1} | a_t, s_t) = 1$$

Si non, il est considéré comme *stochastique*. Un environnement *déterministe* est défini par un environnement qui, pour tout stimulus de même nature, réagit de la même manière à partir d'un état donné. Par exemple, si on suppose le jeu Mario, lorsqu'on exécute l'action "sauter", Mario va sauter à chaque fois (si l'état à partir duquel l'action est faite le permet). Au contraire, un environnement est stochastique si:

$$\forall a_t, s_t \in (A_{s_t}, S), \exists s_{t+1} \in S, P(s_{t+1} | a_t, s_t) < 1$$

Cette particularité associe un facteur probabiliste à l'environnement. Ainsi, suite à une action et un état donnés, l'état suivant est déterminé selon une distribution probabiliste. Supposons une voiture, l'action d'accélérer peut réaliser l'accélération ou, dans d'autres cas, ne pas fonctionner en cas d'une panne par exemple.

- **Observable ou Partiellement Observable:** L'environnement est défini comme observable lorsque l'ensemble des états qui le forme sont pleinement définis et connaissables. Au contraire, l'environnement est défini comme partiellement observable lorsque les états ne sont pas ou partiellement connaissables et que l'acteur n'a accès qu'à des observations. Ainsi, un environnement partiellement observable est défini par une fonction de transitions entre les observations (qui définit la probabilité d'observer un phénomène à partir d'une action dans un état donné) et d'une fonction de perception liant observations et état (qui définit la probabilité d'avoir une observation dans un état). Ainsi, supposons,  $S$ , ensemble des états,  $A$ , ensemble des actions et  $Z$ , ensemble des observations, la fonction de transition entre observations est définie par:

$$(s_t, a_t, z_{t+1}) \in (S, A_{s_t}, Z_{s_t}), T(s_t, a_t, z_{t+1}) = P(z_{t+1} | s_t, a_t)$$

La fonction de perception, noté  $\varphi$ , est définie par:

$$(s_t, z_t) \in (S, Z_{s_t}), \varphi(s, z) = P(z_t | s_t)$$

- **Discret/Continu:** Dans la configuration d'un environnement discret, le temps évolue de manière séquentielle. Ainsi, la fonction temps est définie par un pas discret soit de  $t$  à  $t+1$ . Dans le cas continu, le temps évolue

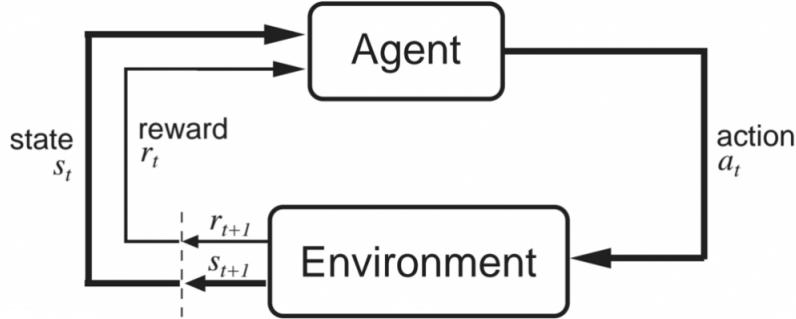


Figure 121: Graphique séquentiel de l'approche par renforcement

de manière continue soit de  $t$  à  $t+\Delta t$  avec  $\Delta t$  infiniment petit. De ce fait, un état n'évolue pas de manière discrète mais varie continuellement selon une quantité  $\frac{ds}{dt}$  qui peut être comparée à une vitesse de variation d'état.

- **Connu/Inconnu:** L'environnement peut être inconnu de l'acteur. Ainsi, il peut ne pas connaître les récompenses obtenues, les fonctions de transitions voire même les états eux-mêmes. Au contraire, il peut aussi évoluer dans un environnement clairement établi et défini en amont. Par exemple, calculer l'itinéraire d'un trajet peut être associé à un environnement connu. Apprendre à un automate à marcher "par lui-même" est un problème à environnement inconnu car l'automate ne connaît pas l'impact de ses actions ni-même l'étendu des possibilités des différentes postures qu'il peut visiter durant son apprentissage.
- **Stationnaire/Non Stationnaire:** L'environnement peut varier au cours du temps. De ce fait, les récompenses et les fonctions de transitions peuvent ne pas être constantes. Un environnement variable implique un apprentissage permanente et évolutif de l'acteur. Si l'évolution de l'environnement suit un processus aléatoire, l'apprentissage ne peut être réalisé pleinement si ce n'est l'adaptation de l'acteur à son nouveau milieu. Si l'évolution est aléatoire et permanente, l'acteur ne peut apprendre. Comment peut-on jouer à un jeu dont les règles évoluent tout le temps de manière aléatoire ?
- **Fini/Infini:** L'environnement peut posséder un nombre fini d'états comme pour un jeu de plateau par exemple. Au contraire, il peut aussi avoir un nombre infini d'états. Un problème avec un nombre infini d'états est souvent associé à un problème à temps continu qui, du fait du nombre infiniment grand d'états, est jugé à environnement infini.

### 13.1.1.2 Formalisation du problème

Supposons un environnement stationnaire entièrement observable où le temps est discret. Le problème peut être formalisé par un **problème de décision de Markov** (PDM). Ce problème est défini par le quadruplet  $(S, A, P, R)$  défini tel que:

- **S**: ensemble d'états **finis**
- **A**: ensemble d'actions fini tel que  $A(s)$  est associé à l'ensemble des actions réalisables à l'état  $s$ <sup>186</sup>
- **T**: Fonction de transition qui explicite la dynamique de transition entre états pour une action donnée:

$$(s, a, s') \in (S, A, S) \rightarrow P(s, a, s') = Pr(s_{t+1} = s' \mid a_t = a, s_t = s)$$

- **R**: Fonction de récompenses sur  $\mathcal{R}$  qui explicite l'espérance (valeur moyenne) des récompenses renvoyées par l'environnement associées à chaque changement d'état. Elle est définie par:

$$(s, a, s') \in (S, A, S) \rightarrow R(s, a, s') = E(r_t \mid a_t = a, s_t = s, s_{t+1} = s')$$

Le problème est à *horizon fini* si il existe un (des) état(s) dits *final(aux)*, i.e provoque la fin de l'expérience. Par exemple, la fin d'une partie d'un jeu. Cette particularité permet ainsi la réalisation de plusieurs expériences distincts nommées *itérations* car le problème est à durée finie. Au contraire, le problème peut être à *horizon infini*. Il ne possède donc pas d'état(s) final(aux) et de ce fait, il n'y a qu'une expérience (à durée non finie). Par exemple, la survie d'un robot en mission<sup>187</sup>.

**Important:** On considère que l'état actuel ne dépend **QUE** de l'état précédent et de l'action qui y a été réalisée. Ainsi, la fonction de transition ne considère que l'action et l'état précédents pour choisir l'état actuel. Ceci est une hypothèse très forte qui suppose qu'une situation à l'instant  $t$  dépend exclusivement de la situation à l'instant  $t-1$ . Ceci est généralement faux dans les faits ! Cette hypothèse est appelée l'**hypothèse de Markov**.

Lorsqu'on ne considère que la situation précédente pour choisir l'état actuel, on décrit un problème de Markov **d'ordre 1**. Si on considère la situation  $t-1$  et  $t-2$ , nous obtenons un problème de Markov **d'ordre 2**. Par convention un PDM est défini si l'hypothèse de Markov d'ordre 1 est définie. Néanmoins, tout problème de Markov d'ordre  $n$  peut être ramené à un problème de Markov d'ordre 1 en augmentant le nombre d'états.

---

<sup>186</sup>Il se peut qu'un sous-ensemble uniquement de  $A$  soit réalisable dans un état  $s$  de  $S$

<sup>187</sup>On peut, selon le point de vue, considérer la panne d'énergie ou l'usure du robot comme un état final si il implique son arrêt total

### 13.1.1.3 Définitions fondamentales

L'objectif de l'acteur est de maximiser<sup>188</sup> les récompenses obtenues durant les interactions avec l'environnement. Notons  $r_t$ , la récompense obtenue par l'acteur à l'instant  $t$ . Le *Retour* définit la somme des récompenses obtenues par l'acteur à partir de l'instant  $t$ . Nous avons donc:

$$R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}$$

Avec  $\gamma \in [0, 1]$ . Ce facteur est appelé *facteur de dépréciation*. Il permet de pondérer l'importance des récompenses obtenues selon la proximité temporelle. Si  $\gamma = 1$ , toutes les récompenses sont considérées avec la même intensité. Le système a donc un raisonnement à très long terme. Au contraire, si  $\gamma \rightarrow 0$ , le système ne considère que les récompenses proches et n'a pas de vision à long terme. Ce coefficient est très important en cas d'*horizon infini* car, étant donné l'absence d'état final, l'expérience n'a "pas de fin". Il est donc nécessaire de borner la somme par l'ajout de ce critère qui délimite la vision à long terme.

La *politique* représente la capacité de décision de l'agent. Elle définit l'action qui est choisie selon les critères qu'elle considère. Ainsi, nous la représentons telle que:

$$(s, a) \rightarrow (S, A), \pi(s, a) = P(a_t = a | s_t = s)$$

La politique est donc caractérisée par la probabilité de réaliser une action spécifique dans un état  $s$  de  $S$ . La définition précédente définit une politique *stochastique*. Selon les spécificités voulues pour le modèle, il est possible de désirer une politique *déterministe* où chaque état est associé à une action unique (probabilité de 1<sup>189</sup>). De ce fait, la politique suit la définition:

$$s \rightarrow S, \pi(s) = a$$

Un état possède une valeur qui représente sa qualité<sup>190</sup>. Cette valeur est liée à la politique et variable dès lors que la politique évolue. Elle est définie par l'espérance des récompenses cumulatives à partir de l'état ciblé peu importe les actions réalisées par la suite. Ainsi, pour un état  $s \in S$  et une politique  $\pi$ , nous avons:

$$V^\pi(s) = E(R_t | s_t = s)$$

Alors que  $V$  évalue la qualité d'un état peu importe les actions réalisées,  $Q$  définit une valeur d'un couple *état-action* pour une politique  $\pi$  donnée.  $Q$  évalue donc la qualité d'un état en considérant l'action qui a mené à cet état, au contraire de  $V$  qui l'ignore. Nous avons donc:

$$Q^\pi(s, a) = E(R_t | s_t = s, a_t = a)$$

---

<sup>188</sup>Ou de minimiser selon la formulation de la problématique

<sup>189</sup>la politique choisit une action unique mais l'environnement peut être stochastique et faire en sorte que l'agent ne finisse pas nécessairement dans l'état choisi par la politique. Il faut bien différencier l'aspect stochastique de la politique et de l'environnement

<sup>190</sup>Par convention, plus la valeur est élevée, plus l'état est qualitatif donc présentant un intérêt à l'atteindre par l'agent

### 13.1.1.4 L'équation fondatrice: l'équation de Bellman

Nous avons vu que:

$$R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}$$

$$V^\pi(s) = E(R_t | s_t = s)$$

Par un jeu d'écriture, nous pouvons donc redéfinir  $V^\pi$ .

$$V^\pi(s) = E(R_t | s_t = s) \quad (10)$$

$$= E\left(\sum_{k=0}^{\infty} \gamma^k r_{t+k} | s_t = s\right) \quad (11)$$

$$= E(r_t + \sum_{k=1}^{\infty} \gamma^k r_{t+k} | s_t = s) \quad (12)$$

$$= E(r_t + \gamma \sum_{k=1}^{\infty} \gamma^{k-1} r_{t+k} | s_t = s) \quad (13)$$

$$= E(r_t) + \gamma E\left(\sum_{k=1}^{\infty} \gamma^{k-1} r_{t+k} | s_t = s\right) \quad (14)$$

$$(15)$$

Or, nous pouvons constater que:

$$V^\pi(s_{t+1}) = E\left(\sum_{k=1}^{\infty} \gamma^{k-1} r_{t+k} | s_t = s\right)$$

$E(r_t)$  présente une difficulté car l'espérance doit considérer toutes les situations possibles. Il est donc nécessaire de tenir de compte de l'aspect *stochastique* de la fonction de transition et de la politique. Dans une configuration *déterministe*, l'équation en sera qu'un cas particulier. Supposons  $R(s_t, a_t, s_{t+1})^{191}$ , espérance de la récompense obtenue en passant de l'état  $s_t$  à  $s_{t+1}$  par l'action  $a_t$ . Dans un premier temps, nous considérerons l'environnement et la politique comme *déterministes*. Ainsi, chaque action garantit d'atteindre l'état-cible et la politique choisit toujours la même action pour un état donné. Nous obtenons donc:

$$E(r_t) = R(s_t, a_t, s_{t+1})$$

Supposons l'environnement comme stochastique. L'état-cible n'est donc plus garanti d'être atteint mais relève d'une considération probabiliste. De ce fait, la valeur de récompense associée à l'état actuel repose sur la somme des différentes valeur de récompenses obtenues en atteignant les différents états de l'environnement possibles. Nous avons donc:

$$E(r_t) = \sum_{s_{t+1} \in S} P(s_t, a_t, s_{t+1}) R(s_t, a_t, s_{t+1})$$

---

<sup>191</sup>Valeur issue de la fonction récompense définie par le PDM

Avec la même approche, supposons une politique et un environnement stochastiques. Nous obtenons alors:

$$E(r_t) = \sum_{a_t \in A(s)} \pi(s_t, a_t) \sum_{s_{t+1} \in S} P(s_t, a_t, s_{t+1}) R(s_t, a_t, s_{t+1})$$

Par un calcul analogue, nous pouvons donc redéfinir  $V^\pi(x)$ . Cette équation, nommée **équation de Bellman**, est au coeur de la théorie de l'apprentissage par renforcement et constitue le socle théorique fondamental de cette famille d'algorithmes. Elle est définie par:

$$V^\pi(s_t) = \sum_{a_t \in A(s)} \pi(s_t, a_t) \sum_{s_{t+1} \in S} P(s_t, a_t, s_{t+1})(R(s_t, a_t, s_{t+1}) + \gamma V^\pi(s_{t+1}))$$

L'objectif du modèle est d'obtenir la politique optimale qui maximise les valeurs d'états. Nous noterons la fonction valeur de la politique optimale,  $V^*(s)$ . Elle constitue l'**équation d'optimalité de Bellman**.

$$V^*(s) = \max_\pi V^\pi(s) = \max_{a \in A(s)} \sum_{s_{t+1} \in S} P(s_t, a_t, s_{t+1}) R(s_t, a_t, s_{t+1})$$

### 13.1.1.5 Model-Free et Model-Based

En connaissant l'ensemble des éléments du MDP, il est "aisé" de définir une solution avant d'interagir avec l'environnement. On peut donc considérer ce problème comme un problème de *planification* usant d'algorithmes de *programmation dynamique*<sup>192</sup>. Dans les faits, l'environnement est rarement parfaitement connu, i.e la fonction de récompenses et de transitions sont inconnues. L'agent doit donc être capable d'évoluer sans connaissance préalables de l'environnement où il se trouve. Pour cela, il existe deux approches d'apprentissage: **Model-Free** et **Model-Based**.

**Model-Based** repose sur l'idée que l'agent doit apprendre un modèle qui approxime les caractéristiques de l'environnement pour définir une politique optimale. A partir de son approximation, il est capable de définir le comportement optimal à suivre pour réaliser son objectif. Ainsi, si l'agent est dans l'état  $s_t$ , qu'il réalise l'action  $a_t$  et qu'il atteint l'état  $s_{t+1}$  avec une récompense  $r_{t+1}$ , alors le modèle cherchera à améliorer son estimation de  $P(s_{t+1}|s_t, a_t)$  et de  $R(s_t, a_t)$  à travers les valeurs de  $V$ . Le modèle apprendra donc les conséquences de la réalisation d'une action dans un état donné. A partir des conséquences connues, le modèle définira la politique la plus performante.

**Model-Free** s'émancipe de l'approximation du modèle pour définir une politique. Il n'y a donc pas d'approximation de l'environnement, i.e de la fonction de transitions et de récompenses. Le modèle cherche donc à apprendre directement la politique à suivre, i.e quand choisir telle action. L'algorithme *Q-Learning*[101] est l'algorithme de référence pour ce type d'approche.

