Deep Learning par la pratique

Jour 2, matin
Images et ConvNets
transfer learning

Recap

On a vu

- Tenseurs
- Keras & Tensorflow
- Fonction d'activation
- Backprop

Plan

- Echauffement avec les callbacks dans Keras
- CNN : convolutional neural networks, image classification
- TensorBoard : pour monitorer l'entraînement et la performance du modèle
- transfer learning : construire sur la base d'un méga modèle

Callbacks

Callbacks

On veut pouvoir exécuter une fonction à chaque itération : definir un callback

- EarlyStopping : arrêter l'entraînement si le gain de performance stagne
- ModelCheckpoint : sauver le modèle régulièrement pour avoir une copie en cas de problème
- TensorBoard : écrire les logs adaptés à TensorBoard
- Gérer le learning rate : décroitre le paramètre quand le gain de performance stagne

Callbacks

Les callbacks prédéfini de Keras : https://keras.io/api/callbacks/

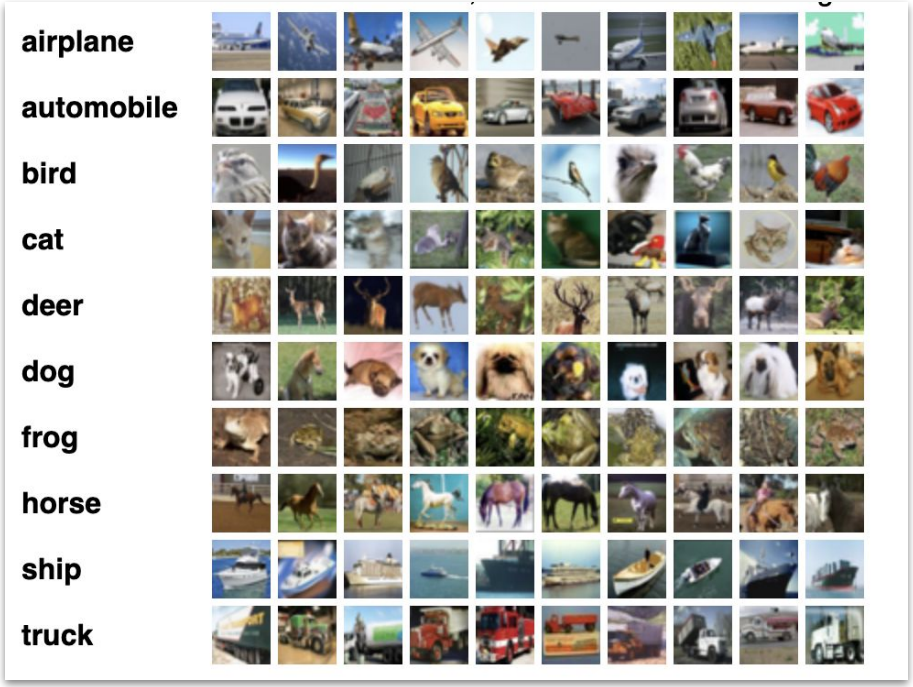
Available callbacks

- Base Callback class
- ModelCheckpoint
- BackupAndRestore
- TensorBoard
- EarlyStopping
- LearningRateScheduler
- ReduceLROnPlateau
- RemoteMonitor
- LambdaCallback
- TerminateOnNaN
- CSVLogger
- ProgbarLogger
- SwapEMAWeights

Reprenons le code d'hier

avec le dataset CIFAR-10

- 50,000 images de training 32x32 et en couleurs (channel = 3)
- 10,000 images de test
- 10 catégories
- [CIFAR-10 and CIFAR-100 datasets]



Implementer un callback EarlyStopping

EarlyStopping : arrêter l'entraînement quand il n'y a plus de gain de performance

https://keras.io/api/callbacks/early_stopping/

```
keras.callbacks.EarlyStopping(
    monitor="val_loss",
    min_delta=0.001,
    patience=5,
    verbose=1,
    mode="auto",
    baseline=None,
    restore_best_weights=False,
    start_from_epoch=0,
)
```

```
model.fit(dataset,
    epochs=10,
    callbacks=my_callbacks
)
```

Atelier - callbacks

https://github.com/SkatAI/deeplearning/blob/master/notebooks/keras_callback.ipynb

CNN
convolutional neural nets
ConvNets

MLP pas fantastique pour les images

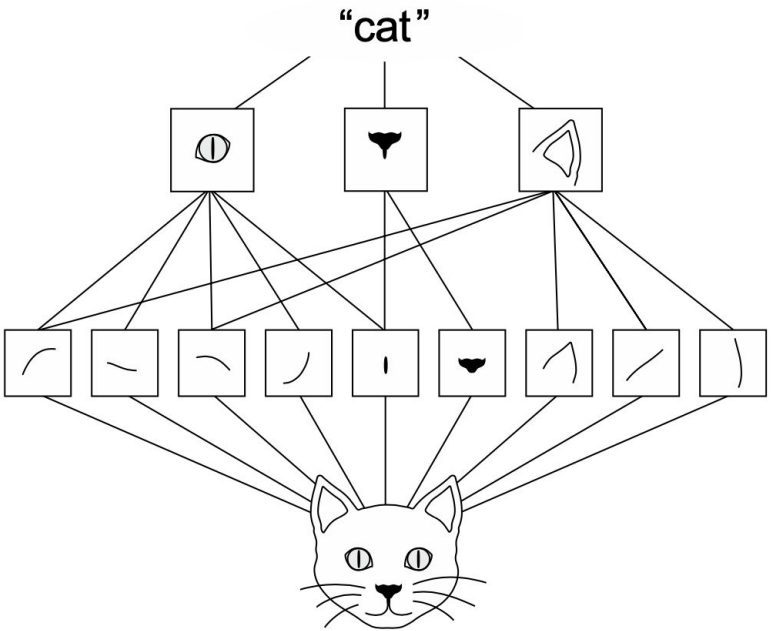
- la couche `Flatten` fait perdre l'information spatiale
- sensible à la translation
  - pourtant si on bouge la chaise … ça reste une chaise !

