ÉCOLE POLYTECHNIQUE EXECUTIVE EDUCATION

Travail Complémentaire
Data Science Starter Program

Mémoire de Certification CNCP 1527
Conduire un projet de sciences de données

Screening and Diagnosis of esophageal cancer from in-vivo microscopy images by

Mauna Kea Technologies

Le travail complémentaire à fournir pour lever la/les réserve(s) : pour compléter son travail, le candidat devra :

- *Appliquer une baseline de type régression logistique multinomiale, en cross validant le paramètre de pénalisation ;*
- *Valider les résultats avec une matrice de confusion bien renormalisée ;*
- *Expliquer les concepts relatifs aux couches de convolution et au pooling ;*
- *Expliquer et vérifier le fait que les courbes d'apprentissage et de validation soient confondues (on s'attend à ce que la courbe de validation remonte après être descendue).*

Sofiane ALLA

bpifrance INVESTISSEMENT

Sommaire

1. Rappel : introduction et motivation pour le projet, objectifs du travail complémentaire

2. Régression logistique multinomiale

3. Couches de convolution et pooling

4. Courbes d'apprentissage

5. Bibliographie

1. Motivation pour le projet et objectifs du travail complémentaire

Le projet proposé est issu de la plateforme ENS-Collège de France, sur la base d'un challenge proposé par l'entreprise *Mauna Kea Technologies*.

Mauna Kea Technologies développe et conçoit des systèmes **d'imagerie laser** utilisant des **sondes à fibres optiques.** En particulier, les sondes ainsi utilisées permettent de réaliser de l'imagerie cellulaire *in vivo* de haute précision, et d'accélérer ainsi la prise en charge et le diagnostic de maladies liées au système digestif.

L'objectif principal de ce projet est la mise en place d'un **algorithme de classification** de cellules de l'œsophage *in-vivo,* en fonction de la **texture observée**. Les cellules sont issues de patients souffrant du syndrome de l'œsophage de Barrett, un état prédisposant au cancer de l'œsophage.

Un algorithme de ce type, associé à une endoscopie permettrait de **réduire considérablement la phase de diagnostic** du cancer de l'œsophage.

En algorithme de comparaison, *Mauna Kea Technologies* s'est basé sur un réseau de neurones CNN à 3 blocs, chacuns suivie d'une phase de *dropout* et de *pooling.* La technique utilisée permet d'aboutir à une précision de **75% sur le jeu de test**.

Mon choix s'était porté sur ce challenge pour trois raisons principales :

- **La structuration du jeu de données** comportant uniquement (i) des images brutes et (ii) un fichier en format .csv répertoriant les labels. Cette configuration se rapproche fortement de ce que l'on pourrait avoir en **conditions réelles** ;
- **La possibilité d'appliquer les concepts vus en cours** concernant le *pre-processing* des images et les méthodes les plus avancées impliquant des réseaux de neurones pré-entrainés ;
- L'opportunité de se familiariser avec l'architecture de **Pytorch,** qui, à posteriori, ouvre de **nombreuses possibilités** et s'est avérée très utile pour un projet de recherche.

⇒ *Le travail complémentaire permettra (i) d'implémenter un classifieur de type **régression logistique** multinomiale et (ii) de préciser certains concepts évoqués lors de la phase de travail initiale.*

⇒ *Cette classification, plus classique, s'est avérée utile pour introduire et développer des méthodes issues du Data Science Starter Program.*

⇒ *Une attention et un effort particuliers seront apportés aux méthodes de cross-validation, et à l'interprétation des résultats obtenus.*

2. Regression logistique multinomiale

2.a. Description et résultats

La régression logistique est une méthode de classification utile pour la prédiction de variables discrètes. Pour rappel, une regression logistique permet de préciser la relation linéaire entre un vecteur de variables aléatoires indépendantes ($X_1, \dots X_k$) et une variable Y.