---

<sup>192</sup>Nous ne détaillerons pas ces algorithmes dans ce cours

### 13.1.1.6 Différence temporelle

Une des principales faiblesses des premières approches par apprentissage par renforcement (sans connaissances explicites du PDM) était la nécessité de devoir finir une itération pour mettre à jour le modèle (par exemple, la méthode de Monte-Carlo). Ce type d'approche demande du temps et surtout, est inefficace dans le cadre d'un environnement à *horizon infini*<sup>193</sup><sup>194</sup>. Afin de palier à cette faiblesse, la notion de *différence temporelle* a été introduite par Sutton dans son article de recherche [90].

Nous avons vu que pour une politique  $\pi$  donnée, nous avons:

$$V^\pi(s_t) = r_t + \gamma V^\pi(s_{t+1})$$

Par un simple jeu d'écriture, nous obtenons:

$$\underbrace{r_t + \gamma V^\pi(s_{t+1})}_{\text{Temporal Difference}} - V^\pi(s_t) = 0$$

Durant l'apprentissage, cette égalité n'est pas vérifiée. Si la valeur actuelle de  $V^\pi(s_t)$  est trop élevée, la valeur TD sera négative et positive dans le cas contraire. Cette valeur permet donc d'orienter l'évolution de  $V^\pi(s_t)$ . La correction se présente sous la forme:

$$V_{t+1}^\pi(s_t) \rightarrow V_t^\pi(s_t) + \alpha(r_t + \gamma V_t^\pi(s_{t+1}) - V_t^\pi(s_t))$$

Avec  $\alpha \in [0, 1]$ .  $\alpha$  est associé à un taux d'apprentissage. Il est pertinent pour se protéger de la condition de stochasticité de l'environnement. En effet, si il est déterministe, nous sommes "sûr" de la modification à réaliser sur V. De ce fait, nous pouvons exploiter un taux égal à 1. Au contraire, si l'environnement est stochastique, les modifications sont incertaines car dépendantes de la distribution des états. Initialement, nous considérerons une valeur proche de 1 car la distribution est inconnue. Cela permet des mises à jour grossières et importantes (souvent très oscillantes) le temps d'estimer les distributions convenablement. Avec le temps, le taux tendra vers 0 car les distributions seront bien estimées via les valeurs de V et pour permettre une bonne convergence du modèle. Pour une approche par défaut, il est commun de définir  $\alpha$  comme une fonction dépendante de l'état considérée tel que:

$$\alpha(s) = \frac{1}{1 + \text{nombre de visite de } s}$$

*Temporal difference* est utilisée au sein d'un algorithme nommé TD(0) qui exploite sa capacité de correction pour évaluer les valeurs d'états pour une politique donnée (model-based). Il est défini sur la Figure 122.

<sup>193</sup>Un environnement sans état final donc sans "fin" d'expérience explicite

<sup>194</sup>Ceci n'est pas complètement vrai. Il existe des "astuces" pour ignorer ce type de contraintes mais nous ne les considérerons pas

```

Input: the policy  $\pi$  to be evaluated
Initialize  $V(s)$  arbitrarily (e.g.,  $V(s) = 0, \forall s \in \mathcal{S}^+$ )
Repeat (for each episode):
    Initialize  $S$ 
    Repeat (for each step of episode):
         $A \leftarrow$  action given by  $\pi$  for  $S$ 
        Take action  $A$ ; observe reward,  $R$ , and next state,  $S'$ 
         $V(S) \leftarrow V(S) + \alpha[R + \gamma V(S') - V(S)]$ 
         $S \leftarrow S'$ 
    until  $S$  is terminal

```

Figure 122: Algorithme du TD(0)

### 13.1.1.7 Un standard: l'algorithme Q-Learning

TD(0) évalue une politique mais ne permet pas de la mettre à jour. Il est donc nécessaire de proposer une autre méthode capable de proposer une politique améliorée. Il est possible d'exploiter des méthodes "standards"<sup>195</sup> pour exploiter le résultat de TD(0) ou d'utiliser un autre algorithme qui permet d'apprendre une politique de manière autonome comme l'algorithme *Q-Learning*.

Cette algorithme repose sur la même approche que TD(0) sauf qu'il exploite les valeurs Q et non V. De ce fait, il considère les couples(état,action) et non juste les états. Son comportement est donc spécifique aux actions et non cumulatif comme le fait TD(0). L'algorithme est détaillé sur la Figure 123.

Deux spécificités importantes sont à approfondir. La première est l'utilisation de  $\max(Q(s_{t+1}, a_{t+1}))$ . Par conséquent, le modèle exploitera toujours la valeur de Q associée à la meilleure action à faire dans l'état t+1, indépendamment de l'action qui sera véritablement réalisée (la meilleure action n'est pas forcément celle qui est réalisée !) dans cet état. On caractérise les algorithmes usant de ce type d'approche comme des algorithmes **off-policy**. Un autre algorithme, nommé *SARSA*[89], exploite la même approche que *Q-Learning* à la différence que *SARSA* n'exploite pas la valeur max mais la valeur effectivement choisie par le modèle. C'est ainsi un algorithme dit **on-policy**.

La différence effective entre les deux algorithmes est "l'audace" de l'agent. En effet, Q-Learning n'exploite que les meilleures valeurs indépendamment des actions réelles effectuées. De ce fait, Q-Learning ne tient pas compte de situations potentiellement néfastes pour l'agent car la mise à jour des valeurs de Q ne considère pas l'action réellement effectuée. Au contraire, SARSA considère l'ensemble des situations que l'agent visite et se met à jour en adéquation avec les situations "dangereuses" observées. Ce modèle sera donc beaucoup plus prévenant et moins audacieux qu'un agent sous Q-Learning. Pour imager, supposons qu'un individu souhaite aller d'un point A à un point B où ces deux points sont au bord d'une falaise. L'agent sous Q-Learning se déplacera le long du bord jusqu'à arriver à destination. En effet, c'est le chemin le "plus court" et

---

<sup>195</sup>non traitées dans ce cours

efficace. Cependant, il ne tient pas compte du risque (peu probable mais réel) de chute possible. Au contraire, l'agent sous SARSA s'éloignera légèrement du bord, augmentant la distance parcourant mais évitant le piège de la chute. Q-Learning favorise l'efficacité par rapport au risque alors que SARSA réalise un compromis entre les deux. Il n'y a pas de meilleure approche. Tout dépend de la problématique à traiter et du contexte de l'agent.

La seconde spécificité à comprendre est la politique choisie par Q-Learning (ou pour SARSA). Avant de présenter ces politiques, il est nécessaire d'introduire la notion de **compromis exploration/exploitation**.

Pour obtenir une bonne compréhension de l'environnement, il est nécessaire de l'explorer et de le visiter. Cette étape est appelée **exploration**. Lorsque l'environnement est assimilé et connu, il n'est plus pertinent de visiter mais au contraire, il est nécessaire d'exploiter la connaissance obtenue. Cette étape est nommée **exploitation**. Un compromis entre ces deux comportements est au cœur de l'apprentissage par renforcement car c'est l'exploration qui permet de détecter les comportements les plus bénéfiques mais uniquement l'exploitation est capable de les exploiter. La difficulté est donc de comprendre à quel moment l'agent a suffisamment exploré pour pouvoir exploiter ses connaissances.

Différentes politiques peuvent être exploitées dont les plus répandues sont:

- Approche indépendante de la fonction Q:

- **Approche gloutonne:** Cette approche est la plus élitiste possible et consiste à exploiter les actions associées aux meilleures valeurs Q dès le début. Elle est définie par:  $a_{s_t, \text{glout}} = \arg(\max(Q(s_t, a)))$ . Cette méthode est peu exploitable car peu efficace. Dès lors qu'un "chemin" aura été découvert, le modèle l'exploitera sans considérer les autres alternatives. Il est donc probable que le modèle n'exploite qu'un minimum local...

- **Approche  $\epsilon$ -gloutonne:** Cette approche corrige la faiblesse de la méthode gloutonne en limitant son élitisme. Ainsi, l'agent réalisera une action gloutonne avec une probabilité  $\epsilon$  sinon, il réalisera une action aléatoire. la valeur de  $\epsilon$  varie au cours du temps pour s'adapter à la situation. Ainsi, au début,  $\epsilon$  sera très faible pour favoriser l'exploration et augmentera progressivement pour finir par exploiter ses connaissances. Cette méthode permet l'étape d'exploration et présente des résultats convenables.

- Approche dépendante de la fonction Q:

- **Approche Softmax:** Cette politique repose sur la fonction Softmax. La probabilité de réaliser une action dans un état donné est proportionnelle à sa valeur par rapport à celles des autres actions de

```

Initialize  $Q(s, a)$  arbitrarily
Repeat (for each episode):
    Initialize  $s$ 
    Repeat (for each step of episode):
        Choose  $a$  from  $s$  using policy derived from  $Q$  (e.g.,  $\varepsilon$ -greedy)
        Take action  $a$ , observe  $r, s'$ 
         $Q(s, a) \leftarrow Q(s, a) + \alpha [r + \gamma \max_{a'} Q(s', a') - Q(s, a)]$ 
         $s \leftarrow s'$ ;
    until  $s$  is terminal

```

Figure 123: Algorithme du Q-Learning(0)

ce même état. Ainsi:

$$P(a_t|s_t) = \frac{Q(s_t, a_t)}{\sum Q(s_t, a_t)}$$

- **Approche Boltzmann:** Cette politique repose sur la distribution de Boltzmann. Elle peut être vue comme un cas particulier de la fonction Softmax. Son intérêt repose sur sa capacité d'adaptation durant l'apprentissage au contraire de Softmax qui est fixé. Elle est définie selon:

$$P(a_t|s_t) = \frac{e^{\frac{Q(s_t, a_t)}{\tau}}}{\sum e^{\frac{Q(s_t, a_t)}{\tau}}}$$

Avec  $\tau \in \mathcal{N}_+$ . Si  $\tau \rightarrow 0$ , le comportement de la politique tend à s'approcher de la politique aléatoire. Au contraire, si  $\tau$  augmente, elle s'approche de la méthode gloutonne. Il est nécessaire de faire varier  $\tau$  au fil de l'apprentissage afin de favoriser l'exploration en début d'apprentissage et progressivement tendre vers l'exploitation.

**Remarque:** Les algorithmes étudiés jusqu'alors possèdent "(0)" dans leurs noms. Cela signifie qu'il n'y a pas de considération des **traces d'éligibilité**. En effet, jusqu'alors, nous avons considéré qu'une récompense est due exclusivement à la dernière transition effectuée. Cette hypothèse est fausse car l'état t-2, t-3... ont aussi une importance sur la situation à l'instant t, ce qui est ignoré par un algorithme de type (0)<sup>196</sup>. L'idée des *traces d'éligibilité* est de propager la récompense aux transitions précédentes aussi car elles sont actrices de la situation actuelle. Cette éligibilité est quantifiée selon un niveau d'importance représenté par la proximité de la transition avec l'état actuel considéré. Cette approche est caractérisée par l'attribut  $(\lambda)$  ( $Q(\lambda)$ ,  $TD(\lambda)\dots$ ). Nous ne détaillerons pas ces algorithmes mais il est important de connaître leur existence.

---

<sup>196</sup>On peut grossièrement faire une analogie avec l'hypothèse de Markov !

### 13.1.1.8 Approximation des valeurs d'états

Jusqu'ici, nous avons étudié des méthodes qui reposent sur un nombre de visites important d'un même état<sup>197</sup>. Cette condition n'est pas tenable en cas d'espace d'états de très grande dimension (ou infini) car les capacités calculatoires ne le permettent pas. Une approche consiste donc à ne plus évaluer chaque valeur d'état possible mais à approximer une fonction capable de prédire n'importe quel état. Cette approche est très puissante car elle permet de prédire n'importe quel état, y compris des états jamais visités. Pour réaliser cela, il est nécessaire d'exploiter une méthode capable d'approximer des fonctions. Il est reconnu que les réseaux de neurones brillent dans cette tâche d'où leurs utilisations.

L'idée est simple. Un réseau reçoit une entrée associée à un état et la sortie du réseau produit une estimation de la valeur de cet état. En général, un état est caractérisé par différents attributs représentant l'entrée du réseau. Par exemple, dans le cas d'une voiture qui roule, on peut supposer les caractéristiques associées à la vitesse, l'état de la voiture, l'état de la route etc... Le réseau produit ainsi une valeur  $\hat{V}(s) = f(s, \theta)$  (ou  $\hat{Q}(s)$ ) avec  $\theta$ , poids du réseau (paramètres) qui pondèrent les caractéristiques de l'état observé. L'objectif du réseau est donc de minimiser l'erreur  $\|f(s, \theta) - V^*(s)\|$  en trouvant la combinaison  $\theta^*$  optimale. L'utilisation des réseaux de neurones soulèvent de nombreuses problématiques notamment l'apprentissage *online* du réseau, l'exploitation de différences temporelles et non d'une valeur labellisée au pouvoir explicatif juste et nette. Il existe de nombreuses approches qui dépassent le cadre de ce cours mais qu'il est important de considérer si vous voulez pleinement maîtriser ce domaine.

## 13.2 Deep Q-Learning

Avec l'avènement du Deep Learning, les algorithmes de renforcement reposent de plus en plus sur des structures neuronales. Dans le cadre de ce cours, nous exploiterons l'application des réseaux convolutifs pour la création de *bots* pour des jeux vidéos de type *ATARI*<sup>198</sup>. Ce n'est qu'un cas particulier de ce qui est possible de réaliser. Ce cours ne fournit qu'une sensibilisation à ce domaine à travers cet exemple mais un travail de recherche personnel sera nécessaire pour réaliser un état de l'art complet.

Un jeu, en apparence simple, peut présenter une complexité sous-jacente notamment liée aux différentes possibilités d'action réalisable dans le cadre d'une partie. Par exemple, le nombre de Shannon,  $10^{120}$ , est une estimation de la complexité du jeu d'échecs, c'est-à-dire du nombre de parties différentes réalisables dans le cadre d'une partie standard<sup>199</sup>. Les approches standards de l'apprentissage par renforcement reposent sur un apprentissage itératif basé sur

<sup>197</sup>la convergence "parfaite" vers  $V^*$  et  $Q^*$  demande une visite infinie de chaque état par exemple...

<sup>198</sup>Atari est une entreprise française (initialement américaine) de jeu vidéo fondée en 1972

<sup>199</sup>Ce nombre est à dissocier du nombre total de partie réalisable qui est nettement plus élevé

une visite de nombreuses fois d'un même état de l'environnement. Il est évident qu'étant donné la dimension massive de l'espace d'états, ces approches ne sont pas humainement réalisables. Il est donc nécessaire d'étudier une approche plus généraliste capable d'étudier des problèmes à haute dimension.

Le Deep Q-learning[68] exploite les capacités des réseaux neuronaux pour obtenir une approximation de la fonction Q qui définit la valeur de  $Q(s,a)$  dans un état s réalisant l'action a. Le réseau neuronal est ainsi capable d'évaluer chaque valeur  $Q(s,a)$  même si l'état s est peu ou pas visité (de même pour l'action réalisable dans cet état). Cette approche permet donc de résoudre la problématique de l'espace d'états à haute dimension.

Un robot-joueur possède les mêmes informations qu'un joueur humain. Ainsi, dans le cadre d'un jeu Atari, les informations à disposition sont les images de jeu<sup>200</sup>(en plus des récompenses associées à l'environnement telles que l'impact sur le score par exemple). Il est donc nécessaire d'adapter le réseau de neurones à cette spécificité d'où l'utilisation d'un réseau convolutif qui présente de bons résultats en analyse d'image.

Les données d'apprentissage du réseau correspondent à des tuples de données de la forme  $< s_t, a_t, r_t, s_{t+1} >$  où  $s_t$  correspond à un ensemble de N images successives (N=4 dans le cadre des jeux Atari),  $a_t$  est l'action réalisée dans l'état  $s_t$ ,  $r_t$  est la récompense obtenue et  $s_{t+1}$ , l'état de l'automate après son action représenté par N images. Ces tuples présentent deux risques majeurs. Le premier est la corrélation entre les données. Il est évident que l'image à l'instant t est liée à l'image à l'instant t+1 car c'est dans la continuité du jeu. Le second correspond à la distribution du jeu de données. Supposons le jeu du casse-briques, il est possible que l'automate, du fait de son initialisation aléatoire, favorise un des deux côtés de l'écran. Ceci est problématique car si non traitée, cette particularité peut inhiber l'exploitation d'une zone complète du jeu en favorisant le sur-apprentissage du réseau convolutif. Il est donc nécessaire de lutter contre la corrélation du jeu d'apprentissage et de permettre qu'il soit le plus représentatif de la distribution réelle du jeu observé.