Le CNN

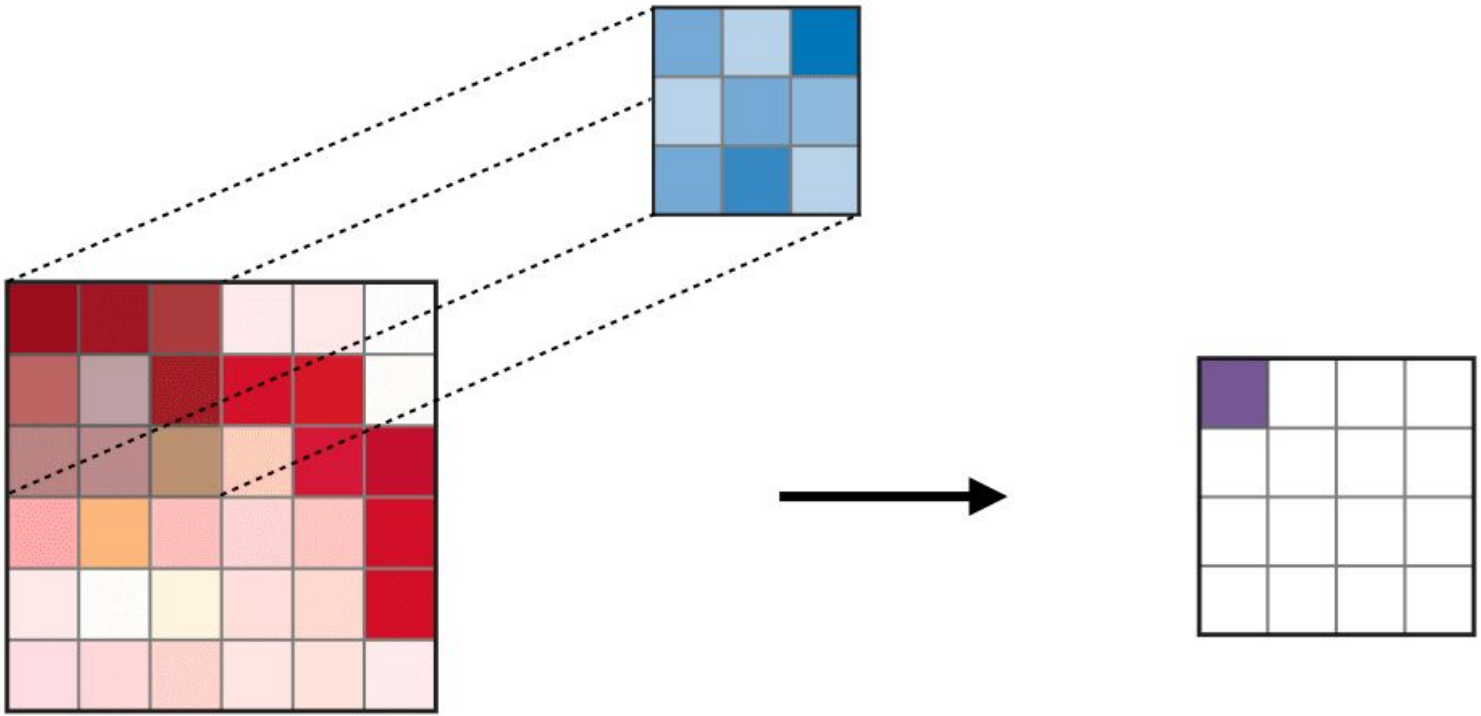
- capture l'information spatiale, les spécificités locales

parfait pour :
- la classification d'images
- la segmentation d'images
- la détection d'objet

mais aussi les séries temporelles :
  - numériques : simples et combinées
  - reconnaissance de la parole
  - classification audio
  - analyse des capteurs (IoT)



Couche de Convolution



https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-convolutional-neural-networks

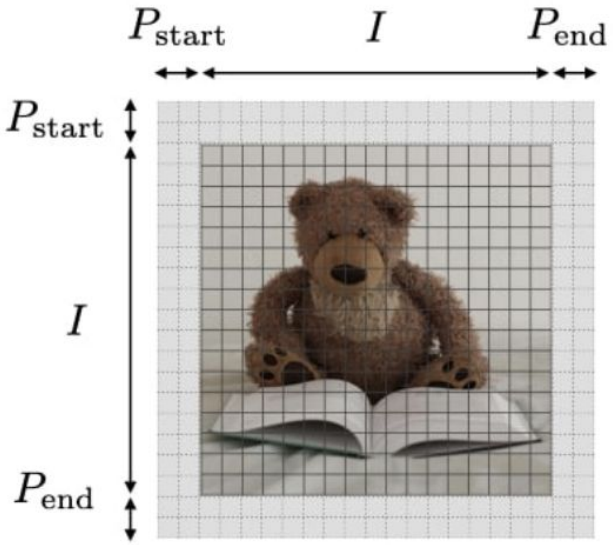
CNN : principe de base

Décomposition de l'image de pixel en pixel, par petit carrés (3x3, 5x5)

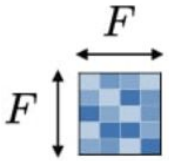
1. calcul de la **convolution** entre chaque carré et un *filtre* donné (tenseur rang 2 de dimension 3x3 ou 5x5, …).
   la valeur en sortie du calcul : un scalaire
2. **pooling** : agréger ces valeurs en prenant le max ou la moyenne des valeurs (sous échantillonnage ou downsampling)
3. enfin, envoyer le tout dans un **MLP** classique

CNN : principe de base

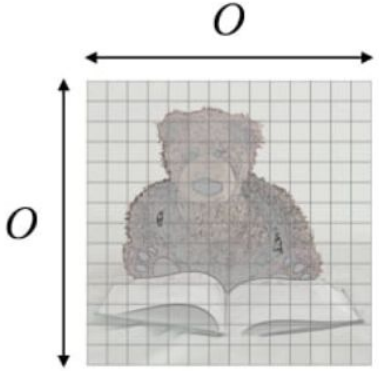
- Le partie **convolution** et **pooling** réalise l'extraction des caractéristiques (features) de l'image d'entrée.
- Les **coefficients du filtre** sont des paramètres d'apprentissage au même titre que les poids des noeuds dans une couche dense
- Combiner plusieurs couches de convolution + pooling permet d'extraire des caractéristiques de plus en plus **complexes**
- On peut enchaîner les couches de convolution sans avoir à chaque fois une couche de pooling