En premier lieu, un travail de préparation de la donnée a été nécessaire, en utilisant notamment la librairie Keras pour (i) convertir les images en format *numpy* et (ii) associer chaque image à son label correspondant :

```python
# Load images
train_image = []
for i in tqdm(range(train.shape[0])):
    img = image.load_img('/Users/sofiane.alla/Desktop/DSSP17_Project/TrainingSetImagesDir/'+
                         train['image_filename'][i], target_size=(28,28,1), color_mode = "grayscale")
    img = image.img_to_array(img)
    img = img/255
    train_image.append(img)
X = np.array(train_image)
```

En première approche, la fonction **LogisticRegression** de *scikit-learn* est utilisée avec les paramètres suivants :

```python
model = LogisticRegression(multi_class='multinomial', solver='lbfgs', penalty='l2')
```

⇒ **Multi_class='multinomial'** : la variable Y que nous cherchons à estimer peut prendre 4 valeurs possibles entre 0 et 3 : *Squamous epithelium (*classe : 0), *Intestinal metaplasia* (classe : 1), *Gastric metaplasia* (classe : 2), *Dysplasia and Cancer* (classe : 3) ;

⇒ **Penalty='l2'** :

- Correspond à la méthode de régularisation utilisée pour minimiser la fonction de coût associée à la méthode de régression logistique. Limitant la capacité du modèle à surraprendre sur le jeu d'entrainement.
- Dans la pratique, dans le modèle de Ridge (L2) minimise les variables ayant leur coefficient proche de zéro mais garde l'ensemble des variables dans le modèle.
- Par défaut, le paramètre de pénalisation C est égal à 1, ce qui signifie qu'aucune pénalisation n'est appliquée (plus le paramètre est proche de 0, plus la régularisation est forte).

⇒ **Solver : 'lbfgs'** :

- Solver : correspond à l'algorithme d'optimisation utilisé pour minimiser la fonction de coût ;
- Limited-memory Broyden-Fletcher-Goldfarb-Shanno ou LBFGS est un algorithme d'optimisation utilisé par défaut de Scikit learn et compatible avec une régularisation de Ridge ;
- La documentation Scikit-learn permet de connaitre la compatibilité du solver avec la pénalisation utilisée :

Solvers					
Penalties	**'liblinear'**	**'lbfgs'**	**'newton-cg'**	**'sag'**	**'saga'**
Multinomial + L2 penalty	no	yes	yes	yes	yes
OVR + L2 penalty	yes	yes	yes	yes	yes
Multinomial + L1 penalty	no	no	no	no	yes
OVR + L1 penalty	yes	no	no	no	yes
Elastic-Net	no	no	no	no	yes
No penalty ('none')	no	yes	yes	yes	yes
Behaviors					
Penalize the intercept (bad)	yes	no	no	no	no
Faster for large datasets	no	no	no	yes	yes
Robust to unscaled datasets	yes	yes	yes	no	no

Dans les faits, il est nécessaire de **stabiliser le résultat obtenu en éliminant le bruit** lorsque le modèle est évalué une seule fois, sur l'ensemble du jeu d'entrainement. La cross-validation n'était pas possible pour l'utilisation d'un Resnet. (la puissance de calcul étant limitée)

Pour cela, nous utilisons la méthode dite de ***Repeated k-Fold Cross-Validation*** : le jeu de données est divisé en **k-parties**, chacune de ces parties étant à la fois un sous-jeu d'entrainement et de validation. La procédure est répétée n-fois, le résultat affiché étant la moyenne de l'ensemble des résultats :

```python
from sklearn.model_selection import RepeatedStratifiedKFold
from numpy import mean
from numpy import std
from sklearn.datasets import make_classification
from sklearn.model_selection import cross_val_score

# define the multinomial logistic regression model
model = LogisticRegression(multi_class='multinomial', solver='lbfgs', penalty='l2')
# define the model evaluation procedure
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
# evaluate the model and collect the scores
n_scores = cross_val_score(model, X, y, scoring='accuracy', cv=cv, n_jobs=-1)
# report the model performance
print('Mean Accuracy: %.3f (%.3f)' % (mean(n_scores), std(n_scores)))
```

Dans notre cas, le jeu d'entrainement a été divisé en 10 parties et la procédure répétée 3 fois. **En utilisant cette méthode de cross-validation, nous obtenons une *accuracy* moyenne de 71% sur le jeu de données :**

```
Mean Accuracy: 0.705 (0.013)
```

Toutefois, ne pourrions-nous pas atteindre un meilleur score en calibrant le paramètre de pénalisation ?