Ces tuples sont obtenus au cours des expériences finies du jeu et sont conservés au sein du *replay memory*, jeu d'apprentissage de K tuples où K élevé pour limiter les variations de distribution du jeu d'apprentissage associées aux expériences possiblement très variées de l'automate vis-à-vis du jeu. L'idée est que le comportement aléatoire initial de l'automate va permettre une exploration satisfaisante et la dimension élevée du jeu d'apprentissage permettra de limiter le risque de sur-spécialisation des images conservées. L'apprentissage du réseau de neurones sera réalisé avec un minibatch de k tuples tirés au hasard dans le *replay memory* afin de limiter la corrélation entre les tuples qui est dan-

---

<sup>200</sup>Dans le cadre d'autres jeux, comme le Cartpole par exemple, l'information à disposition peut être présentée sous forme de données numériques telles que la vitesse, l'angle par rapport à la verticale etc...

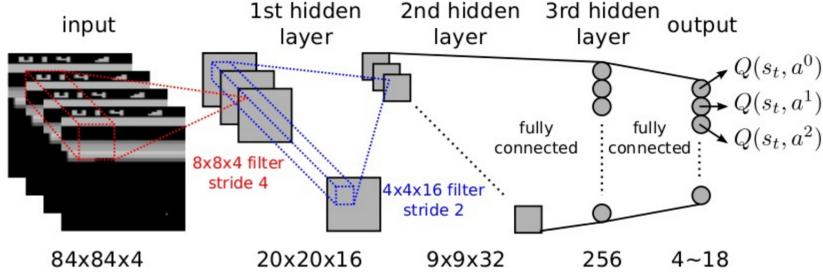


Figure 124: Algorithme du Deep Q-Learning

gérieuse pour l'apprentissage du réseau. Cette approche constitue l'*Experience Replay*.

Dans le cadre du Q-learning, nous pouvons définir:

$$Q_{i+1}(s_t, a_t) = E[r_t + \gamma * \max_{a_{t+1} \in A_{s_{t+1}}} Q_i(s_{t+1}, a_{t+1}) | s_t, a_t]$$

Ainsi, l'objectif du réseau est d'approximer une fonction qui, pour tout s et a, minimise la fonction de perte suivante (de type *Squared Error Loss*):

$$Loss = \frac{1}{2} \cdot \underbrace{[r_t + \gamma * \max_{a_{t+1}} (Q(s_{t+1}, a_{t+1}; \theta))]}_{\text{target}} - \underbrace{Q(s_t, a_t; \theta)}_{\text{prediction}}]^2$$

L'architecture du réseau est visible sur la Figure 124. Cette architecture souffre des limitations théoriques des réseaux de neurones. Ainsi, il n'y a pas de convergence théorique démontrée bien qu'empiriquement stable. De plus, la distribution des scores de l'automate sont très bruitées et témoignent d'une variance importante. Cette variance témoigne du manque de robustesse du modèle qui présente une trop grande sensibilité aux variations des observations. De plus, on peut noter qu'une variation de faible ampleur des poids du réseau agit de manière significative sur les performances de l'automate (en bien comme en mal). Néanmoins, l'évolution des Q-values tend à converger. Cette dernière observation est contestable car la diminution des taux d'apprentissage simule ce comportement sans véritablement corriger les risques de divergence réels. Ca soigne les symptômes sans s'attaquer aux causes en quelque sorte... Néanmoins, les performances du modèle dépasse ses concurrents (de l'époque !) et dépasse parfois, l'homme lui-même. Cet algorithme présente, cependant, une faiblesse sur des jeux qui demandent une stratégie à long terme. La notion de mémoire est donc à améliorer sur cette algorithme.

### 13.3 Prioritized Experience Replay

Cette approche propose un postulat où des expériences sont porteuses de plus d'informations que d'autres. La sélection des expériences stockées dans l'Experience

Replay étant aléatoire, il est probable que les expériences "riches" pour l'apprentissage se voient noyées dans l'ensemble des expériences pour la majorité peu riches. Afin de contrer ce problème, Prioritized Experience Replay[81] propose une approche afin de pondérer les expériences afin d'exploiter au mieux les expériences porteuses d'informations.

Pour juger de la pertinence d'une expérience, un indice de priorité est proposé. Il est défini par:  $p_i = |\delta_i| + \epsilon$  avec  $i$ , index de l'i-ième expériences,  $\delta$ , erreur entre la prédiction réalisée par le modèle et la valeur effective à obtenir et  $\epsilon$ , constante non nulle pour éviter qu'une expérience ait un indice de priorité nul.

Utiliser une approche *greedy* sur les indices de priorité (sélectionner que les expériences avec un fort indice) va imposer une sur-représentation de certaines expériences au détriment d'autres et de ce fait, favoriser l'overfitting du modèle. Pour limiter ce phénomène, un compromis entre une approche *greedy* et une sélection aléatoire est nécessaire. L'approche *Softmax* a donc été choisie pour réaliser ce compromis. Ainsi, à chaque expérience, nous associons une probabilité d'être sélectionnée  $P_i$  définie telle que  $P_i = \frac{p_i^\alpha}{\sum_k p_k^\alpha}$  avec  $p_i$ , indice de priorité de l'expérience i,  $k$ , nombre d'expériences et  $\alpha$ , constante qui définit l'importance de l'indice dans la détermination de la probabilité de sélection. Si  $\alpha$  est égale à 0, nous obtenons une sélection aléatoire et si  $\alpha \rightarrow \infty$ , la sélection est *greedy*.

Afin de limiter l'impact des *outliers* (une expérience avec une erreur significativement plus grande que les autres)<sup>201</sup>, il peut être nécessaire d'utiliser une autre métrique pour calculer l'indice de priorité. Ainsi, au lieu d'exploiter l'erreur de prédiction, l'indice reposera sur le *rang* de cette erreur. Ainsi, l'erreur la plus importante aura un rang de 1 et l'erreur la moins importante, un rang de k (en supposant k expériences). L'indice de priorité est donc défini par:  $\frac{1}{rank(i)}$ . Cette approche tend à être plus robuste grâce à son insensibilité aux outliers mais les deux favorisent une vitesse de convergence significativement plus forte qu'avec une approche aléatoire.

Prioritized Experience Replay introduit un biais du fait de sa sélection non aléatoire. La sélection prioritaire de certaines expériences va forcer le modèle à apprendre de nombreuses fois sur un sous-ensemble d'expériences alors que d'autres seront peu ou pas examinées. Ce phénomène risque de provoquer un overfitting important à terme. Afin de corriger cette faiblesse, il est nécessaire de limiter l'impact d'une expérience souvent exploitée.

Pour cela, la valeur de l'erreur du modèle associée à l'expérience i est redéfinie par  $w_i \delta_i$  avec  $w_i = (\frac{1}{N} * \frac{1}{P_i})^\beta$  où N, taille du batch d'apprentissage du modèle et b, facteur d'importance permettant de définir l'impact de la régulation sur

---

<sup>201</sup>Supposons un cas extrême où une expérience est infiniment plus grande que les autres. Cette expérience aura une probabilité quasi absolue d'être sélectionnée à chaque fois.

l'erreur produite par l'expérience. Une expérience souvent observée aura donc une influence régulière dans l'apprentissage du modèle mais son impact sera plus faible que celle des expériences plus rarement exploitées. Au début de l'apprentissage, on exploite une valeur de  $\beta$  proche de 0 car, à cette étape, la présence d'un biais n'est pas trop impactant. Lorsque le modèle commence à converger, réguler le biais est nécessaire et de ce fait, la valeur de  $\beta$  doit tendre vers 1. Ainsi, il sera pertinent, durant l'apprentissage, de faire varier continuellement  $\beta$  de 0 vers 1 afin d'exploiter le pouvoir de converge d'une approche *greedy* et la protection contre l'overfitting d'une approche stochastique.

### 13.4 Double Deep Q-Learning

Dans l'algorithme du Q-Learning, l'opérateur max utilise la même valeur pour choisir une action et l'évaluer. Cette particularité va favoriser la sélection de valeurs surestimées suite à un excès d'optimisme dû aux erreurs d'apprentissage. En effet, supposons un état où chaque  $Q$  doit avoir une valeur identique lors d'une convergence idéale du modèle. Durant l'apprentissage, la présence de l'opérateur max dans l'algorithme du Q-Learning va introduire un biais associé aux valeurs de  $Q$  encore bruitées selon les expériences de l'apprentissage. En effet, la valeur de  $Q$  avec la plus grande erreur positive sera sélectionnée. Ce phénomène est appelé *Overoptimism*. L'idée du Double Deep Q learning (DDQN)[28] est d'exploiter deux réseaux distincts pour évaluer une action et choisir l'action à réaliser. Cette approche permet ainsi de limiter l'optimisme de l'algorithme et d'être plus "modéré" sur les changements de comportements en limitant le biais de prédiction de  $\max_a(Q(s,a,w))$ . Cette approche est très utile en début d'apprentissage car le manque d'informations fausse la pertinence des valeurs de  $Q$ .

Supposons deux réseaux dont l'ensemble des poids est représenté par  $w_1$  et  $w_2$ . L'estimation de l'erreur par le DDQN devient donc:

$$Loss = \frac{1}{2} \cdot (r_t + \gamma Q(s_{t+1}, \arg\max_{a_{t+1}} Q(s_{t+1}, a_{t+1}, w_1), w_2) - Q(s_t, a_t; w))^2$$

Les deux modèles sont appris indépendamment et le réseau mis à jour choisi aléatoirement. Il est possible d'alterner le réseau qui prédit l'action à réaliser et sa valeur pour un apprentissage symétrique.

Pour des raisons de temps d'apprentissage et de coût matériel, cette approche n'est pas exploitée comme présentée précédemment. Une méthode subsidiaire permet de profiter de la majorité des bénéfices de l'approche *Double* tout en limitant les exigences de calculs. Pour cela, le second modèle n'est pas complètement dissocié du premier mais est équivalent à une copie du modèle initial dont les poids correspondent aux poids du modèle initial à un instant t-n. La prédiction de la valeur de  $Q$  sera réalisée avec le modèle ancien et la prédiction de l'action à réaliser, par le réseau à jour. Cette particularité permet de s'émanciper d'un double apprentissage qui est gourmand en ressources matérielles et temporelles

en se limitant à l'apprentissage d'un unique réseau.

Supposons 2 Q-network: un "à jour" ( $w$ ), un "ancien" ( $w^-$ ). ( $w$ ) sélectionne l'action à réaliser et ( $w^-$ ) évalue la valeur de l'action. L'apprentissage du réseau suit la relation suivante:

$$Loss = \frac{1}{2} \cdot (r_t + \gamma Q(s_{t+1}, \text{argmax}_{a_{t+1}} Q(s_{t+1}, a_{t+1}, w), w^-) - Q(s_t, a_t; w))^2$$

### 13.5 Dueling Deep Q-Learning

Le modèle Dueling (DDQN)[100] repose sur l'idée que la valeur de Q est une combinaison de deux fonctions: A(a,s) qui représente la pertinence de faire une action a dans un état s et V(s), la pertinence d'être dans un état S<sup>202</sup>. Cette séparation permet de considérer la qualité d'être dans un état indépendamment d'une action. Elle permet ainsi de mieux évaluer un état en l'émancipant d'une action associée. En effet, la plupart du temps, la valeur associée à la réalisation d'une action dans un état s n'a pas d'influence significative sur la valeur de l'état s. Pour considérer cette particularité, V et A sont calculées à partir de deux réseaux feed-forward issus d'une même source (couche de convolution ou un autre réseau feed-forward) et parfaitement isolés l'un de l'autre. Leurs sorties sont par la suite additionnées pour produire la valeur de Q.

Nous avons donc:

$$\begin{aligned} Q(s, a) &= V(s) + A(s, a) \\ V(s) &= E(Q(s, a)) \\ A(s, a) &= Q(s, a) - E(Q(s, a)) \\ E(A(s, a)) &= 0 \\ Q(s, a) &= V(s) \text{ and } A(s, a) = 0 \text{ if deterministic policy} \end{aligned}$$

Dans les faits, nous exploitons une valeur de A normalisée pour l'apprentissage soit:

$$Q(s, a, \theta, \alpha, \beta) = V(s, \theta, \beta) + (A(s, a, \theta, \alpha) - \frac{1}{|\mathcal{A}|} \sum_{a \in \mathcal{A}} A(s, a, \theta, \alpha))$$

Cette régularisation permet d'assurer que l'un des flux du réseau Dueling déterminera bien la valeur d'un état. Il est aussi possible de régulariser par la valeur max de  $A(s, a, \theta, \alpha)$  mais l'utilisation de la moyenne présente de meilleurs résultats.

Le réseau est illustré sur la Figure 125. Expérimentalement, il présente de meilleurs résultats que le DQN et le DDQN.

---

<sup>202</sup>Cette combinaison peut laisser penser à une approche model-based qui vise à étudier les spécificités de l'environnement

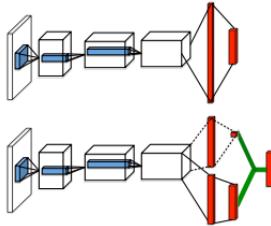


Figure 125: Algorithme du Dueling Deep Q-Learning

### 13.6 Double Dueling Q-Learning

le modèle Double Dueling Q-Learning (D-DDQN) est une approche qui unit le Double Q-Learning et le Dueling Q-Learning. Ainsi, DDQN exploitera deux réseaux distincts pour réaliser la prédiction de la valeur de Q et de l'action à réaliser (approche Double) et, au sein de ces réseaux, la prédiction de la valeur d'état et de la pertinence d'une action sera dissociée (approche Dueling).

Son efficacité tend à dépasser l'approche Double ou Dueling seule en profitant des bénéfices des deux approches.

## 14 Lecture et approfondissement

Si vous désirez approfondir votre compréhension théorique des Fondamentaux du Deep Learning<sup>203</sup>, les lectures suivantes peuvent orienter votre recherche. Ces liens fournissent une source vulgarisée mais de qualité afin d'éviter de se référer aux papiers de recherches. Néanmoins, pour une pleine compréhension, se référer à la bibliographie est nécessaire car ces liens ne couvrent pas l'intégralité des champs explorés dans ce rapport ni toutes leurs complexités:

### 14.1 Fondamentaux

- <http://cs231n.github.io/> (cours de l'Université de Stanford - Fondamentaux)
- <https://www.coursera.org/specializations/deep-learning> (MOOC<sup>204</sup> de Andrew Ng d'excellente qualité)
- [https://handong1587.github.io/deep\\_learning/2015/10/09/training-dnn.html](https://handong1587.github.io/deep_learning/2015/10/09/training-dnn.html) (liens pour des articles avancés)
- <https://isaacchanghau.github.io/2017/06/07/Loss-Functions-in-Artificial-Neural-Networks/> (Fonction de coût)

---

<sup>203</sup>Ces sources ont été exploitées pour réaliser cette introduction

<sup>204</sup>Les vidéos sont trouvables sur Youtube

- <http://ruder.io/optimizing-gradient-descent/> (Optimizer et régularisation)

## 14.2 Réseaux convolutifs

- Le papier [23] présente un bilan des structures générales des réseaux convolutifs et de leurs applications. C'est un article bien écrit et très instructif pour avoir un bilan des avancées récentes.
- <https://nrupatunga.github.io/convolution-2/#recap-from-part-1> (Approfondissement mathématique du calcul des convolutions)

## 14.3 Jeu de données d'apprentissage

- [https://en.wikipedia.org/wiki/List\\_of\\_datasets\\_for\\_machine\\_learning\\_research#Image\\_data](https://en.wikipedia.org/wiki/List_of_datasets_for_machine_learning_research#Image_data)
- <https://deeplearning4j.org/opendata>
- <http://deeplearning.net/datasets/>

## 14.4 Recueil thématique d'articles de recherche

### 14.4.1 Multi-thématique

- <https://github.com/floodsung/Deep-Learning-Papers-Reading-Roadmap>
- [https://medium.com/tag/\[machine-learning, deep-learning, computer-vision\]](https://medium.com/tag/[machine-learning, deep-learning, computer-vision]) (Site proposant des articles de très grandes qualités sur les dernières nouveautés de l'Intelligence Artificielle et de la Data Science. Articles vulgarisateurs très pertinents associés à des auteurs-rechercheurs de réputation)

### 14.4.2 Analyse d'image - Toute thématique

- <https://github.com/kjw0612/awesome-deep-vision#object-detection>
- <https://github.com/abhineet123/Deep-Learning-for-Tracking-and-Detection>
- <https://github.com/jbhuang0604/awesome-computer-vision>

## 14.5 Image Segmentation

- [http://cs231n.stanford.edu/slides/2016/winter1516\\_lecture13.pdf](http://cs231n.stanford.edu/slides/2016/winter1516_lecture13.pdf) (Cours de l'Université de Stanford - Discute aussi de la notion d'*Attention*)

## 15 Installation de Tensorflow et Keras

Keras est un librairie pour développer des algorithmes de Deep Learning. Cette librairie (codée en Python) propose une API haut niveau pour simplifier le développement et l'implémentation de réseaux profonds. Cette librairie simplifie l'exploitation d'une autre librairie plus bas niveau tel que Theano, Tensorflow ou encore CNTK. Il est important de comprendre que Keras ne propose pas une API pleinement fonctionnelle de manière isolée mais se comporte comme un *wrapper* d'une autre librairie dédiée. De part sa facilité d'utilisation et son approche modulaire (facilitant l'évolution de l'implémentation du réseau), Keras est une librairie grandement utilisée dans les étapes d'expérimentation et de prototypage.