Input

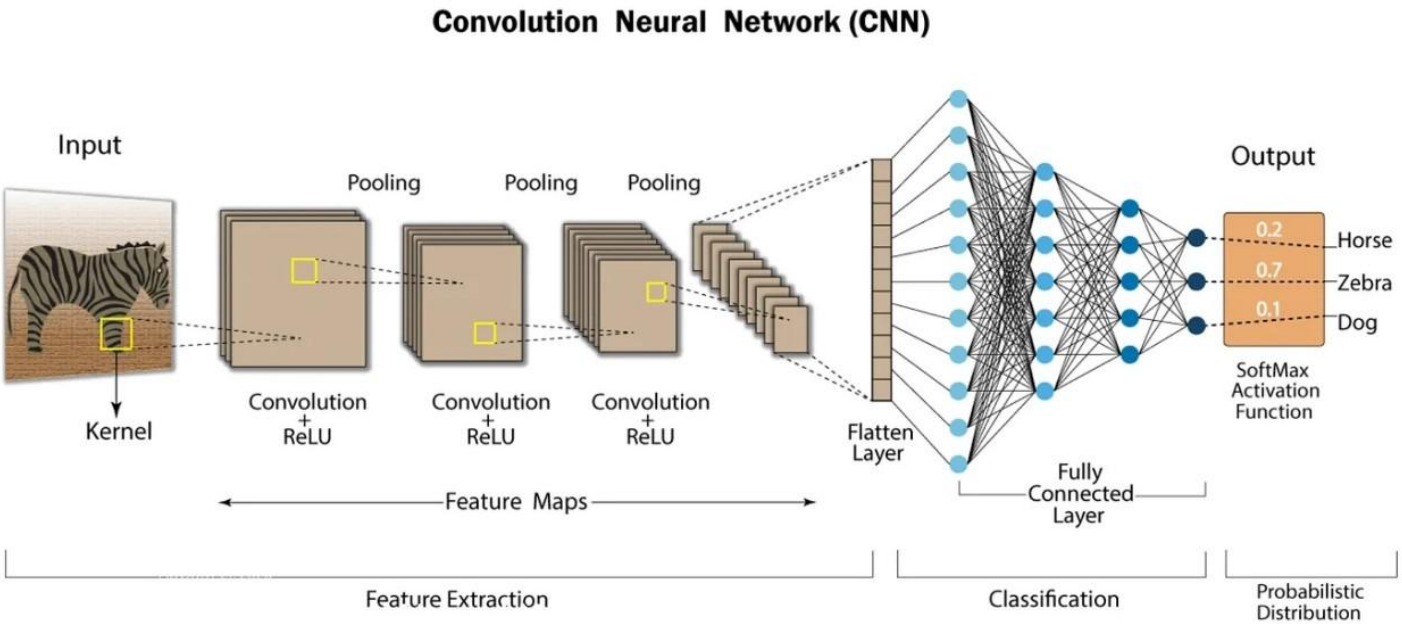


Filter



Output

CNN typique : multiple Conv + Pooling



le calcul de "*convolution*"

Dans les CNN, la "*convolution"* consiste à multiplier les éléments des matrices 2 à 2.

C'est différent de la **convolution** en traitement du signal et des images qui retourne et décale le signal d'entrée.

Cependant le mot *"convolution"* est utilisé en raison du contexte historique, de la similarité conceptuelle, des considérations pratiques et de la cohérence terminologique.

L'idée principale reste la suivante : un noyau (filtre) est appliqué à une entrée (image) pour produire une carte de caractéristiques, qui capture les caractéristiques spatiales importantes de l'entrée.

le calcul de "*convolution*"

Let's assume the following 3x3 filter:

$$\begin{bmatrix} f_{00} & f_{01} & f_{02} \\ f_{10} & f_{11} & f_{12} \\ f_{20} & f_{21} & f_{22} \end{bmatrix}$$

and the following 3x3 region of pixels from the input image:

$$\begin{bmatrix} p_{00} & p_{01} & p_{02} \\ p_{10} & p_{11} & p_{12} \\ p_{20} & p_{21} & p_{22} \end{bmatrix}$$

The convolution operation is performed as follows:

$$\text{Output value} = (f_{00} \cdot p_{00}) + (f_{01} \cdot p_{01}) + (f_{02} \cdot p_{02}) + (f_{10} \cdot p_{10}) + (f_{11} \cdot p_{11}) + (f_{12} \cdot p_{12}$$

De la couche **Dense** à la couche de **Convolution**

Le calcul dans la couche **dense** est la régression : 1/N \sum w_i x_i + b

Convolution :
soit une image de taille h x h

Dans la couche de convolution à 1 filtre de taille pxp:

- parcourt l'image pixel par pixel
  - considère un carré de taille pxp autour du pixel
  - réalise une convolution avec le filtre de taille pxp
- le résultat de la convolution est un scalaire

La couche de convolution à 1 filtre

- a pxp + 1 coefficients : les valeurs du filtre plus le **biais**
- produit en sortie un tenseur de même taille que l'entrée