2.b. Cross-validation du paramètre de pénalisation C

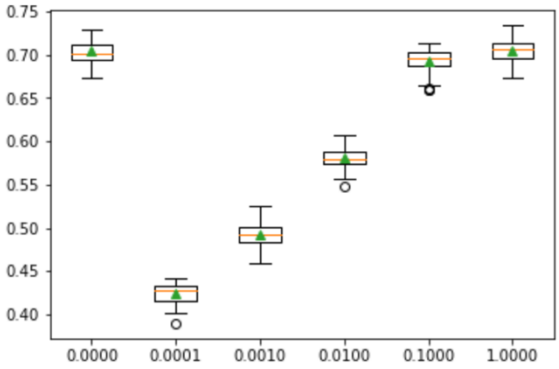
Le paramètre de pénalisation C est un hyperparamètre clé permettant de calibrer la régularisation L2. L'objectif est de cross-valider le paramètre C compris entre 0 et 1. Plus C est proche de 0, plus la pénalisation est forte.

Pour l'effectuer, nous introduisons les fonctions *get_models* et *evaluate_models* permettant de tester **cinq valeurs de C entre 0,0001 et 1,0 :**

```python
# get a list of models to evaluate
def get_models():
    models = dict()
    for p in [0.0, 0.0001, 0.001, 0.01, 0.1, 1.0]:
        # create name for model
        key = '%.4f' % p
        # turn off penalty in some cases
        if p == 0.0:
            # no penalty in this case
            models[key] = LogisticRegression(multi_class='multinomial', solver='lbfgs', penalty='none')
        else:
            models[key] = LogisticRegression(multi_class='multinomial', solver='lbfgs', penalty='l2', C=p)
    return models

# evaluate a give model using cross-validation
def evaluate_model(model, X, y):
    # define the evaluation procedure
    cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
    # evaluate the model
    scores = cross_val_score(model, X, y, scoring='accuracy', cv=cv, n_jobs=-1)
    return scores
```

En effectuant la cross-validation, nous obtenons les résultats pour les 5 valeurs de C testées, modélisées à travers des *box plots* :

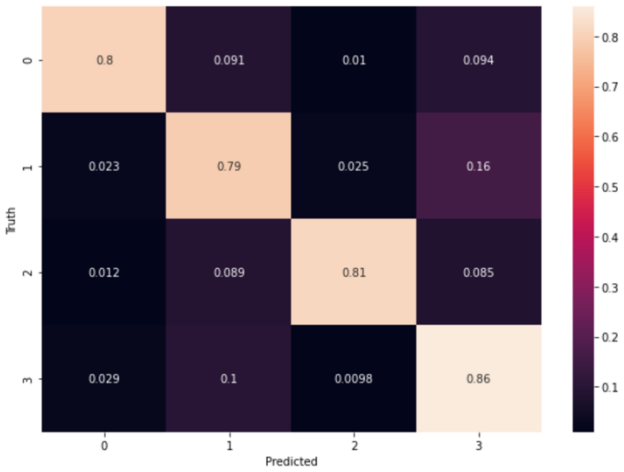


Plus la pénalisation est forte, moins le modèle est précis et donc moins le modèle est flexible pour prédire des variables ne faisant pas partie du jeu d'entrainement de départ.

2.c. Focus : validation des résultats par une matrice de confusion

La matrice de confusion est indicateur simple, permettant d'estimer le taux d'erreur ou le taux de mauvais classement, qui est le rapport entre le nombre de mauvaises prédictions et la taille de l'échantillon. Lors du travail initial.

En testant le modèle ayant la meilleure a*ccuracy* (C = 1), la matrice de confusion **normalisée** (i.e. chaque résultat est divisé par le nombre d'images testées) permet d'avoir la répartition suivante :



La matrice de confusion est une excellente méthode permettant de rapidement déceler les faux négatifs, dans notre cas les images classées en catégorie 0 à 2 alors qu'elles reflétaient des cellules cancéreuses ou dysplasiques (catégorie 3). **Le taux de faux négatifs est de 14% sur le modèle utilisé.**