**Note:** Keras supporte l'exploitation CPU et GPU.

**Important:** Nous utiliserons Tensorflow comme base pour Keras. Keras supporte d'autres librairies. Référez-vous à la documentation officielle pour plus de détails si vous ne souhaitez pas Tensorflow comme base.

### 15.1 Installation de Tensorflow pour utilisation GPU sur Ubuntu

1. S'assurer d'avoir une carte compatible CUDA: <https://developer.nvidia.com/cuda-gpus>.

**Attention:** Si *Compute Capability* < 3.0, votre carte n'est pas exploitable.

2. S'assurer de l'installation à jour du driver de la carte graphique
3. Installer CUDA: <https://developer.nvidia.com/cuda-downloads>

**Attention:** A ce jour, Tensorflow ne supporte pas la version 9.1. Pour la version la plus à jour de Tensorflow, installez CUDA 9.0: <https://developer.nvidia.com/cuda-90-download-archive>

**Attention:** Utilisez `sudo apt-get install cuda-9-0` pour installer la version 9.0. Si vous suivez le processus indiqué, vous installerez la dernière version sortie soit la 9.1 !

4. Installer CUDNN: <https://developer.nvidia.com/cudnn>.

**Attention:** la création d'un compte est nécessaire !

**Attention:** Installez une version de CUDNN en adéquation avec votre version de CUDA ! Par exemple, le couple <*Cuda9.0,Cudnn7.0.5*> est fonctionnel

5. Redémarrez votre machine
6. **Si *Compute Capability* de votre carte > 3.0:**

Installez Tensorflow pour usage GPU: `sudo pip3 install tensorflow-gpu`

**Attention:** Le paquet *tensorflow* (sans "-gpu") installe tensorflow dans sa version CPU uniquement !

**Si *Compute Capability* de votre carte = 3.0:**

**Attention:** L'installation proposée n'est pas *fiable*. Elle ne peut être garantie de fonctionner !

**Attention:** Cette installation demande la présence du paquet Numpy de python sinon Bazel ne fonctionnera pas correctement.

- (a) Installez Git: `sudo apt-get install git`
- (b) Téléchargez le code source de Tensorflow: `git clone https://github.com/tensorflow/tensorflow`
- (c) Choisissez la branche de git (release) correspondant à la version désirée. Ne rien faire si vous souhaitez conserver la branche *master*. Si vous désirez changer de release: `git checkout r1.0` pour la version r1.0 par exemple.

La branche *master* a été fonctionnelle. Certains préconisent la branche *r1.0* - Non fonctionnelle sur mon expérimentation.

**Attention:** Cette étape est délicate et incertaine. Certaines release marchent chez certains, pas pour d'autres. Elles dépendent des versions de différents paquets (notamment le paquet *Bazel*). En cas de problème durant l'installation, faites varier la branche de manière empirique...

- (d) Installer Bazel: <https://docs.bazel.build/versions/master/install-ubuntu.html#install-on-ubuntu>
- (e) Déplacez-vous dans le répertoire de tensorflow obtenu via git
- (f) Lancez la configuration de l'installation souhaitée: `./configure`
- (g) Répondez aux questions selon vos préférences. Une attention particulière pour les questions suivantes:

**Attention:** Les questions peuvent varier selon la version de tensorflow !

- Please specify the location of python: `/usr/bin/python3` si python3 souhaité

- Do you wish to build TensorFlow with CUDA support?: Yes
  - Please specify the Cuda SDK version you want to use: selon votre configuration
  - Please specify the Cudnn version you want to use: selon votre configuration
  - Specify comma separated Cuda compute capabilities: **3.0**
- (h) Construisez le package-pip:
- CPU only:  

```
# sudo bazel build -config=opt //tensorflow/tools/pip_package:build_pip_package
```
  - Avec GPU:  

```
# sudo bazel build -config=opt -config=cuda //tensorflow/tools/pip_package:build_pip_package
```
- Attention:** Bazel ne gère pas les accents. Assurez-vous que le dossier Tensorflow ne se trouve pas dans un dossiers au nom accentués (notamment Téléchargements).
- (i) Créez la seed:  

```
# bazel-bin/tensorflow/tools/pip_package/build_pip_package /tmp/tensorflow_pkg
```
- (j) Installez le paquet:  

```
# sudo pip install /tmp/tensorflow_pkg/tensorflow-1.6.0-py2-none-any.whl
```
- Attention:** le nom du paquet dépend de la version. Vérifiez-le dans le répertoire *tmp*
- (k) Vérifiez la détection du GPU par tensorflow dans une console Python3:

Listing 1: Visualisation sous Tensorflow des GPU disponibles

```
1      from tensorflow.python.client import device_lib
2      print(device_lib.list_local_devices())
```

En cas de problème, ces liens peuvent orienter vos pistes de recherche:

- [https://www.tensorflow.org/install/install\\_sources#clone\\_the\\_tensorflow\\_repository](https://www.tensorflow.org/install/install_sources#clone_the_tensorflow_repository)
- <https://github.com/tensorflow/tensorflow/issues/7542>
- <https://github.com/nicolaifsf/Installing-Tensorflow-with-GPU>

## 15.2 Installation de Keras

1. Installez Keras: *sudo pip3 install keras*
2. Vérifiez la liaison avec tensorflow dans une console Python3:



Figure 126: Logo des framework Tensorflow et Caffe

Listing 2: Visualisation sous Keras des GPU disponibles

```

1   from keras import backend as K
2   K.tensorflow_backend._get_available_gpus()

```

## 16 Tensorflow, théorie et exemple

### 16.1 Introduction

*Tensorflow*[2] est un framework développé par Google en Novembre 2015. Conçu initialement pour l'optimisation de calculs complexes et massifs<sup>205</sup>, son utilisation a rapidement été orientée vers le Deep Learning qui présente des spécificités similaires. Dans ce domaine, il s'est rapidement imposé comme l'une des librairies les plus performantes et populaires. Aujourd'hui, ce framework est reconnu pour ses performances, sa fiabilité<sup>206</sup> et sa popularité grandissante qui lui permet d'avoir des mises à jours rapides et une communauté active nécessaire dans un secteur de recherche aussi évolutif. Bien qu'il existe de nombreux framework concurrents, Tensorflow semble être en voie de devenir l'outil de référence dans l'industrie. Néanmoins, le framework *Caffe*<sup>207</sup>[41] semble le plus employé dans le secteur académique.

Tensorflow est un framework *bas-niveau*. De nombreuses API ont été créées afin de faciliter son utilisation. Les plus connues sont *TF layers*<sup>208</sup> et *Keras*, développée par un chercheur de Google. Il possède une interface en Python<sup>209</sup>, Java, Go, C++ et supporte le calcul sur GPU, indispensable dans le cadre du Deep Learning.

### 16.2 Architecture d'exécution

**Important:** L'architecture de tensorflow présente une complexité importante qui ne sera pas détaillée dans ce rapport. Si vous désirer approfondir votre

---

<sup>205</sup>Notamment pour le calcul scientifique en physique fondamentale

<sup>206</sup>Google a porté la majorité de ses outils logiciels sur ce framework

<sup>207</sup>Caffe2 a été développé pour améliorer Caffe notamment les capacités de portabilité et de flexibilité

<sup>208</sup>Intégrée à Tensorflow

<sup>209</sup>L'interface en Python est la plus complète et utilisée

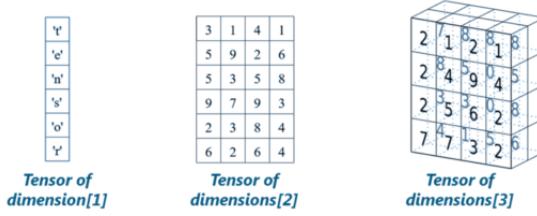


Figure 127: Représentation d'un Tensor de dimension 1, 2 et 3

compréhension du framework, veuillez vous référer à la documentation officielle:  
<https://www.tensorflow.org/extend/architecture>.

### 16.2.1 Généralités

Tensorflow propose une architecture définissant un **graphé d'exécution**. Un noeud définit une action à réaliser (*Operation*) et une arrête, une donnée (*Tensor*). Une *Operation* exploite un ou plusieurs *Tensor* (ou même aucun) en entrée, réalise une action à partir des données considérées et renvoie un ou plusieurs *Tensor* (ou aucun). Ainsi, par exemple, dans le cadre d'un perceptron unitaire, la donnée d'entrée serait associée à un *Tensor*, de même que la sortie du neurone. Le produit matriciel pour calculer le logit serait associé à une *Operation* par exemple<sup>210</sup>.

### 16.2.2 Tensor

Un *Tensor* définit une représentation standardisée et généralisée d'une variable vectorisée à n dimensions. Ainsi, pour n=0, le *Tensor* serait un scalaire, pour n=1, un vecteur, pour n=2, une matrice etc... Supposons l'exemple d'un mini-batch d'images RGB alors le *Tensor* associé serait de la forme [nombre\_image, hauteur, largeur, channel<sup>211</sup>]. On obtient ainsi un *Tensor* à 3 dimensions (cube de nombres). Un exemple illustratif est visible sur la Figure 127.

### 16.2.3 Instanciation et Session

Le graphé d'exécution est automatiquement géré par *Tensorflow*. Cette automatisation permet une meilleure optimisation d'exécution et surtout, une parallélisation facilitée. Cette spécificité implique une approche différente de codage. Dans un premier temps, on définit l'architecture du graphé (les *Tensors* exploités, les *Operations* à réaliser, la segmentation du graphé<sup>212</sup>...) puis

<sup>210</sup>L'ajout du biais serait vu comme une autre *Opération* définie par une simple addition

<sup>211</sup>Le channel correspond à la couleur considérée soit rouge, vert ou bleu

<sup>212</sup>Il est possible de considérer des sous-ensembles d'entités du graphé. Ils font partis du même graphé mais dans un espace de nommage distincts. Cette spécificité est capitale dans le cadre d'un réseau où les poids apprennent de manière différée.

nous exploitons ce graphe en interagissant avec lui par la détermination des variables à considérer et des *Operations* à réaliser, indépendamment de son fonctionnement interne. Ainsi, Tensorflow s'occupe de réaliser l'interface avec les supports matériels (notamment GPU) et de l'optimisation en interne.

Ce fonctionnement implique de définir un environnement spécifique: une **Session**. Pour pouvoir exploiter le graphe, ce dernier dans être lancé dans une *Session*. C'est la *Session* qui gère le fonctionnement du graphe. Il est possible de créer différentes *Session* à partir d'une même architecture de graphe. Néanmoins, les deux graphes seront parfaitement indépendants. Dans le cas d'instanciation de variables aléatoires, leurs valeurs seront différentes par exemple.

Ainsi, la structure d'un code sous *Tensorflow* se découpe en deux parties:

- Une partie d'**architecture** où le réseau est défini sans être instancié. Seules les différentes entités et leurs relations sont construites.
- Une partie d'**exécution** où une *Session* est définie et qui exploite les différentes *Operations* proposées par le graphe.

### 16.3 Fondamentaux du framework

#### 16.3.1 Déclaration des variables du graphe

Pour déclarer des variables de graphes, il existe 3 approches principales dépendant de la nature de la variable et de son évolution au fil du temps.

- Tensor **constante**: Lorsqu'une variable est une constante (invariable au fil du temps), son appel est réalisé par la fonction *constant*. Ainsi, pour déclarer un vecteur constant défini par [1, 2], on déclarera:

Listing 3: Déclaration d'une constante avec Tensorflow

```

1   import tensorflow as tf
2   const = tf.constant([1, 2])
3   # 1 et 2 sont les valeurs d'un vecteur 1-D,
4   # non la dimension du Tensor !

```

- Tensor **variable**: Lorsqu'un Tensor est variable au cours de l'exécution d'une session, on utilise la fonction *Variable*. cette fonction permet de définir un Tensor qui sera maintenu lors d'une exécution avant de pouvoir être mis à jour (ou non) lors de prochaines itérations du graphe (ou l'itération actuelle). Il est important de noter que le Tensor est conservé jusqu'à la fin de la *Session*. Ainsi, entre plusieurs appels d'une même *Operation* au sein d'une même *Session*, le Tensor est conservé et non réinitialisé. Ces variables servent à définir les Tensors associés aux différents poids du réseau par exemple. Pour déclarer un Tensor variable

de dimension [20,30] généré par un loi normale tronquée ( $\sigma = 0.1$ ), on déclarera:

Listing 4: Déclaration bas niveau d'un Tensor variable avec Tensorflow

```

1   import tensorflow as tf
2   # 20 et 30 correspondent aux dimensions du Tensor.
3   # Ainsi, [20,30] definira une matrice de dimension 20*30
4   # Il ne faut pas confondre dimensions et valeurs !
5   w0p = tf.Variable(tf.truncated_normal([20,30], stddev=0.1),
6                     name="w0")

```

L'attribut *name* possède une importance subtile. En effet, on peut voir que la variable possède un nom associé à Tensorflow en interne (via l'attribut *name*) et à Python, via l'assignation standard. La variable issue de Python n'est qu'un pointeur vers la vraie variable de Tensorflow. Cette différence n'a pas d'incidence sur le fonctionnement direct du graphe car les deux noms peuvent être utilisés<sup>213</sup>. Cependant, la variable de Tensorflow est permanente au modèle alors que la variable Python n'a d'existence qu'à travers le script exécuté. Ainsi, supposons une sauvegarde du modèle et une réutilisation dans un nouveau script. Le nom *w0* aura toujours une existence et *w0p* non. Ainsi, pour toute définition interne au graphe, il est préférable d'utiliser les variables de graphe (via l'attribut *name*) alors que dans une *Session*, les appels étant extérieurs au graphe, les variables déclarées avec Python sont utilisées pour faciliter l'utilisation et la lisibilité.

La fonction utilisée pour créer le Tensor est une fonction bas niveau. Il existe une autre méthode plus haut niveau permettant une plus grande souplesse d'utilisation. Cette fonction est *get\_variable*.