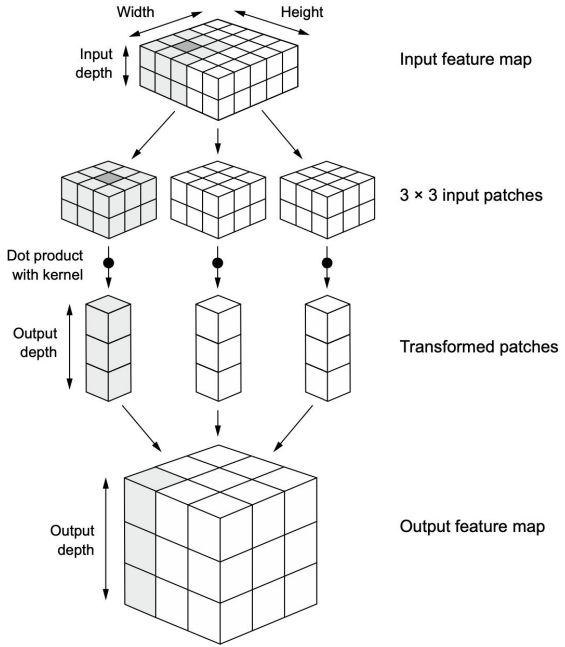
Convolution

Image avec 2 channels

3 Filtres

Sortie de chaque filtre

Sortie de la couche de convolution



Calcul dimension en sortie convolution

Calcul du format en sortie d'une Conv2D de 32 filtres avec en entrée une image couleur (28,28,3) :

- Dimensions de l'image: H=28, W=28, C=3
- Nombres de filtres : N=32
- Dimension du filtre: kH=3, kW=3

en sortie : (28,28,32)

soit 32 x plus de données qu'au départ !!!

Convolution d'une image couleur (3 channels)

```
model = tf.keras.models.Sequential()

model.add(
    tf.keras.layers.Conv2D(
        filters=32,
        kernel_size=(3, 3),
        strides=(1, 1),
        padding='same',
        input_shape=(28, 28, 3))
)
```

Total :

- $32 \times (3 \times 3 \times 3) = 864$
- **896** = 864 + 32

Layer (type)	Output Shape	Param #
conv2d_3 (Conv2D)	(None, 28, 28, 32)	896

Total params: 896 (3.50 KB)

Pooling

Apres la convolution vient le **pooling**

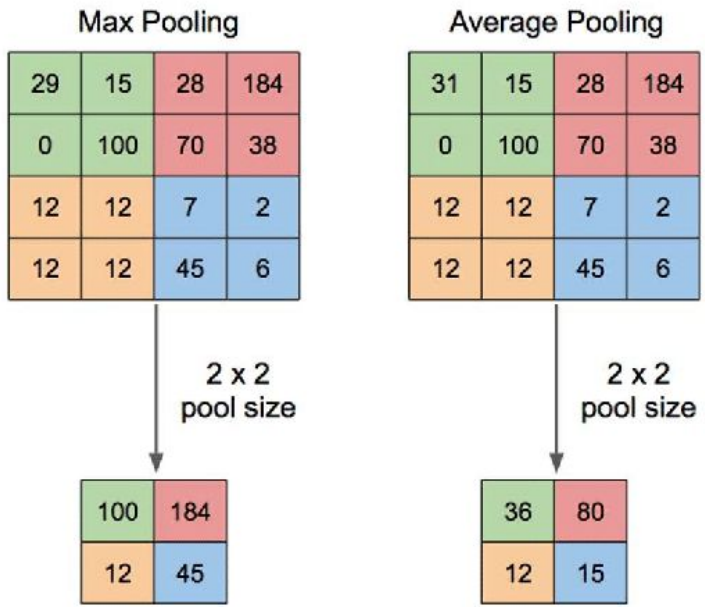
aussi appelée **sous-échantillonnage**

But :

- réduire la taille des données
- extraire les features importantes
- réduire la dépendance à la translation

2 techniques

- prendre le max (plus fréquent)
- prendre la moyenne



Paramètres d'apprentissage d'un CNN

	shape	Nombre de paramètres que le réseau doit apprendre
Input: images grise	28x28x1	-
Conv2D, 32 filtres (3x3)	28x28x32	320
MaxPooling, size 2x2, pas 2	14x14x32	0
Flatten	6272	0
Dense 128 noeuds	128	**802944**
Output 10 noeuds - Softmax	10	1290
		804554

32 x (9 + 1)

14 x 14 x 32

6272 x (128+1)

Interet de multiplier les couches Conv - Pooling

Chaque set va extraire des caractéristiques de plus en plus complexes et générales

Premier ensemble de convolution + pooling :

- La couche de convolution initiale détecte des caractéristiques de bas niveau telles que les contours, les textures et les motifs simples.
- La première couche de pooling réduit les dimensions spatiales, en conservant les caractéristiques les plus importantes tout en réduisant les calculs et en atténuant le surapprentissage.

Deuxième ensemble de convolution + pooling :

- La deuxième couche de convolution s'appuie sur les caractéristiques détectées par la première couche, permettant au réseau de détecter des **motifs plus complexes** et des caractéristiques de niveau supérieur telles que les coins, les formes et les textures.
- La deuxième couche de pooling réduit encore les dimensions spatiales, permettant aux couches plus profondes de détecter des caractéristiques plus abstraites.