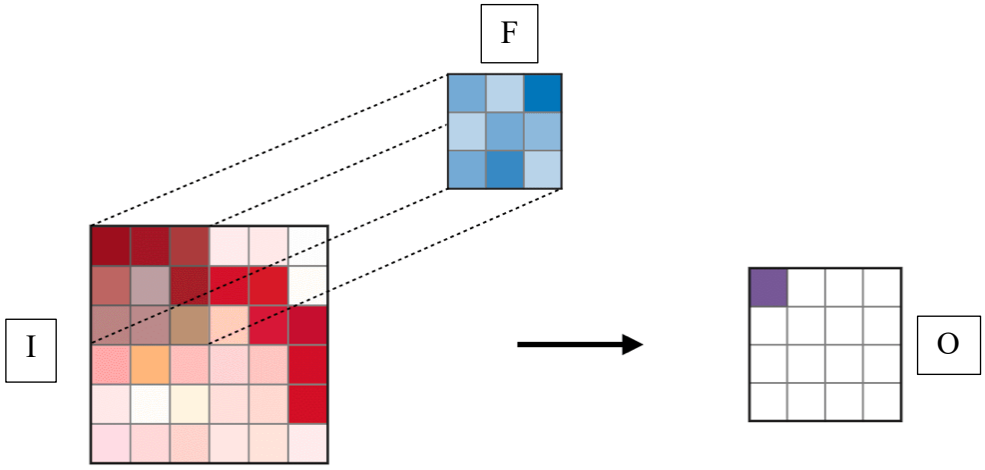
3. CNN : couches de convolution et *pooling*

3.a. Convolution

Mathématiquement parlant, une convolution est une opération permettant d'obtenir une fonction h à partir fonction f et g.

Dans un réseau de neurones convolutionnel ou CNN, la couche convolutionnelle utilise des filtres qui *scannent l'entrée I suivant ses dimensions en effectuant des opérations de convolution. Elle peut être réglée en ajustant la taille du filtre F et le stride S. La sortie O de cette opération est appelée feature map ou aussi activation map.* (*source : Stanford CS 230 – apprentissage profond*)

En *deep learning,* le principal intérêt de la convolution est la capacité du modèle à **extraire des caractéristiques** (*features)* communes à plusieurs images classées dans la même catégorie, et ce même si elles ne ressemblent pas strictement. La convolution peut être à 1 dimension (traitement de signal), 2 dimensions (traitement d'image) ou 3 dimensions (traitement vidéo).



L'opération de convolution est définie par 3 hyper-paramètres :

⇒ **Stride** : Dans le contexte d'une opération de convolution ou de pooling, la stride S est un paramètre qui dénote le nombre de pixels par lesquels la fenêtre se déplace après chaque opération. **Plus S est grand, plus le volume de sortie O est faible.**

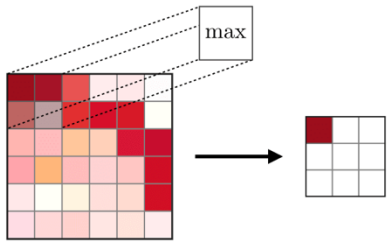
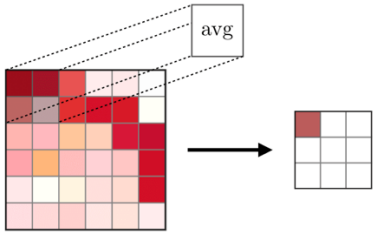
⇒ **Zero-padding** : Le zero-padding est une technique consistant à ajouter P zeros à chaque côté des frontières de l'entrée. **Le padding est utile car les pixels au centre de l'image sont techniquement couverts plus souvent par la convolution que les pixels de trouvant à l'extrémité.**

⇒ **Dimensions du filtre** : le filtre est la matrice (symétrique) utilisée pour extraire les *features* issues de chaque sous-région de l'image analysée, et se déplaçant à une vitesse déterminée par le Stride S. le plus communément utilisé est le filtre de dimension 3x3.