Listing 5: Déclaration haut niveau d'un Tensor variable avec Tensorflow

```

1   import tensorflow as tf
2   # Si un Tensor de nom (dans le graphe) "w0" et
3   # de dimension (2,3) n'existe pas, il est construit,
4   # sinon on utilise le Tensor existant associé aux paramètres
5   w0p = tf.get_variable("w0", (2,3))

```

- **Tensor sans valeur définie:** Ce type de Tensor est comparable à un Tensor Variable sans valeur fixée. Il s'agit d'une variable qui recevra ses valeurs *plus tard* qu'à son initialisation. Ce type de Tensor est classiquement employé pour définir les valeurs d'entrée du réseau, i.e le minibatch. En effet, la matrice<sup>214</sup> représentant le minibatch est indépendante du graphe et définie à chaque exécution du graphe. Il n'y a donc aucune conservation des valeurs du minibatch. Ce Tensor est donc érasé

---

<sup>213</sup>Ils ciblent la même chose

<sup>214</sup>On va supposer le cas d'un ensemble d'images

à chaque itération par une nouvelle valeur. Il n'est pas indispensable de définir une valeur pour ce Tensor si il n'est pas utilisé durant une itération. Dans le cas contraire, lui associer une valeur est une nécessité. La fonction utilisée pour déclarer ce type de variable est *Placeholder*. Supposons la création d'un Tensor chargé de récupérer les valeurs d'un minibatch de taille indéterminée et composé d'images grayscale de dimension 28\*28. On obtient:

Listing 6: Déclaration d'un Placeholder

```

1 import tensorflow as tf
2 # On stockera les valeurs au format float32
3 x_ = tf.placeholder(tf.float32, [None, 28, 28, 1])
4 # None signifie que le nombre associé à cette dimension est
5 # non fixe. Sa valeur peut donc varier au fil des iterations.
6 # Dans le cas d'un minibatch, le nombre d'images du minibatch
7 # est un hyperparametre. Le reseau doit donc etre capable de
8 # supporter toute valeur et non juste une valeur unique.
9 # Il ne faut pas abuser du None car le gain de souplesse favorise
10 # l'instabilite due aux tolerances des formats de donnees

```

### 16.3.2 Nomination des variables

La structure nominative des variables suit une architecture en arbre. Ainsi, pour avoir accès à une variable, il est nécessaire de connaître son *path* et la position relative (offset) de l'observateur. Par défaut, toute création de variable est faite à la racine du graphe mais cette spécificité pose problème dans le cas où nous voudrions créer un sous ensemble de variables dissociées des autres. Ce genre d'action est très utile dans le cas d'un entraînement indépendant de certains poids<sup>215</sup>.

Afin de créer un noeud dans le graphe, deux méthodes sont utilisées: *name\_scope()* et *variable\_scope()*. Commençons par étudier *variable\_scope()* avec un exemple:

Listing 7: Gestion du namespace d'une variable par variable\_scope()

```

1 import tensorflow as tf
2                                         # Path+nom du Tensor
3 # Creation du Tensor a la racine
4 tf.get_variable("bar", (2,3))           # bar:0
5
6 # Creation du tensor dans le sous noeud scope1
7 with tf.variable_scope("scope1"):
8     bar1 = tf.get_variable("bar", (2,3))  # scope1/bar:0

```

---

<sup>215</sup>C'est utilisé avec les GAN par exemple

```

9
10    # Création du Tensor dans le sous noeud scope 2
11    with tf.variable_scope("scope2"):
12        bar2 = tf.get_variable("bar", (2,3))  # scope2/bar:0
13
14    # Récupération de la variable bar du sous noeud scope2
15    # reuse=True signifie qu'on accepte de réutiliser une
16    # variable déjà utilisée durant l'itération en cours
17    # "" nous place à la racine de l'arbre de variables
18    with tf.variable_scope("", reuse=True):
19        bar3 = tf.get_variable("scope2/bar")  # scope2/bar:0
20
21    # Récupération de la variable bar du sous noeud scope1
22    with tf.variable_scope("scope1", reuse=True):
23        bar4 = tf.get_variable("bar")          # scope1/bar:0

```

Cette approche permet ainsi de créer des sous-ensembles de variables afin de faciliter leur segmentation. Ainsi, il est facile d'isoler différentes variables pour ne pas toutes les considérer lors d'actions spécifiques. L'approche par `name_scope()` est similaire mais présente une subtilité. En effet, lors de l'utilisation de `tf.get_variable`, le sous-noeud défini n'est pas considéré. Observons l'exemple suivant:

Listing 8: Gestion du namespace d'une variable par `name_scope()`

```

1  import tensorflow as tf
2
3  with tf.name_scope("my_scope"):
4      v1 = tf.get_variable("var1", [1], dtype=tf.float32)
5      v2 = tf.Variable(1, name="var2", dtype=tf.float32)
6      a = tf.add(v1, v2)
7
8      print(v1.name)  # var1:0
9      print(v2.name)  # my_scope/var2:0
10     print(a.name)  # my_scope/Add:0
11     # La création ou récupération de la variable via get_variable()
12     # se fait indépendamment de name_scope mais les créations via
13     # Variable() ou opération de graphe sont considérées par le scope
14
15    with tf.name_scope("foo"):
16        with tf.variable_scope("var_scope"):
17            v = tf.get_variable("var", [1])
18    with tf.name_scope("bar"):
19        with tf.variable_scope("var_scope", reuse=True):
20            v1 = tf.get_variable("var", [1])

```

```

21
22     assert v1 == v
23     print(v.name)    # var_scope/var:0
24     print(v1.name)   # var_scope/var:0
25     # On observe que v1 == v malgré un name_scope() différent

```

L'utilité de `name_scope` n'est pas évidente. Cette fonction est utile pour délimiter différentes parties du réseau tout en permettant à des variables d'être facilement transférées entre chacune de ces parties. On peut comparer cette approche avec la logique qui nous pousse à créer des variables globales indépendantes de l'environnement local d'une fonction par exemple.

Afin d'obtenir une liste de variables d'un sous-noeud, une méthode simple est disponible:

Listing 9: Sélection par condition d'un sous-ensemble de variable

```

1  import tensorflow as tf
2
3  # Creation des variables et des sous-noeuds
4  with tf.variable_scope("scope1"):
5      v1 = tf.get_variable("v1", [1])
6  with tf.variable_scope("scope2"):
7      v2 = tf.get_variable("v2", [1])
8
9  # Récupération du nom des variables du scope1
10 # tf.GraphKeys.TRAINABLE_VARIABLES correspond à la liste
11 # intégrale de toutes les variables dites entrainables donc
12 # non constantes tels que les poids ou les biais des neurones
13 # ATTENTION: il est possible de définir un poids comme non
14 # entrainable. De ce fait, il ne sera plus dans la liste de
15 # tf.GraphKeys.TRAINABLE_VARIABLES. Voir
16 # "https://www.tensorflow.org/api_docs/python/tf/GraphKeys"
17 # pour plus de détails sur les différents sous-groupes définis
18 # 'scope1' est le paramètre de discrimination choisi
19 value = tf.get_collection(tf.GraphKeys.TRAINABLE_VARIABLES, 'scope1')
20 # scope1/v1:0

```

### 16.3.3 Création d'une Session

Afin d'exécuter les *Operations* d'un graphe, il est nécessaire de l'instancier dans une *Session*. Une *Session* exploite des ressources et ne la restitue qu'à la fin de son exécution. Il est donc important de veiller à clore la Session ou il existe un risque de saturation de la mémoire ou des capacités de calculs du support

matériel.

Lorsqu'une *Session* est ouverte, elle va réaliser l'ensemble des *Operations* présentes dans son environnement. Le calcul d'*Operations* peut être récursif. Ainsi, si une *Operation* fait référence à une autre *Operation*, cette dernière sera considérée.

Listing 10: Création d'une Session

```
1 import tensorflow as tf
2
3 # Création d'un vecteur
4 vec1 = tf.constant([1, 1])
5
6 # Création d'un vecteur
7 vec2 = tf.constant([2, 2])
8
9 # Operation: additionner les deux vecteurs
10 somme = tf.add(vec1, vec2)
11 # Important: à ce niveau, l'addition n'est pas réalisée !
12 # Elle est définie dans le graphe uniquement.
13 # Le calcul sera réalisé après un appel dans la Session
14
15 # Création d'une Session
16 sess = tf.Session()
17
18 # Appel de la fonction run() pour réaliser une Operation
19 result = sess.run(somme)
20 # result contient le résultat de l'Operation somme
21
22 # Plusieurs Operations peuvent être réalisées avec un appel
23 res_somme, res_vec1 = sess.run([somme, vec1])
24 # res_somme contient le résultat de l'Operation somme
25 # res_vec1 a récupéré la valeur de vec_1 sans transformation
26
27 # Ferme la Session
28 # Cette étape est capitale dans le cadre de multiples réseaux
29 # afin d'éviter la saturation de la mémoire qui est facile à
30 # atteindre avec des graphes de Deep Learning
31 sess.close()
```

Il est possible de réaliser un bloc *with* à fin d'automatiser la destruction de la *Session* à la fin de son exécution. Cette approche est préférable car plus stable et plus lisible.

Listing 11: Création d'une Session avec un bloc "with"

```
1 import tensorflow as tf
2
3 # Création d'un vecteur
4 vec1 = tf.constant([1, 1])
5
6 # Création d'un vecteur
7 vec2 = tf.constant([2, 2])
8
9 # Opération: additionner les deux vecteurs
10 somme = tf.add(vec1, vec2)
11 # Important: à ce niveau, l'addition n'est pas réalisée !
12 # Elle est définie dans le graphe uniquement.
13 # Le calcul sera réalisé après un appel dans la Session
14
15 # Création d'une Session
16 with tf.Session() as sess:
17     # Appel de la fonction run() pour réaliser une Opération
18     result = sess.run(somme)
19     # result contient le résultat de l'Opération somme
20
21     # Plusieurs Opérations peuvent être réalisées avec un appel
22     res_somme, res_vec1 = sess.run([somme, vec1])
23     # res_somme contient le résultat de l'Opération somme
24     # res_vec1 a récupéré la valeur de vec_1 sans transformation
```

#### 16.3.4 Application aux réseaux de neurones: l'exemple du Perceptron multicouche

Afin d'illustrer la partie précédente, nous allons développer un Perceptron multicouche à une couche cachée (200 neurones) sur le jeu de données MNIST<sup>216</sup>. Pour cet exemple, nous n'exploiterons pas d'API haut niveau de Tensorflow.

**Attention:** Ce réseau est à but pédagogique. Il n'a pas vocation à être performant !

Dans un premier temps, il est nécessaire de définir le graphe représentant le réseau. La première étape consiste à définir les variables nécessaires au réseau. Ainsi, il est nécessaire de définir les différents poids, biais, entrées du réseau. En entrée du réseau, le perceptron multicouche étant un algorithme d'apprentissage supervisé, il est nécessaire de lui donner une donnée et son label. Le réseau aura donc deux entrée.

---

<sup>216</sup>MNIST est un jeu de données représentant les chiffres de 0 à 9. Il est une référence dans les benchmarking de réseau bien que très simpliste

Listing 12: Perceptron multicouche: Initialisation des variables

```
1 # Dictionnaire des poids du réseau
2 # Initialisation via une distribution normale
3 # 3 couches: entrée, cachée, sortie
4 # 1) 200, 2) 100, 3) 10
5
6 weights = {
7     # [784, 200] -> 784 entrées et 200 neurones sur la couche
8     'w0': tf.Variable(tf.random_normal([784, 200], stddev=0.1),
9                         name="w0"),
10    # La sortie précédente est l'entrée de la couche suivante
11    # Il y a donc 200 entrées par neurones dans la couche cachée
12    'w1': tf.Variable(tf.random_normal([200, 100], stddev=0.1),
13                      name="w1"),
14    # Il y a 10 neurones sur la couche de sortie car il y a
15    # 10 classes possibles. Le réseau va donc prédire
16    # la probabilité de chaque label pour une image donnée
17    'w2': tf.Variable(tf.random_normal([100, 10], stddev=0.1),
18                      name="w2")
19 }
20
21 # Dictionnaire des biais du réseau
22 # Initialisation définie sur 0.1 pour chaque biais
23
24 bias = {
25     'b0': tf.Variable(tf.ones([200]) / 10, name="b0"),
26     'b1': tf.Variable(tf.ones([100]) / 10, name="b1"),
27     'b2': tf.Variable(tf.ones([10]) / 10, name="b2")
28 }
29
30 # Minibatch d'apprentissage pour le réseau
31
32 # On utilise None si le nombre d'image est variable
33 # Ce nombre est souvent un hyperparamètre d'ou
34 # la présence de None. Si le nombre est fixe et
35 # invariable, None peut être remplacé par la valeur
36 # voulue de taille de minibatch.
37
38 # [None, 28, 28, 1] -> nombre, hauteur, largeur, channel
39 # Ce format de Tensor est celui associé au jeu de données
40 # durant l'extraction des informations
41 x_ = tf.placeholder(tf.float32, [None, 28, 28, 1])
42
43 # Label d'apprentissage pour le réseau
44 # 10 = nombre de classe possible [0-9]
```

```

45     # One hot Encoding utilise pour formater la donnee
46     y_ = tf.placeholder(tf.float32, [None, 10])
47
48     # Fonction d'appel pour initialiser les variables definies
49     init = tf.global_variables_initializer()

```

Les variables sont ainsi définies. Il est donc possible de créer le réseau. Pour cela, nous utiliserons la fonction ReLu comme fonction d'activation sauf pour la couche de sortie où la fonction softmax sera utilisée. En effet, nous voulons une représentation probabiliste pour chaque label et le problème n'est pas à deux classes donc la fonction sigmoïde n'est pas recommandée.

Listing 13: Perceptron multicouche: Architecture du réseau

```

1      # Le minibatch doit etre reformaté pour avoir la structure
2      # décrite dans la section précédente. En effet, nous
3      # voulons une matrice avec deux dimensions et qu'une ligne
4      # représente une image et une colonne, un pixel
5
6      # -1 est un indice permettant de ne pas déclarer
7      # explicitement la valeur de cette dimension.
8      # La valeur sera celle permettant de réaliser
9      # le redimensionnement tout en respectant les
10     # autres dimensions indiquées.
11
12     # Dans cette situation, en imposant 784, on
13     # forcera -1 à être redéfini par le nombre
14     # d'image dans le minibatch car il y a
15     # 784*nbr_image pixels dans le minibatch
16
17     # -1 est complémentaire avec None et permet
18     # une souplesse d'utilisation pour éviter de
19     # "coder en dur" le réseau
20     x = tf.reshape(x_, [-1, 784])
21
22     # On réalise le produit matriciel
23     # entre le poids et les entrées.
24     # On ajoute le biais par simple addition
25     # On applique la fonction d'activation ReLu
26     y0 = tf.nn.relu(tf.matmul(x, weights_[ 'w0' ]) + bias_[ 'b0' ])
27     y1 = tf.nn.relu(tf.matmul(y0, weights_[ 'w1' ]) + bias_[ 'b1' ])
28
29     # IMPORTANT: on observe que la sortie n'a pas été soumise
30     # à une fonction d'activation. Cette spécificité est un cas

```

```

31     # particulier qui va etre expliqué par la suite
32     y_output = tf.matmul(y1, weights_[‘w2’]) + bias_[‘b2’]

```

L'architecture du réseau est maintenant déclarée. Il ne reste qu'à déterminer la fonction de coût et la méthode d'optimisation choisie pour entraîner le réseau. On utilisera la Cross-Entropy comme fonction d'activation (problème de classification multiclass) et l'optimizer Adam pour l'apprentissage.

Listing 14: Perceptron multicouche: Fonction de coût et optimisation

```

1      # Cette fonction prend un logit en argument,
2      # applique la fonction softmax puis calcule
3      # la Cross-Entropy de la sortie
4
5
6      # La spécificité de cette fonction est d'éviter
7      # la problématique du log(0) qui provoque une
8      # erreur fatale pour l'apprentissage du réseau
9      cross_entropy = tf.nn.softmax_cross_entropy_with_logits
10         (logits=y_output, labels=y_)
11
12     # Application de la rétropropagation du gradient: Tensorflow
13     # automatise la maj des poids concernés
14     train_step = tf.train.AdamOptimizer(0.0002).minimize(cross_entropy)
15
16     # Statistiques générales du réseau
17
18     # cross_entropy est un vecteur contenant l'erreur pour chaque image
19     # Il est utile de calculer la moyenne pour évaluer la tendance
20     # d'évolution du modèle
21     cross_entropy_result = tf.reduce_mean(tf.cast(cross_entropy, tf.float32))
22
23     # Il est utile d'avoir l'accuracy du modèle pour évaluer
24     # sa performance métier. Pour obtenir l'accuracy, on évalue
25     # le nombre de bonne prédiction sur le total
26
27     # tf.argmax permet de récupérer l'indice de la plus grande
28     # probabilité de label. Si l'indice correspond avec le label
29     # d'apprentissage, la prediction est bonne
30     is_correct = tf.equal(
31                 tf.argmax(tf.nn.softmax(y_output), 1),
32                 tf.argmax(y_, 1)
33             )
34

```

```

35     # Calcule l'accuracy du modèle
36     # (en se basant sur le minibatch fourni)
37     accuracy = tf.reduce_mean(tf.cast(is_correct, tf.float32))

```

Le graphe du réseau est maintenant terminé: les variables sont déclarées, le réseau construit et ses fonctions d'apprentissage définies. Il ne reste qu'à créer la *Session*.