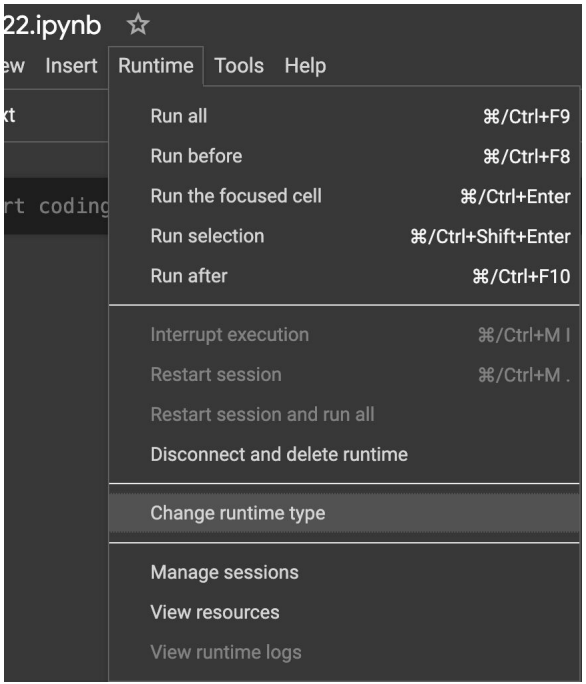
Notion de champ récepteur

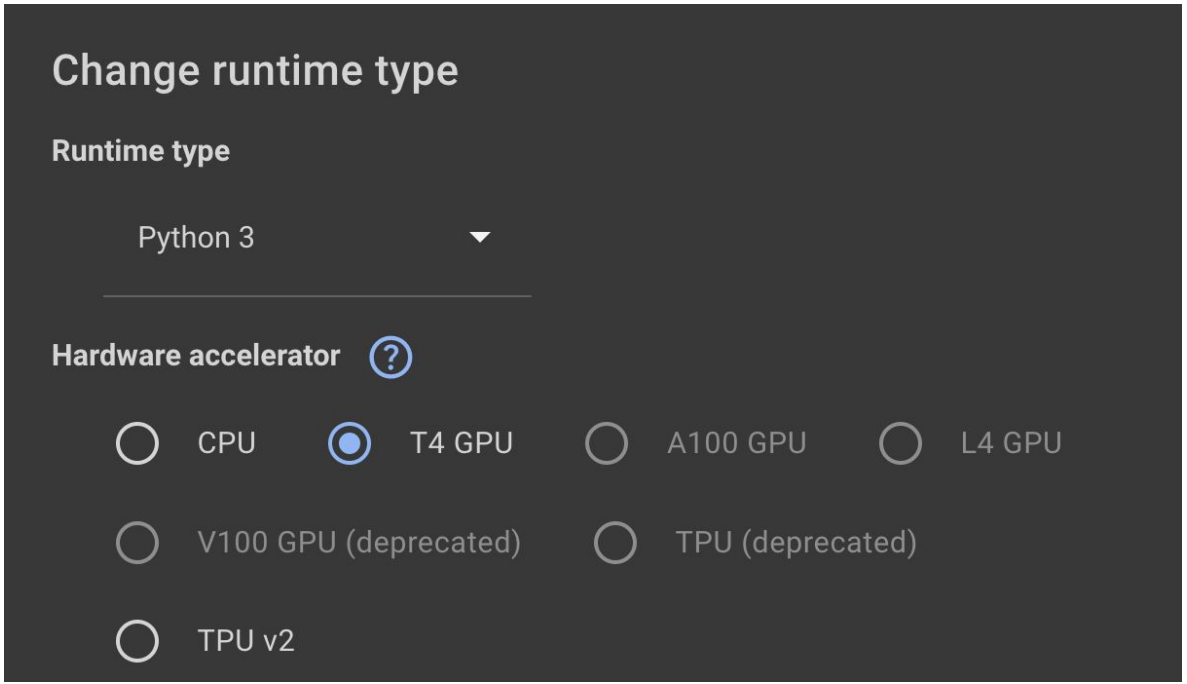
Receptive Field: The receptive field of a neuron in a layer is the region of the input that affects its output.
By stacking convolutional and pooling layers, the receptive field increases, allowing the network to consider larger parts of the input image. This enables the detection of more complex and larger patterns.

Champ Récepteur : Le champ récepteur d'un neurone dans une couche est la région de l'entrée qui affecte sa sortie.
En empilant des couches de convolution et de pooling, le champ récepteur augmente, permettant au réseau de considérer des parties plus larges de l'image d'entrée. Cela permet de détecter des motifs plus complexes et plus grands.

Notebook CNN

Comment activer le GPU dans colab



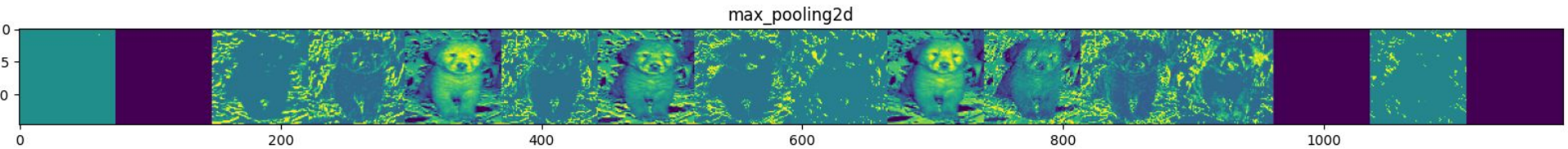


Notebook CNN

Sur le dataset Cats and Dogs

- construire un CNN à 1 puis à 2 Conv + Poolling
- Explorer différentes architectures
- monitorer l'entraînement avec tensorboard
- Augmenter le dataset

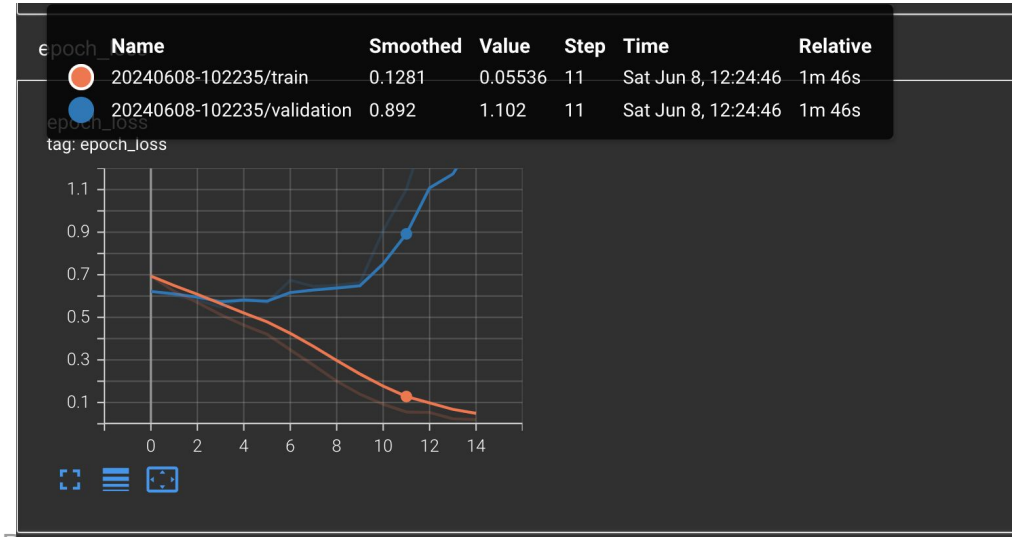
https://github.com/SkatAI/deeplearning/blob/master/notebooks/CNN.ipynb