Listing 15: Perceptron multicouche: Création de la Session

```

1      epoch = 30000
2      minibatch_size = 200
3      display_epochs = 100
4      mnist = mnist_data.read_data_sets("data",
5                                         one_hot=True,
6                                         reshape=False,
7                                         validation_size=0)
8      with tf.Session() as sess:
9
10         # Appel de la fonction init pour
11         # initialiser les variables du graphe
12         sess.run(init)
13
14         for i in range(epoch):
15
16             # Création du minibatch
17             batch_X, batch_Y = mnist.train.next_batch(minibatch_size)
18
19             # Création des données pour les placeholder
20             train_data = {x_: batch_X, y_: batch_Y}
21             test_data = {x_: mnist.test.images, y_: mnist.test.labels}
22
23             # Appel des Operations du graphe pour l'apprentissage
24             # L'argument feed_dict permet de donner les valeurs
25             # au placeholder correspondant
26             _, a_train, c_train, = sess.run([train_step,
27                                             accuracy,
28                                             cross_entropy_result],
29                                             feed_dict=train_data)
30
31             # On réalise une évaluation du modèle
32             # toutes les <display_epochs> itérations
33             if i % display_epochs == 0:
34                 a_test, c_test = sess.run([accuracy,

```

```

35                                     cross_entropy_result] ,
36                                     feed_dict=test_data)
37
38         print("Train accuracy: " + str(a_train) +
39               " c_e: " + str(c_train) +
40               " - " + "Test accuracy: " + str(a_test) +
41               " c_e: " + str(c_test))

```

Le réseau est maintenant implémenté et opérationnel. Il est important de noter qu'il est extrêmement simpliste et nécessite de grandes améliorations pour présenter une efficacité convenable sur une problématique complexe.

#### 16.3.5 Exemple de Perceptron Multicouche avec une API haut niveau de Tensorflow: Layers

L'approche bas-niveau de Tensorflow est très souple et puissante mais complexe à assimiler tout en rendant difficile la compréhension du code. Afin de faciliter le développement, une API haut niveau est disponible: **Layers**.

Cette API propose des couches pré-faites (convolution, full-connected, dropout...) capables de comprendre divers paramètres d'optimisation (régulation, initialisation, ...). L'utilisation de ces fonctions permettent de s'émanciper de la construction des variables internes du réseau telles que les poids et les biais. Ainsi, il n'y a plus besoin de les déclarer en amont, leurs créations étant automatiquement faites lors de l'initialisation des couches.

L'implémentation de l'architecture devient ainsi:

Listing 16: Perceptron multicouche: Création de l'architecture du réseau avec Layers

```

1      input_data = tf.reshape(input_data, [-1, 784])
2
3      # Il existe de nombreux autres paramètres disponibles
4      # L'exemple ci-dessous est très simpliste !
5      layer1 = tf.layers.dense(inputs=input_data,
6                                # Variables d'entrees
7                                units=200,
8                                # Nombre de neurones dans la couche
9                                activation=tf.nn.relu,
10                               # Fonction d'activation
11                               name="dense1")
12                               # Nom de la couche
13

```

```

14          # Les variables internes de la couche
15          # sont de la forme dense1/XXX
16
17      layer2 = tf.layers.dense(inputs=layer1,
18                                units=100,
19                                activation=tf.nn.relu,
20                                name="dense2")
21
22      output = tf.layers.dense(inputs=layer2,
23                                units=10,
24                                name="dense1_output")
25      # Cette couche n'a pas de fonction d'activation définie
26      # Par défaut, la fonction d'activation est la fonction linéaire
27      # Ainsi, la sortie est renvoyée sans transformation préalable

```

Les fonctions de *Layers* sont compatibles avec les fonctions bas niveau de Tensorflow. Il est donc ais  de m langer des fonctions issues de diff entes API.

#### 16.3.6 R aliser des graphiques de suivi avec Tensorboard

R aliser des graphiques de suivi d'un r seau peut  tre difficile   r aliser. Afin de simplifier les impl mentations, un outil est int gr    Tensorflow: **Tensorboard**.

Cet outil permet de r aliser la visualisation (en quasi temps-r el) des diverses variables du r seau et de faire un suivi de l'apprentissage en  valuant la consommation de ressources ou les parties du r seau particuli rement gourmande en calculs. Cet outil poss de d'autres fonctionnalit s (affichage d'image par exemple) int ressantes   tudier telles que la repr sentation graphique du r seau par exemple (automatiquement g n r e par *Tensorboard*).

*Tensorboard* n'extract   l'int gralit  des variables cr es par le graphe. Il est n cessaire de lui indiquer quelles sont les variables   consid rer. Pour cela, nous r aliserons des *Summary Operations*. Il y a 3 grandes cat gories   consid rer: *tf.summary.scalar()* pour observer une courbe d'accuracy par exemple, *tf.summary.histogram* pour observer une distribution des activations d'une couche et *tf.summary.(image/text/audio)* pour visualiser un contenu multim dia. Cette derni re est tr s utile pour  valuer visuellement les r sultats du r seau.

*Tensorboard* fonctionne comme un gestionnaire de logs. On d finit les variables   observer, on unit leurs valeurs au sein d'une m me entit  compr hensible par *Tensorboard* puis un log est export  dans un dossier quelconque de la machine-h te. *Tensorboard* cr e un serveur local et ouvre une interface Web qui affichera les r sultats.

La première étape à réaliser est de définir les variables à observer et l'organisation d'affichage de ces variables dans *Tensorboard*. Reprenons l'exemple précédent et supposons que nous voulons suivre l'évolution de la fonction de coût et de l'accuracy durant l'apprentissage uniquement (pas le test). Pour cela, il nous faut donc suivre la variable *cross\_entropy\_result* et *accuracy*. Nous voulons aussi que ces deux variables soient représentées dans deux catégories distinctes: Accuracy et Loss.

Listing 17: Tensorboard: Récupération des variables à observer

```

1      # Les étapes suivantes ne sont exécutées
2      # que lors d'un appel dans une Session
3
4      # On récupère la variable x contenant l'image
5      # Il s'agit de la donnée d'apprentissage
6      tf.summary.image("image", x)
7
8      # On crée une catégorie avec tf.name_scope()
9      with tf.name_scope('Loss'):
10
11         # On récupère la variable cross_entropy_result
12         # On nomme le graph de cette variable, loss
13         tf.summary.scalar("loss", cross_entropy_result)
14
15     with tf.name_scope('Accuracy'):
16         tf.summary.scalar("accuracy", accuracy)
17
18     # Cette fonction collecte l'ensemble des entités
19     # ciblées par une fonction summary.
20     # Il est possible de le faire manuellement si on
21     # veut réaliser des logs qui ne mélangent pas les
22     # données de différentes variables.
23     merged_summary_op = tf.summary.merge_all()
```

La seconde étape consiste à définir les paramètres du fichier de log et à créer ce dernier. Cette étape est réalisé dans une *Session*.

Listing 18: Tensorboard: Création du fichier de log

```

1
2     path_log = "/tmp/summary_train"
3     with tf.Session() as sess:
4         sess.run(init)
```

```

5
6      # Paramétrage du fichier de log
7      # Path du fichier et graph en cours d'observation
8      train_writer = tf.summary.FileWriter(path_log,
9                                         graph=sess.graph)
10
11     for i in range(epoch):
12
13         batch_X, batch_Y = mnist.train.next_batch(minibatch_size)
14
15         train_data = {x: batch_X, y_: batch_Y}
16         test_data = {x: mnist.test.images, y_: mnist.test.labels}
17
18         _, a_train, c_train = sess.run([train_step,
19                                         accuracy,
20                                         cross_entropy_result],
21                                         feed_dict=train_data)
22
23         # On va réaliser une exportation de log
24         # toutes les display_epochs iterations
25         if i % display_epochs == 0:
26
27             # run() appelle aussi la fonction merged_summary_op
28             # Les données sont préparées pour l'exportation
29             a_train, c_train, summary = sess.run([accuracy,
30                                         cross_entropy_result,
31                                         merged_summary_op],
32                                         feed_dict=train_data)
33
34             # Exportation vers le fichier de log
            train_writer.add_summary(summary, i)

```

*Tensorboard* crée un serveur local accessible via le port 6006 par défaut. Pour lancer le serveur, on exécute la commande shell:

Listing 19: Tensorboard: Lancement du serveur local

```

1      # L'argument logdir doit être modifié
2      # selon le path choisi pour les logs
3      tensorboard --logdir=/tmp/summary_

```

L'interface obtenue par cet exemple est visible sur la Figure 128<sup>217</sup>. Il existe dif-

---

<sup>217</sup>L'aspect très oscillant de la courbe d'apprentissage est dû à la présence de DropOut dans

férentes options associées à l'interface pour modifier l'affichage ou personnaliser les paramètres de représentation des données. On peut observer que les courbes sont séparées en deux parties Accuracy et Loss (utilisation de *name\_scope()*). Une catégorie par défaut a été créée pour les images.

Au-delà de représenter l'évolution d'une variable, il peut être utile de représenter plusieurs courbes sur un même repère pour réaliser des comparaisons. C'est utile pour évaluer l'évolution de l'accuracy sur jeu d'apprentissage et de test par exemple. Pour cela, il est nécessaire de créer un nouveau fichier de log qui récupérera les mêmes entités que le fichier de log précédemment créé mais avec d'autres données utilisées par le réseau (dans notre cas, les données de test et d'apprentissage). La modification du code est donc:

Listing 20: Tensorboard: Création du fichier de log

```

1      path_log_test = "/tmp/summary_/train"
2      path_log_train = "/tmp/summary_/test"
3
4      with tf.Session() as sess:
5          sess.run(init)
6
7      train_writer = tf.summary.FileWriter(path_log_train,
8                                         graph=sess.graph)
9
10     # Cr ation d'un nouveau fichier de log
11     test_writer = tf.summary.FileWriter(path_log_test,
12                                       graph=sess.graph)
13
14     for i in range(epoch):
15
16         batch_X, batch_Y = mnist.train.next_batch(minibatch_size)
17
18         train_data = {x: batch_X, y_: batch_Y}
19         test_data = {x: mnist.test.images, y_: mnist.test.labels}
20
21         _, a_train, c_train = sess.run([train_step,
22                                         accuracy,
23                                         cross_entropy_result],
24                                         feed_dict=train_data)
25
26
27         if i % display_epochs == 0:
28
29             a_train, c_train, summary = sess.run([accuracy,

```

---

le réseau utilisé pour réaliser la courbe

```

30                                         cross_entropy_result ,
31                                         merged_summary_op] ,
32                                         feed_dict=train_data)
33
34             train_writer.add_summary(summary , i)
35
36             # Récupération des valeurs du reseau
37             a_test , c_test , summary = sess.run([accuracy ,
38                                         cross_entropy_result ,
39                                         merged_summary_op] ,
40                                         feed_dict=test_data)
41
42             # Copie dans le fichier de log
43             test_writer.add_summary(summary , i)

```

Cette modification permet l'affichage obtenu sur la Figure 129. On observe que les deux courbes sont cumulées sur un même graphique au sein du même groupe. Dans cet exemple, les indices des valeurs coïncident, i.e la valeur  $i$  de *test* coïncide avec la valeur  $i$  de *train*). Il est possible que les deux jeux de données ne possèdent pas le même nombre de valeur. Cette particularité n'est pas couvert par ce rapport. Il peut être intéressant de développer ce type de problématique.

#### 16.3.7 Intéraction entre couches et isolation de variable

Tensorflow propose une manière simple d'interagir avec les Tensors. Il est donc facile de réaliser des transformations arithmétiques telles que des sommes, concaténation, multiplication...

Pour illustrer cette caractéristiques, nous allons implémenter une version basique d'un block Resnet. Ce block est graphiquement représenté sur la Figure 130.

Listing 21: Implémentation basique d'un block resnet

```

1
2     def vanilla_block_resnet(input_ , output_depth):
3
4         # Création d'une couche de convolution
5         # l'argument "same" permet de réaliser
6         # un zéro-padding afin de ne pas diminuer
7         # la dimension de la matrice
8         conv1_ = tf.layers.conv2d(inputs=input_ ,
9                               filters=output_depth ,
10                             kernel_size=3,

```

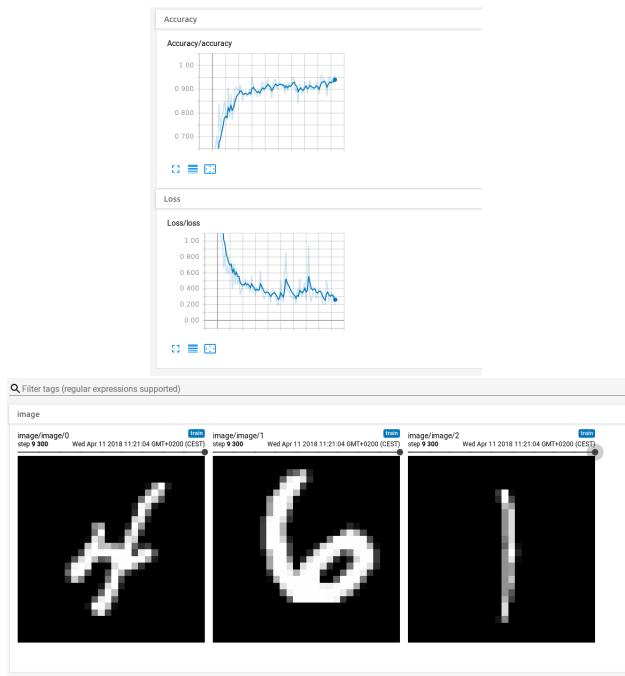


Figure 128: Exemple d'interface basique de Tensorboard

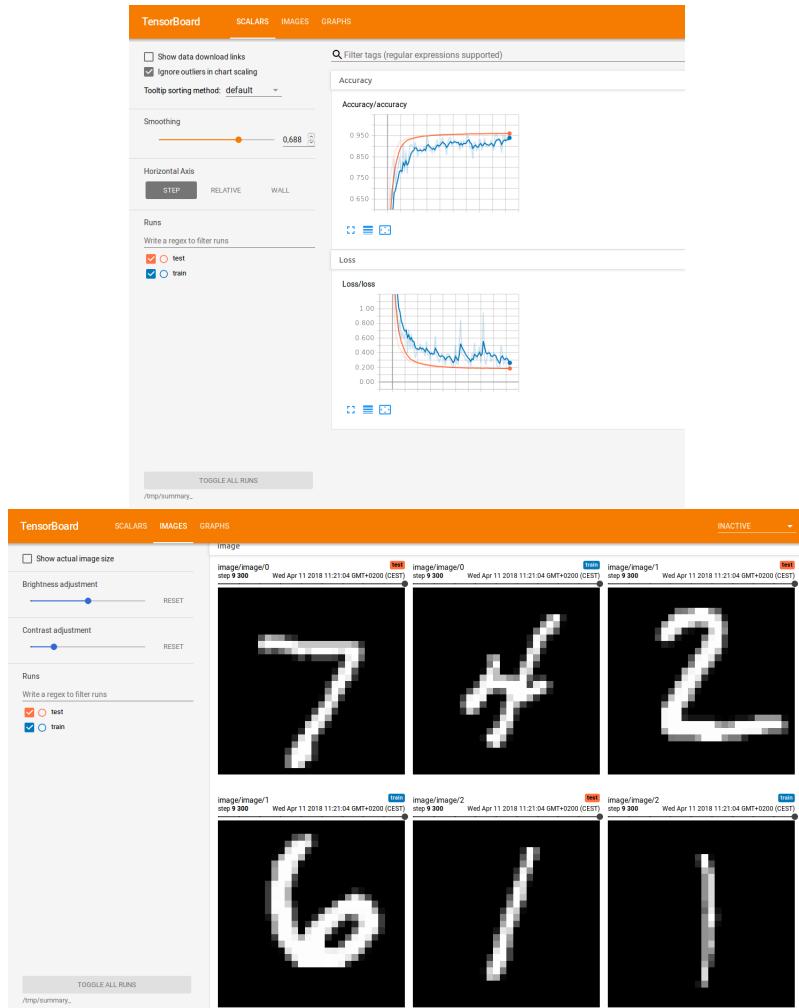


Figure 129: Exemple d'interface avancée de Tensorboard

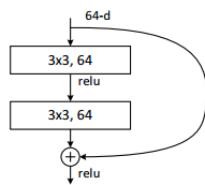


Figure 130: Block Resnet d'illustration

```

11                         padding="same",
12                         name="conv1_")
13
14     conv1= tf.nn.relu(conv1)
15
16     conv2 = tf.layers.conv2d(inputs=conv1,
17                             filters=output_depth,
18                             kernel_size=3,
19                             padding="same",
20                             name="conv2")
21
22     # Les deux Tensors sont additionnés
23     # L'opérateur "+" marche aussi
24     output_block_ = tf.add(conv2, input_)
25     output_block = tf.nn.relu(output_block_)
26
27     return output_block

```

De même, il est utile de pouvoir délimiter les Tensors soumis à la rétropropagation du gradient. Par défaut, tous les Tensors concernés par l'apprentissage sont mis à jour mais pour des réseaux spécifiques, ceci pose problème. C'est le cas des GANs qui demandent un entraînement indépendant de deux sous-réseaux. La méthode consiste à isoler les poids à mettre à jour - pour cela, l'utilisation de la fonction `variable_scope()` est utile - puis à spécifier les variables à considérer à l'optimizer.

Listing 22: Choix des poids à mettre à jour lors d'une rétropropagation du gradient

```

1
2     # Récupération de l'ensemble des
3     # variables entraînables du graphe
4     tvar = tf.trainable_variables()
5
6     # Récupération des Tensors du groupe
7     # "discriminator" uniquement
8     dvar = [var for var in tvar if 'discriminator' in var.name]
9
10    # L'argument var_list permet de définir
11    # le groupe de Tensor à considérer
12    # Tout Tensor non présent dans
13    # cette liste sera ignorée
14
15    # On suppose <fn>, valeur

```

```

16      # représentant la sortie de
17      # la fonction de perte
18      tf.train.AdamOptimizer(0.0002).minimize(<fn>, var_list=dvar)

```

### 16.3.8 Sauvegarder et restaurer un modèle

Durant le développement d'un réseau, il est important de considérer l'exportation du modèle afin qu'il puisse être réutilisé sans réaliser de nouveau l'apprentissage. De même, la restauration d'un modèle est capitale dans le cadre du *Transfer Learning*. Cette partie est donc une étape importante dans la phase de développement d'un réseau neuronal.

#### 16.3.8.1 Structure de données

Un modèle exporté se divise en 2 parties:

- **Meta graph:** ce fichier est un *protocol buffer*<sup>218</sup>. Ce fichier contient l'architecture du modèle: Operations, Variables etc... Mais ne **contient pas** les valeurs des variables ! Ce fichier peut être considéré comme le *squelette* du réseau sauvegardé.
- **Fichier « checkpoint »:** Il est composé de 3 fichiers<sup>219</sup>. Le premier est un .data et contient l'intégralité des valeurs numériques sauvegardées. Le second est un .index qui identifie un *checkpoint* lors d'une exportation et le dernier est un fichier nommé *checkpoint* qui référence une liste de plusieurs *checkpoint* récents.

#### 16.3.8.2 Sauvegarder un modèle

Afin de sauvegarder un modèle, on exploitera la classe *Saver* de tensorflow. Cette classe est utilisable dans une Session. Il est **important** de noter que les valeurs des *placeholder* ne sont pas sauvegardées !

Listing 23: Sauvegarder un modèle

```

1
2      # Initialisation d'un objet Saver
3      saver = tf.train.Saver()
4
5      with tf.Session() as sess:

```

---

<sup>218</sup>Format de sérialisation développé par Google pour stocker des structures de manière optimisée

<sup>219</sup>Pour les versions antérieures à 0.11, la structure est différente. Nous n'approfondirons pas l'ancienne configuration

```

6
7     sess.run(tf.global_variables_initializer())
8
9     # Création du modèle sous le nom
10    # "my_model" (chemin courant)
11    saver.save(sess, 'my_model')

```

Il est possible de réaliser une exportation sous condition. Il est donc possible de choisir les variables à exporter, et les conditions d'exportation (notamment temporelles).

Listing 24: Sauvegarder un modèle sous condition

```

1
2     # Sauvegarder le modèle après 5000 itérations
3     saver = tf.train.Saver()
4     saver.save(sess, 'my_model', global_step = 5000)
5
6     # Sauvegarder le modèle après 10000 itérations
7     # sans création du fichier .meta
8
9     # Ce fichier n'est pas utile lorsqu'une
10    # sauvegarde a déjà été faite durant l'exécution
11    # L'architecture du modèle ne varie pas au fil du temps
12    saver = tf.train.Saver()
13    saver.save(sess, 'my_model', global_step=10000,
14                           write_meta_graph=False)
15
16    # Il est possible d'initialiser l'objet Saver
17    # avec des paramètres de sauvegarde
18
19    # Cette configuration permettra de considérer
20    # que les 4 derniers checkpoint et d'en
21    # réaliser un toutes les 2 heures
22    saver = tf.train.Saver(max_to_keep=4,
23                           keep_checkpoint_every_n_hours=2)
24    saver.save(...)
25
26    # Cette configuration permet de ne sauvegarder
27    # que la variable v1 et v2 du graphe
28    saver = tf.train.Saver([v1, v2])
29    saver.save(...)
30
31    # Idem mais en choisissant le nom

```

```

32      # de la variable ciblée
33      saver = tf.train.Saver({ "v1": v1,
34                                "v2": v2})
35      saver.save(...)
```

### 16.3.8.3 Importation d'un modèle

L'importation d'un modèle se fait en deux étapes: la restauration de l'architecture et la restauration des valeurs. La restauration se fait à travers une Session.

Listing 25: Importer un modèle

```

1   with tf.Session() as sess:
2
3       # Importation du graphe
4       # Il remplace le graphe courant
5       new_saver = tf.train.import_meta_graph('my_model.meta')
6
7       # Importation des valeurs issues du dernier checkpoint
8       new_saver.restore(sess, tf.train.latest_checkpoint('./'))
```

Il est possible d'interagir avec le modèle via le nom des variables et opérations<sup>220</sup>.

Listing 26: Importer un modèle

```

1   # On suppose un modèle importé contenant
2   # une variable w1, une opération
3   # "adding_value" et une couche fc1
4
5   # Accéder à la variable w1
6   # w1 = tf.placeholder("float", name="w1")
7   w1 = graph.get_tensor_by_name("w1:0")
8
9   # Accéder à l'opération "adding_value"
10  # op1 = tf.multiply(w1,w2, name="adding_value")
11  adding_value = graph.get_tensor_by_name("adding_value")
12
13  # Accéder à la sortie d'une couche
14  # fc = Dense(..., name="fc1")
```

---

<sup>220</sup>L'importance des noms de variables internes au graphe se situe précisément ici. En effet, les noms issus de l'environnement de Python ont "disparu" lors de l'exportation du modèle !

```
15      fc= graph.get_tensor_by_name('fc1:0')
```

## 17 Ajouts à venir

- 17.1 Application à la reconnaissance vocale
- 17.2 Application au traitement du langage écrit
  - 17.2.1 Traduction - Neural Machine Translation
  - 17.2.2 Traduction - Attention is all your need
  - 17.2.3 Recherche d'informations
  - 17.2.4 Désambiguïsation d'entités
- 17.3 L'analyse d'image et ses formes
  - 17.3.1 Segmentation: méthodes State-of-the-art
  - 17.3.2 Object Tracking: méthodes State-of-the-art
- 17.4 Machine Learning et Éthique
- 17.5 Deep Learning Bayésien
- 17.6 Tutoriel applicatif de Keras

## 18 Appendix

### 18.1 Object Detection API de Google/Tensorflow

**Attention:** Cette API est une API en cours de développement. Elle est susceptible d'être modifiée. De ce fait, le guide qui va suivre pourra être (partiellement) obsolète au fil du temps, notamment la partie *Installation*. Nous supposerons que vous disposez d'un système sous Ubuntu 16.04 avec Python3 d'opérationnel.

L'API *Object Detection*[36] de Tensorflow-Google est une API en cours de développement qui propose une architecture simple et évolutive pour exploiter des modèles d'*Object Detection*. Cette API a pour en ambition de faciliter l'accès aux modèles récents dont l'implémentation est difficile et peu détaillée dans les articles de recherches et de devenir une référence industrielle pour des projets d'analyse d'images.

### 18.1.1 Installation

L'installation de *Object-Detection* est aisée et parfaitement expliquée sur le lien suivant: [https://github.com/tensorflow/models/blob/master/research/object\\_detection/g3doc/installation.md](https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/installation.md).

Il est **important** de ne pas oublier la mise à jour de PYTHONPATH ! Pour rendre la modification permanente, modifier le fichier .bashrc associé à votre Shell.

### 18.1.2 Difficultés possibles: Version de Protobuf incorrecte

Le tutoriel d'installation de l'API installe protobuf 2.6 alors que l'API, dans ses dernières mises à jour<sup>221</sup>, exploite la version 3. Il est nécessaire d'installer manuellement une version 3+ de protobuf.

**Attention:** Assurez-vous de l'absence de la version 2.6 sur votre machine ou un risque de conflit existe !

Listing 27: Installation de protobuf3

```
1 # Téléchargement du paquet
2 # Attention à la version du paquet !
3 # Modifier la commande en adéquation avec la version souhaitée
4 curl -OL https://github.com/google/protobuf/releases/download/v3.2.0/
5 protoc-3.5.1-linux-x86_64.zip
6
7 unzip protoc-3.5.1-linux-x86_64.zip -d protoc3
8
9 # Installation manuelle du paquet dans les répertoires
10 sudo mv protoc3/bin/* /usr/local/bin/
11 sudo mv protoc3/include/* /usr/local/include/
12
13 # Création d'un lien pour /usr/bin nécessaire à l'API Tensorflow
14 sudo ln -s /usr/local/bin/protoc /usr/bin/protoc
```

### 18.1.3 Labellisation d'image

Afin de pouvoir entraîner un modèle d'*Object Detection*, il est nécessaire d'avoir un jeu d'apprentissage (et de test). De nombreux jeux de données sont disponibles en gratuitement mais selon la spécificité des entités à détecter, il peut être nécessaire de créer soi-même un jeu de données.

**Attention:** Créer un jeu de données d'apprentissage est une tâche longue, fastidieuse et difficile. En effet, il est nécessaire de labelliser une quantité importante de données (potentiellement plusieurs millions...) et de s'assurer qu'elles sont

---

<sup>221</sup>attention à la date de ce guide !

représentatives du phénomènes qu'on cherche à observer. Par exemple, si nous voulons détecter des chats, il n'est pas pertinent d'utiliser des images ne contenant que des chats noirs par exemple.

Pour labelliser nos données, il existe des outils facilitant le travail notamment *LabelImg*[95]. Le code source est disponible sur GitHub(<https://github.com/tzutalin/labelImg>). Cet outil offre une interface graphique simple et épurée pour labelliser les données et réalise une conversion automatique de la labellisation au format PASCAL VOC (xml) qui est un format de référence pour ce type de jeu de données. Ce format est apprécié car *Tensorflow* propose un script de conversion automatisée de ce format vers le format TFRecord utilisé par l'API.

Supposons un cas simple de labellisation de 3 images regroupées dans un dossier *image\_test*. La Figure 131 résume les différentes étapes de la labellisation avec ce logiciel.

La labellisation est résumé dans un fichier XML au format PASCAL VOC. Elle possède une structure facilement compréhensible et permet un traitement ultérieur si nécessaire. A terme, le dossier *image\_test* contiendra 6 fichiers (3 images, 3 fichiers XML).

Listing 28: Exemple de labellisation au format PASCAL VOC d'une image contenant uniquement un alpaga

```

1 <annotation>
2   # Dossier source
3   <folder>image_test</folder>
4   # Nom de l'image
5   <filename>lama1.jpeg</filename>
6   # Chemin absolu de l'image
7   <path>/home/boyer/Bureau/image_test/lama1.jpeg</path>
8   <source>
9     <database>Unknown</database>
10    </source>
11    # Dimension de l'image originale
12    <size>
13      <width>275</width>
14      <height>183</height>
15      <depth>3</depth>
16    </size>
17    <segmented>0</segmented>
18    # Spécificités du label
19    # Classe, dimension de la bounding box, difficulté de l'image...
20    <object>
21      <name>alpaga</name>
22      <pose>Unspecified</pose>
23      <truncated>0</truncated>
```

```
24          <difficult>0</difficult>
25          <bndbox>
26              <xmin>56</xmin>
27              <ymin>28</ymin>
28              <xmax>218</xmax>
29              <ymax>181</ymax>
30          </bndbox>
31      </ object>
32  </annotation>
```

## 18.2 Présentation de l'API

En cours

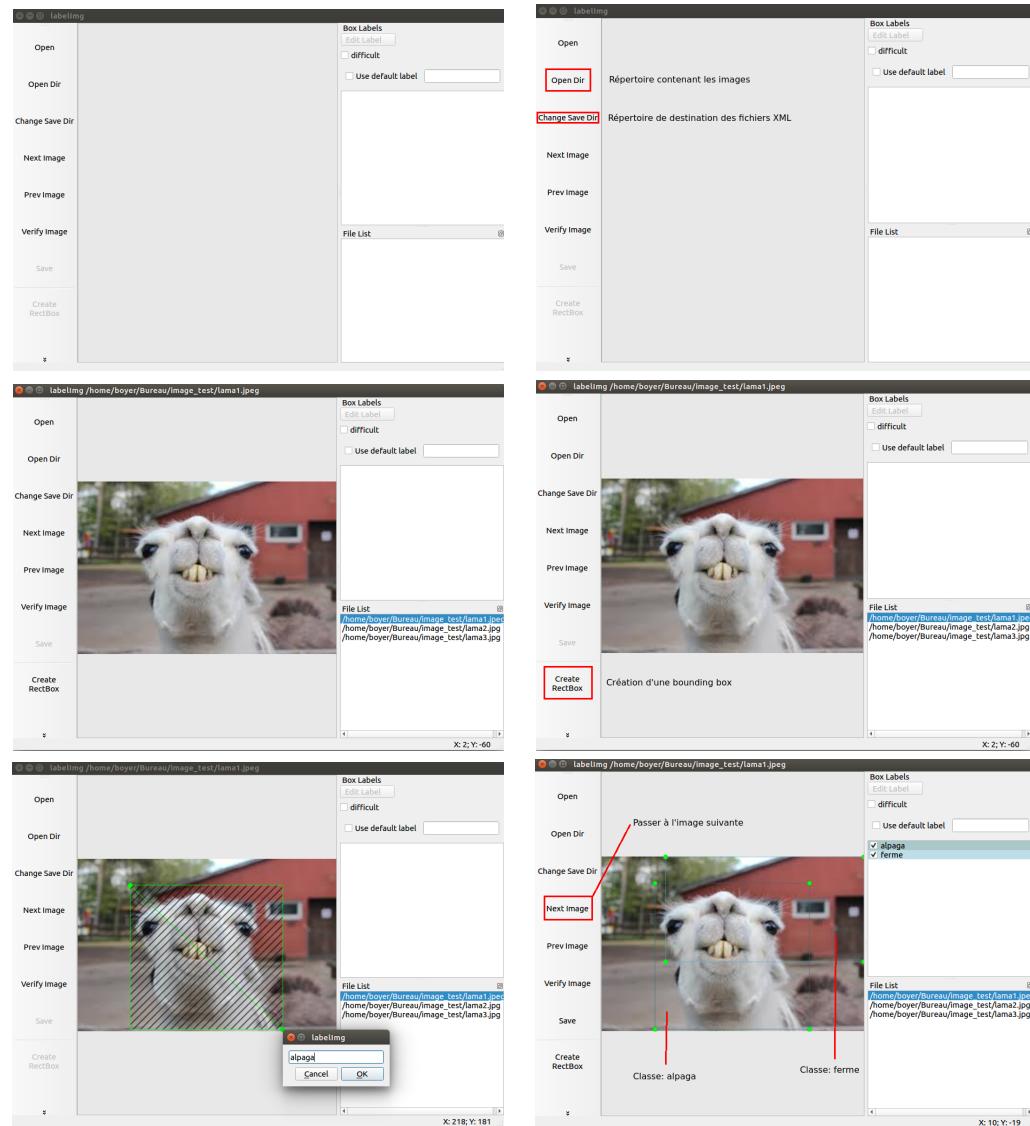


Figure 131: Utilisation de LabelImg

## References

- [1] Hierarchical variational autoencoders for music. 2017.
- [2] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek Gordon Murray, Benoit Steiner, Paul A. Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zhang. Tensorflow: A system for large-scale machine learning. *CoRR*, abs/1605.08695, 2016.
- [3] M. Arjovsky, S. Chintala, and L. Bottou. Wasserstein GAN. *ArXiv e-prints*, January 2017.
- [4] Quoc V. Le Ilya Sutskever Lukasz Kaiser Karol Kurach James Martens Arvind Neelakantan, Luke Vilnis. Adding gradient noise improves learning for very deep networks. *arXiv*, novembre 2015.
- [5] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *CoRR*, abs/1409.0473, 2014.
- [6] Dana H. Ballard. Modular learning in neural networks. pages 279–284, 1987.
- [7] Mateusz Buda, Atsuto Maki, and Maciej A. Mazurowski. A systematic study of the class imbalance problem in convolutional neural networks. *CoRR*, abs/1710.05381, 2017.
- [8] Alfredo Canziani, Adam Paszke, and Eugenio Culurciello. An analysis of deep neural network models for practical applications. *CoRR*, abs/1605.07678, 2016.
- [9] G. Chen, D. Ye, Z. Xing, J. Chen, and E. Cambria. Ensemble application of convolutional and recurrent neural networks for multi-label text categorization. pages 2377–2383, May 2017.
- [10] KyungHyun Cho, Bart van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *CoRR*, abs/1409.1259, 2014.
- [11] François Chollet. Xception: Deep learning with depthwise separable convolutions. *CoRR*, abs/1610.02357, 2016.
- [12] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *CoRR*, abs/1511.07289, 2015.