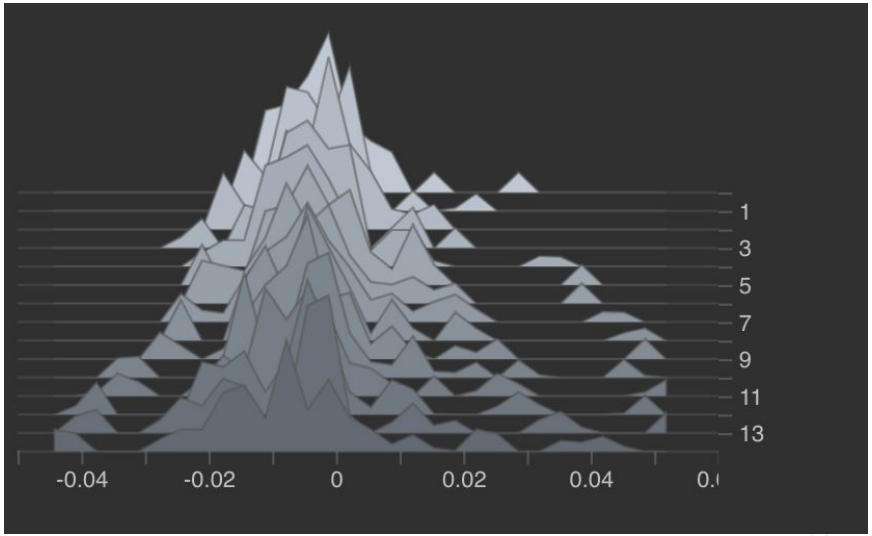


Tensorboard

https://www.tensorflow.org/tensorboard

- Le tableau de bord **Scalar** montre comment le loss et les paramètres changent à chaque epoch. Vous pouvez également l'utiliser pour suivre la vitesse d'entraînement, le taux d'apprentissage et d'autres valeurs scalaires.
- Le tableau de bord **Graphs** vous permet de visualiser votre modèle. Dans ce cas, le graphique Keras des couches s'affiche, ce qui peut vous aider à vous assurer qu'il est correctement construit.
- Les tableaux de bord et **Histogrammes** montrent la distribution d'un Tensor au fil du temps. Cela peut être utile pour visualiser les poids et les biais et vérifier qu'ils changent de manière attendue.





Tensorboard - callback

- https://keras.io/api/callbacks/tensorboard/

```
tensorboard_callback = tf.keras.callbacks.TensorBoard(
            logdir,
            histogram_freq=1
)
```

Tensorboard - pour aller plus loin

- https://www.geeksforgeeks.org/how-to-use-tensorboard-in-google-colab/
- https://neptune.ai/blog/tensorboard-tutorial

flow from directory

pour éviter de charger toutes les données en mémoire on les charge a partir d'un repertoire avec `flow_from_directory` et on en profite pour spécifier

- resize
- couleur ou gris (channels 3 ou 1)
- **batch_size**
- shuffle

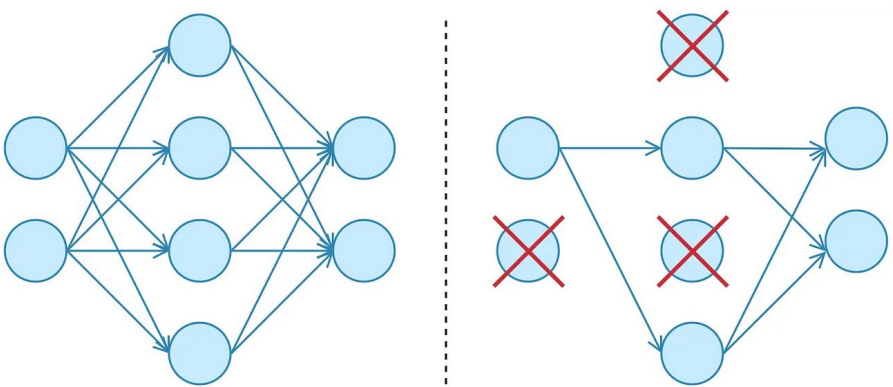
CNN : prévenir l'overfit avec le dropout

On va rajouter une couche de dropout (à la fin du modèle) pour contrer l'overfit

Le paramètre `rate` dicte le % de noeuds qui sont supprimés à chaque epoch

https://keras.io/api/layers/regularization_layers/dropout/

```
keras.layers.Dropout(rate, noise_shape=None, seed=None, **kwargs)
```



techniques de reduction de l'overfit en deep learning

1. Ajouter plus de couches convolutionnelles pour apprendre des caractéristiques plus complexes.
2. Utiliser la batch normalisation pour stabiliser et accélérer l'entraînement. (BatchNormalization layer)
3. Augmenter les données pour avoir plus de diversité dans le dataset d'entraînement
4. Simplifier le modèle en réduisant le nombre de neurones dans les couches denses ou la taille des filtres de convolution.
5. Ajouter de la régularisation L2 aux couches denses.
6. Accroître le dropout, par exemple à 0,5 ou 0,6.
7. Tune le learning rate.
8. Accroître le batch_size et par conséquent le volume de données utilisé pour estimer le gradient

Augmenter les données

A chaque etape, le ImageDataGenerator prend un certain nombre (batch_size) d'images et les transforme aléatoirement (ou non)

De cette façon, le modèle voit des variations des images ce qui réduit l'overfitting

2 modes : création effective des images dans un répertoire

ou à la volée : lorsque les images sont chargées, certaines sont transformées d'autres non

Augmenter les données

Les transformations possibles sont

- translation
- rotation
- Shear Transformation
- Zoom
- Brightness / luminosité
- symetrie verticale ou horizontale
- et Preprocessing

```
datagen = ImageDataGenerator(
    rotation_range=20,
    width_shift_range=0.1,
    height_shift_range=0.1,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)
```

Pour aller plus loin : construire un pipeline input
https://www.tensorflow.org/guide/data

BATCH SIZE

Le gradient qui sert à mettre à jour les poids du noeud lors de la rétropropagation est estimé à chaque fois sur une partie du dataset le nombre d'échantillon estimé à chaque étape est appelé le batch size.

Valeurs courantes : N = 16, 32, 64, …

Le batch size est défini dans le `train_generator`, et le `test_generator`

```
train_generator = train_datagen.flow_from_directory(train_dir,
                                                    batch_size=20,
                                                    class_mode='binary',
                                                    target_size=(150, 150))
```

Comment régler le batch size

Augmenter le batch size	Réduire le batch size
● itérations plus rapides (exploite les GPU) ● besoin de plus de mémoire ● meilleure estimation du gradient ● moins d'étapes	● itérations plus lente ● besoin de moins de mémoire ● estimation du gradient plus variable ● plus de maj

Faut il réduire ou augmenter le batch size pour compenser l'overfit ?

D'un côté **réduire** le batch size réduit aussi l'overfit car :

- plus de bruit dans l'estimation du gradient
- donc moins tendance à coller aux données de training

Mais de l'autre côté **augmenter** le batch size réduit aussi l'overfit car:

- Meilleure estimation du gradient
- Effet de regularization

Qu'en pensez-vous ? Qu'observez-vous ?

approfondissement

```
INPUT -> [[CONV -> RELU]*N -> POOL?]*M -> [FC -> RELU]*K -> FC
```

Lire la partie architecture du cours de Stanford **Convolutional Neural Networks**

https://cs231n.github.io/convolutional-networks/#architectures

Pour aller plus loin

L'excellent cours de stanford, notamment la section sur l'architecture

https://cs231n.github.io/convolutional-networks/#architectures

et ce cours récapitulatif

https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-convolutional-neural-networks

Le chapitre sur la convolution du Deep Learning book

https://www.deeplearningbook.org/contents/convnets.html

Transfer Learning
(fine tuning)

Changement de paradigme ?

Prompt : Could you please generate an image that illustrate a shift in how we do machine learning

- before : create blank model, process data and train model
- now : download pre trained large model and fine tune on small set of data

Résultat de chatGPT :

Here is the illustration showing the shift in machine learning approaches.

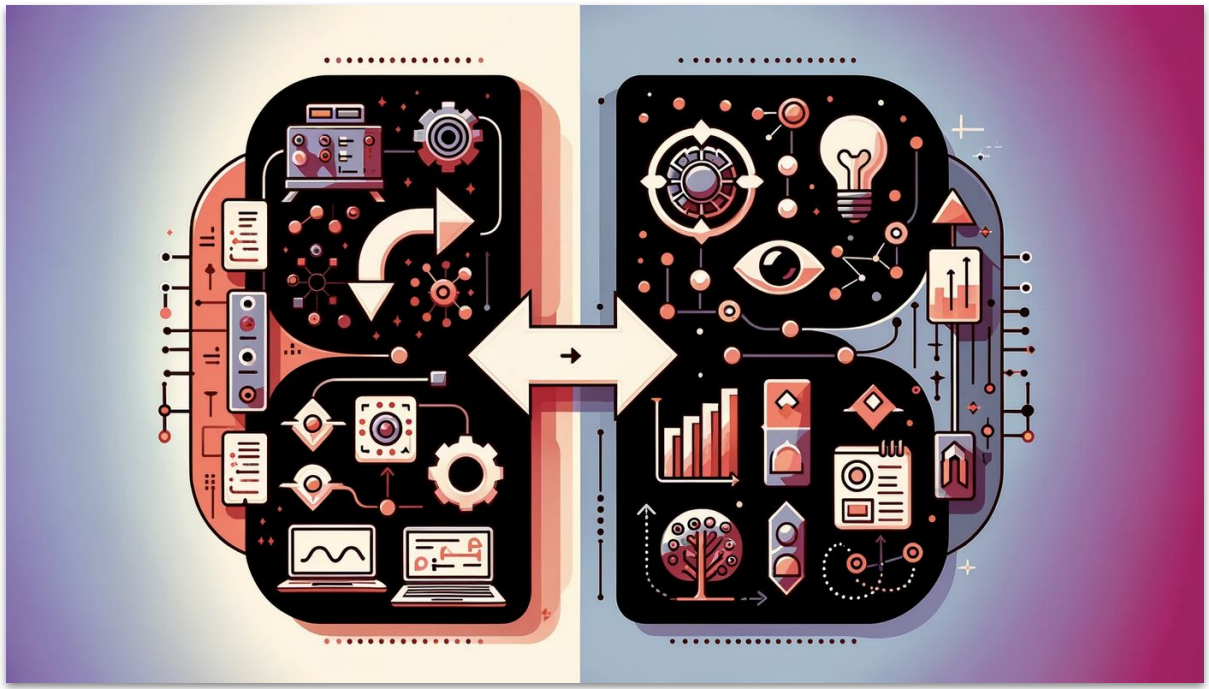
- The left side represents the traditional process of creating a blank model, processing data, and training the model.
- The right side shows the modern approach of downloading a pre-trained large model and fine-tuning it on a small set of data.
- The transition is indicated by an arrow, highlighting the change in methodology.

Changement de paradigme ?