- [13] Alexis Conneau, Holger Schwenk, Loïc Barrault, and Yann LeCun. Very deep convolutional networks for natural language processing. *CoRR*, abs/1606.01781, 2016.
- [14] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.
- [15] Carl Doersch. Tutorial on variational autoencoders. *CoRR*, abs/1606.05908, 2016.
- [16] Mark Everingham, S. M. Eslami, Luc Gool, Christopher K. Williams, John Winn, and Andrew Zisserman. The pascal visual object classes challenge: A retrospective. *Int. J. Comput. Vision*, 111(1):98–136, January 2015.
- [17] Pedro F. Felzenszwalb and Daniel P. Huttenlocher. Efficient graph-based image segmentation. *International Journal of Computer Vision*, 59:2004, 2004.
- [18] I. Freeman, L. Roesel-Koerner, and A. Kummert. EffNet: An Efficient Structure for Convolutional Neural Networks. *ArXiv e-prints*, January 2018.
- [19] Cheng-Yang Fu, Wei Liu, Ananth Ranga, Ambrish Tyagi, and Alexander C. Berg. DSSD : Deconvolutional single shot detector. *CoRR*, abs/1701.06659, 2017.
- [20] Ross B. Girshick. Fast R-CNN. *CoRR*, abs/1504.08083, 2015.
- [21] Ross B. Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. *CoRR*, abs/1311.2524, 2013.
- [22] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. pages 2672–2680, 2014.
- [23] Jiuxiang Gu, Zhenhua Wang, Jason Kuen, Lianyang Ma, Amir Shahroudy, Bing Shuai, Ting Liu, Xingxing Wang, and Gang Wang. Recent advances in convolutional neural networks. *CoRR*, abs/1512.07108, 2015.
- [24] Ishaan Gulrajani, Kundan Kumar, Faruk Ahmed, Adrien Ali Taiga, Francesco Visin, David Vázquez, and Aaron C. Courville. Pixelvae: A latent variable model for natural images. *CoRR*, abs/1611.05013, 2016.
- [25] Raia Hadsell, Sumit Chopra, and Yann Lecun. Dimensionality reduction by learning an invariant mapping. 2006.
- [26] Song Han, Huizi Mao, and William J. Dally. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. *CoRR*, abs/1510.00149, 2015.

- [27] Song Han, Jeff Pool, Sharan Narang, Huizi Mao, Shijian Tang, Erich Elsen, Bryan Catanzaro, John Tran, and William Dally. Dsd: Regularizing deep neural networks with dense-sparse-dense training flow. 07 2016.
- [28] Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, AAAI'16, pages 2094–2100. AAAI Press, 2016.
- [29] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. *CoRR*, abs/1406.4729, 2014.
- [30] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.
- [31] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobiilenets: Efficient convolutional neural networks for mobile vision applications. *CoRR*, abs/1704.04861, 2017.
- [32] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. *CoRR*, abs/1709.01507, 2017.
- [33] Zhiting Hu, Zichao Yang, Xiaodan Liang, Ruslan Salakhutdinov, and Eric P. Xing. Controllable text generation. *CoRR*, abs/1703.00955, 2017.
- [34] Gao Huang, Zhuang Liu, and Kilian Q. Weinberger. Densely connected convolutional networks. *CoRR*, abs/1608.06993, 2016.
- [35] Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Q. Weinberger. Deep networks with stochastic depth. *CoRR*, abs/1603.09382, 2016.
- [36] Jonathan Huang, Vivek Rathod, Chen Sun, Menglong Zhu, Anoop Korattikara, Alireza Fathi, Ian Fischer, Zbigniew Wojna, Yang Song, Sergio Guadarrama, and Kevin Murphy. Speed/accuracy trade-offs for modern convolutional object detectors. *CoRR*, abs/1611.10012, 2016.
- [37] Aapo Hyvärinen and Urs Köster. Complex cell pooling and the statistics of natural images. *Network: Computation in Neural Systems*, 18(2):81–100, 2007. PMID: 17852755.
- [38] Forrest N. Iandola, Matthew W. Moskewicz, Khalid Ashraf, Song Han, William J. Dally, and Kurt Keutzer. SqueezeNet: Alexnet-level accuracy with 50x fewer parameters and <1mb model size. *CoRR*, abs/1602.07360, 2016.
- [39] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015.

- [40] Max Jaderberg, Karen Simonyan, Andrew Zisserman, and Koray Kavukcuoglu. Spatial transformer networks. *CoRR*, abs/1506.02025, 2015.
- [41] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross B. Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. *CoRR*, abs/1408.5093, 2014.
- [42] Shaoqing Ren Jian Sun Kaiming He, Xiangyu Zhang. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *arXiv*, février 2015.
- [43] Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. A convolutional neural network for modelling sentences. *CoRR*, abs/1404.2188, 2014.
- [44] Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. A convolutional neural network for modelling sentences. *CoRR*, abs/1404.2188, 2014.
- [45] Tapas Kanungo, David M. Mount, Nathan S. Netanyahu, Christine D. Piatko, Ruth Silverman, Angela Y. Wu, Senior Member, and Senior Member. An efficient k-means clustering algorithm: Analysis and implementation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24:881–892, 2002.
- [46] Wojciech Marian Czarnecki Katarzyna Janocha. On loss functions for deep neural networks in classification. *arXiv*, février 2017.
- [47] Yoon Kim. Convolutional neural networks for sentence classification. *CoRR*, abs/1408.5882, 2014.
- [48] Gary King and Langche Zeng. Logistic regression in rare events data. *Political Analysis*, 9:137–163, 2001.
- [49] D. P Kingma and M. Welling. Auto-Encoding Variational Bayes. *ArXiv e-prints*, December 2013.
- [50] Günter Klambauer, Thomas Unterthiner, Andreas Mayr, and Sepp Hochreiter. Self-normalizing neural networks. *CoRR*, abs/1706.02515, 2017.
- [51] G. R. Koch. Siamese neural networks for one-shot image recognition. 2015.
- [52] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. Cifar-10 (canadian institute for advanced research).
- [53] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. Cifar-100 (canadian institute for advanced research).

- [54] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. pages 1097–1105, 2012.
- [55] Gustav Larsson, Michael Maire, and Gregory Shakhnarovich. Fractalnet: Ultra-deep neural networks without residuals. *CoRR*, abs/1605.07648, 2016.
- [56] Hoa T. Le, Christophe Cerisara, and Alexandre Denis. Do convolutional networks need to be deep for text classification ? *CoRR*, abs/1707.04108, 2017.
- [57] Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010.
- [58] Zeming Li, Chao Peng, Gang Yu, Xiangyu Zhang, Yangdong Deng, and Jian Sun. Detnet: A backbone network for object detection. *CoRR*, abs/1804.06215, 2018.
- [59] Depeng Liang and Yongdong Zhang. AC-BLSTM: asymmetric convolutional bidirectional LSTM networks for text classification. *CoRR*, abs/1611.01884, 2016.
- [60] Min Lin, Qiang Chen, and Shuicheng Yan. Network in network. *CoRR*, abs/1312.4400, 2013.
- [61] Tsung-Yi Lin, Piotr Dollár, Ross B. Girshick, Kaiming He, Bharath Hariharan, and Serge J. Belongie. Feature pyramid networks for object detection. *CoRR*, abs/1612.03144, 2016.
- [62] Tsung-Yi Lin, Priya Goyal, Ross B. Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. *CoRR*, abs/1708.02002, 2017.
- [63] Tsung-Yi Lin, Michael Maire, Serge J. Belongie, Lubomir D. Bourdev, Ross B. Girshick, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft COCO: common objects in context. *CoRR*, abs/1405.0312, 2014.
- [64] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott E. Reed, Cheng-Yang Fu, and Alexander C. Berg. SSD: single shot multibox detector. *CoRR*, abs/1512.02325, 2015.
- [65] Minh-Thang Luong, Hieu Pham, and Christopher D. Manning. Effective approaches to attention-based neural machine translation. *CoRR*, abs/1508.04025, 2015.
- [66] Alireza Makhzani and Brendan J. Frey. k-sparse autoencoders. *CoRR*, abs/1312.5663, 2013.

- [67] Matt Mazur. A step by step backpropagation example. <https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/>.
- [68] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning, 2013. cite arxiv:1312.5602Comment: NIPS Deep Learning Workshop 2013.
- [69] Andrew Ng. Sparse autoencoder, cs294a lecture notes.
- [70] Hamid Palangi, Li Deng, Yelong Shen, Jianfeng Gao, Xiaodong He, Jian-shu Chen, Xinying Song, and Rabab K. Ward. Deep sentence embedding using the long short term memory network: Analysis and application to information retrieval. *CoRR*, abs/1502.06922, 2015.
- [71] Lutz Prechelt. Early stopping — but when? pages 53–67, 2012.
- [72] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *CoRR*, abs/1511.06434, 2015.
- [73] Yoshua Bengio Razvan Pascanu, Tomas Mikolov. On the difficulty of training recurrent neural networks. *arXiv*, février 2013.
- [74] J. Redmon and A. Farhadi. YOLOv3: An Incremental Improvement. *ArXiv e-prints*, April 2018.
- [75] Joseph Redmon, Santosh Kumar Divvala, Ross B. Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. *CoRR*, abs/1506.02640, 2015.
- [76] Joseph Redmon and Ali Farhadi. YOLO9000: better, faster, stronger. *CoRR*, abs/1612.08242, 2016.
- [77] Mengye Ren, Wenyuan Zeng, Bin Yang, and Raquel Urtasun. Learning to reweight examples for robust deep learning. *CoRR*, abs/1803.09050, 2018.
- [78] Shaoqing Ren, Kaiming He, Ross B. Girshick, and Jian Sun. Faster R-CNN: towards real-time object detection with region proposal networks. *CoRR*, abs/1506.01497, 2015.
- [79] Salah Rifai, Pascal Vincent, Xavier Muller, Xavier Glorot, and Yoshua Bengio. Contracting auto-encoders: Explicit invariance during feature extraction. 2011.
- [80] Abhijit Guha Roy, Nassir Navab, and Christian Wachinger. Concurrent spatial and channel squeeze & excitation in fully convolutional networks. *CoRR*, abs/1803.02579, 2018.

- [81] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *CoRR*, abs/1511.05952, 2015.
- [82] Florian Schroff, Dmitry Kalenichenko, and James Philbin. Facenet: A unified embedding for face recognition and clustering. *CoRR*, abs/1503.03832, 2015.
- [83] Stanislau Semeniuta, Aliaksei Severyn, and Erhardt Barth. A hybrid convolutional variational autoencoder for text generation. *CoRR*, abs/1702.02390, 2017.
- [84] Wenling Shang, Kihyuk Sohn, Diogo Almeida, and Honglak Lee. Understanding and improving convolutional neural networks via concatenated rectified linear units. *CoRR*, abs/1603.05201, 2016.
- [85] Evan Shelhamer, Jonathan Long, and Trevor Darrell. Fully convolutional networks for semantic segmentation. *CoRR*, abs/1605.06211, 2016.
- [86] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.
- [87] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. 15:1929–1958, 06 2014.
- [88] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. *CoRR*, abs/1409.3215, 2014.
- [89] R. Sutton and A. Barto. *Reinforcement learning: An introduction*. Cambridge, MA, MIT Press, 1998.
- [90] Richard S. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3(1):9–44, Aug 1988.
- [91] Christian Szegedy, Sergey Ioffe, and Vincent Vanhoucke. Inception-v4, inception-resnet and the impact of residual connections on learning. *CoRR*, abs/1602.07261, 2016.
- [92] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *CoRR*, abs/1409.4842, 2014.
- [93] Christian Szegedy, Scott E. Reed, Dumitru Erhan, and Dragomir Anguelov. Scalable, high-quality object detection. *CoRR*, abs/1412.1441, 2014.
- [94] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. *CoRR*, abs/1512.00567, 2015.
- [95] Tzutalin. Labelimg. *Git code*, 2015.

- [96] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017.
- [97] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. pages 1096–1103, 2008.
- [98] Li Wan, Matthew D Zeiler, Sixn Zhang, Yann Lecun, and Rob Fergus. Regularization of neural networks using dropconnect. 01 2013.
- [99] Xingyou Wang, Weijie Jiang, and Zhiyong Luo. Combination of convolutional and recurrent neural network for sentiment analysis of short texts. 2016.
- [100] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Hasselt, Marc Lanctot, and Nando Freitas. Dueling network architectures for deep reinforcement learning. In Maria Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 1995–2003, New York, New York, USA, 20–22 Jun 2016. PMLR.
- [101] Christopher J. C. H. Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8(3):279–292, May 1992.
- [102] Alexander Wong, Mohammad Javad Shafiee, Francis Li, and Brendan Chwyl. Tiny SSD: A tiny single-shot detection deep convolutional neural network for real-time embedded object detection. *CoRR*, abs/1802.06488, 2018.
- [103] Saining Xie, Ross B. Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. *CoRR*, abs/1611.05431, 2016.
- [104] Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. Empirical evaluation of rectified activations in convolutional network. *CoRR*, abs/1505.00853, 2015.
- [105] Zichao Yang, Diyi Yang, Chris Dyer, Xiaodong He, Alexander J. Smola, and Eduard H. Hovy. Hierarchical attention networks for document classification. 2016.
- [106] Wenpeng Yin and Hinrich Schütze. Multichannel variable-size convolution for sentence classification. *CoRR*, abs/1603.04513, 2016.
- [107] D. Yogatama, C. Dyer, W. Ling, and P. Blunsom. Generative and Discriminative Text Classification with Recurrent Neural Networks. *ArXiv e-prints*, March 2017.

- [108] Dingjun Yu, Hanli Wang, Peiqiu Chen, and Zhihua Wei. Mixed pooling for convolutional neural networks. 2014.
- [109] Fisher Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions. *CoRR*, abs/1511.07122, 2015.
- [110] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. *CoRR*, abs/1605.07146, 2016.
- [111] Matthew D. Zeiler and Rob Fergus. Stochastic pooling for regularization of deep convolutional neural networks. *CoRR*, abs/1301.3557, 2013.
- [112] Matthew D. Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. *CoRR*, abs/1311.2901, 2013.
- [113] Ke Zhang, Miao Sun, Tony X. Han, Xingfang Yuan, Liru Guo, and Tao Liu. Residual networks of residual networks: Multilevel residual networks. *CoRR*, abs/1608.02908, 2016.
- [114] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. *CoRR*, abs/1707.01083, 2017.
- [115] Ye Zhang, Stephen Roller, and Byron C. Wallace. MGNC-CNN: A simple approach to exploiting multiple word embeddings for sentence classification. *CoRR*, abs/1603.00968, 2016.
- [116] Ye Zhang and Byron C. Wallace. A sensitivity analysis of (and practitioners' guide to) convolutional neural networks for sentence classification. *CoRR*, abs/1510.03820, 2015.
- [117] Chunting Zhou, Chonglin Sun, Zhiyuan Liu, and Francis C. M. Lau. A C-LSTM neural network for text classification. *CoRR*, abs/1511.08630, 2015.
- [118] L. Zhu, R. Deng, Z. Deng, G. Mori, and P. Tan. Sparsely Connected Convolutional Networks. *ArXiv e-prints*, January 2018.