Prompt : could you please generate an image that illustrate a shift in how we do machine learning

- before : create blank model, process data and train model
- now : download pre trained large model and fine tune on small set of data

Résultat :



Changement de paradigme

Jusqu'a récemment le machine learning consiste à

- instancier un modèle vierge (Foret Aleatoire, XGBoost, MLP, CNN, …)
- entraîner le modèle sur des données propriétaires

Maintenant,

- charger un **large modèle pré-entrainé** sur des datasets massifs
- réutiliser le modèle pour l'adapter aux données propriétaires

Ce qui a amené ce changement : la libre mise à disposition des modèles

- https://www.tensorflow.org/hub
- https://huggingface.co : 700k+ modèles
- …

1) le Fine tuning

En prenant pour départ un grand modèle déjà entraîné, l'idée est de réutiliser le modèle pour l'extraction des données

- on ne gèle que quelques unes des couches du modèle
- on ré-entraîne le modèle sur une nouveau dataset de taille conséquente
    - avec un learning rate assez faible pour ne pas perdre la connaissance du modèle initial

Nécessite : **beaucoup de données** et des **ressources informatiques puissantes**

Transfer learning (plus facile)

En prenant pour départ un grand modèle déjà entraîné, l'idée est de réutiliser le modèle pour l'extraction des données

- on gèle les N-1 couches du modèle
- on supprime la dernière couche (celle de classification)
- on ré-entraîne seulement la dernière couche du modèle sur un jeux de donnée de taille réduite

Les couches N-1 sont en fait utilisées pour l'extraction des features

Peu de données, faibles ressources informatiques

Transfer learning - 2 façon d'opérer

Workflow complet

- Charger un modèle pré-entraîné.
- Geler toutes les couches du modèle `trainable = False`.
- Ajouter une couche non gelée en sortie du modèle
- Entraîner votre nouveau modèle sur votre nouveau jeu de données.

Alternative plus légère

1. Charger un modèle pré-entraîné.
2. Faire passer votre nouveau jeu de données à travers ce modèle et enregistrer la sortie du modèle pré-entraîné.
3. Utiliser cette sortie comme données d'entrée pour un nouveau modèle plus petit que vous entraînez. (MLP par exemple)

Dans ce cas vous ne faites passer le modèle de base sur vos données qu'une seule fois, plutôt qu'une fois par époque d'entraînement. Donc c'est beaucoup plus rapide et moins coûteux.

Transfer learning

Combiner les modèles keras

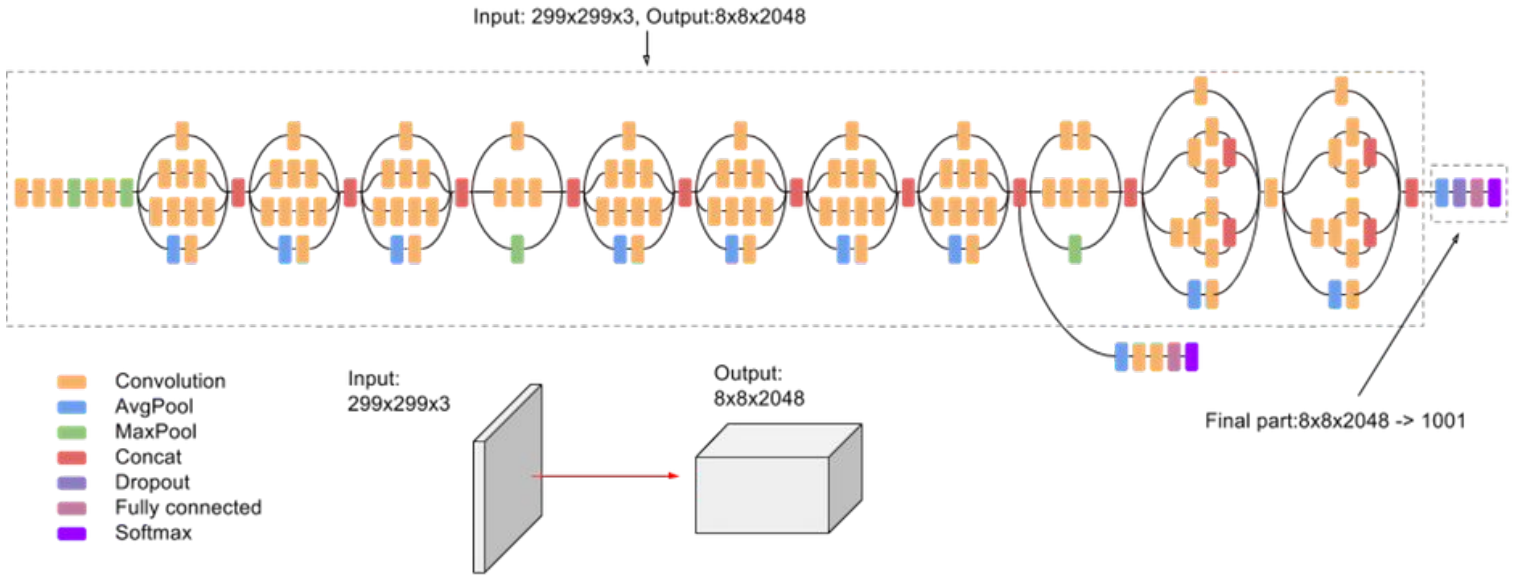
```
inner_model = Sequential([
    layers.Input(shape=(10,)),
    layers.Dense(32, activation='relu'),
    layers.Dense(16, activation='relu')
])

outer_model = Sequential([
    inner_model,  # Include the inner model as a layer
    layers.Dense(64, activation='relu'),
    layers.Dense(1, activation='sigmoid')
])

# Compile the outer model
outer_model.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy'])
```

Transfer learning

On va utiliser le modèle InceptionV3
Total params: 21802784 (83.17 MB)
311 layers!!!



https://paperswithcode.com/method/inception-v3

InceptionV3 - Google

Inception V3

- Introduit en 2015
- Les **modules Inception** combinent des convolutions de 1x1, 3x3, et 5x5 et des pooling layers dans une même couche ce qui capture des caractéristiques à différentes échelles.
- L'architecture inclut également des couches factorisées, des couches de réduction de dimensionnalité et des connexions résiduelles.
- InceptionV3 a été entraîné sur le dataset **ImageNet**, comprenant des millions d'images réparties en 1000 classes.
- environ 24 millions de paramètres.
- Des techniques comme le **batch normalization** et la **régularisation** sont utilisées pour améliorer la stabilité et la performance du réseau.

Notebook Transfer Learning

https://github.com/SkatAI/deeplearning/blob/master/notebooks/transfer_learning.ipynb

Pour aller plus loin

Guide complet sur le transfer learning

https://keras.io/guides/transfer_learning/

Tutorial sur le transfer learning

https://neptune.ai/blog/transfer-learning-guide-examples-for-images-and-text-in-keras