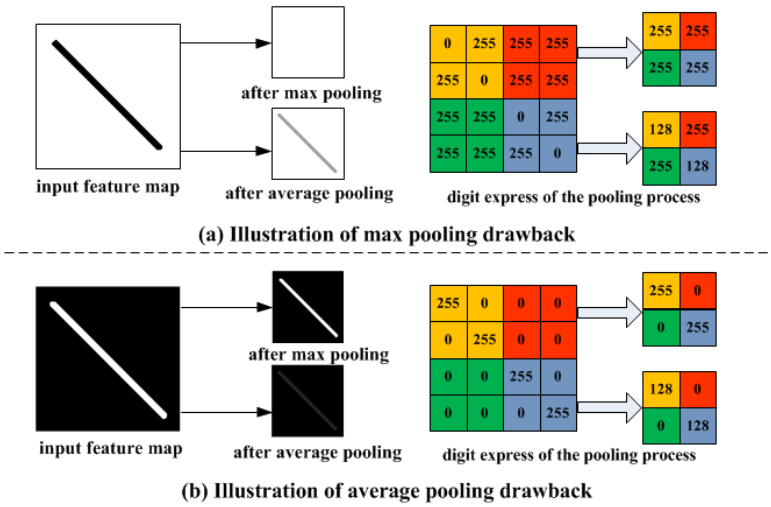
3.b. Pooling

La couche de *pooling* (en anglais pooling layer) (POOL) est une opération de sous-échantillonnage typiquement appliquée après une couche convolutionnelle. **Le principal intérêt du *pooling* est la réduction de dimension, tout en conservant les caractéristiques extraites de la couche de convolution.**

En particulier, les types de *pooling* les plus populaires sont le max et l'average pooling, où les valeurs maximales et moyennes sont prises.(*source : Stanford CS 230 – apprentissage profond*)

Type	Max pooling	Average pooling
But	Chaque opération de pooling sélectionne la valeur maximale de la surface	Chaque opération de pooling sélectionne la valeur moyenne de la surface
Illustration		
Commentaires	• Garde les caractéristiques détectées • Plus communément utilisé	• Sous-échantillonne la feature map

Comment choisir entre une méthode de *max pooling* et une méthode *d'average pooling ?*



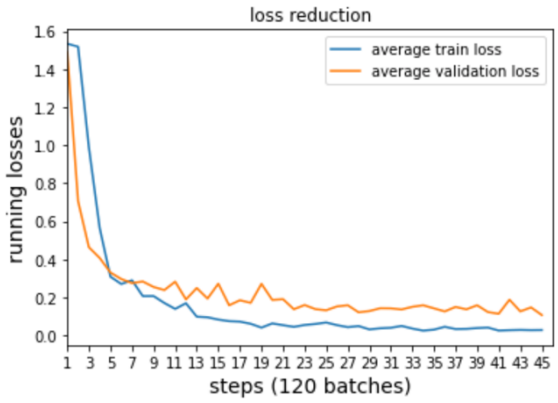
(Source : Research Gate)

En général, la méthode d'*average pooling* est plus souvent utilisée :

"We believe that while the max and average functions are rather similar, the use of average pooling encourages the network to identify the complete extent of the object. The basic intuition behind this is that the loss for average pooling benefits when the network identifies all discriminative regions of an object as compared to max pooling" (*Zhou et al. 2016*)

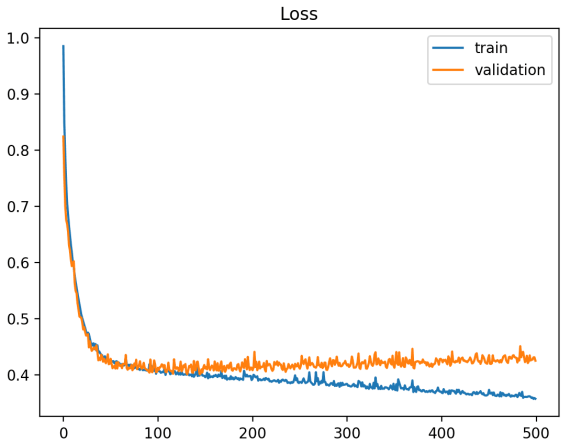
4. CNN : courbe d'apprentissage

Pour rappel, en utilisant un algorithme *VGG16* nous avions obtenu la courbe d'apprentissage suivante :



La courbe de perte sur le jeu de validation se confond avec la courbe sur le jeu d'entrainement. La confusion des deux courbes est un signe de stabilité du modèle : la courbe de validation remontant étant un signe *d'overfitting*.

Toutefois, la courbe de perte sur le jeu de validation devrait remonter après un certain nombre d'itérations selon le modèle suivant :



(source : machine learning mastery)

Une des explications possibles est **l'algorithme d'entrainement** utilisé pour calibrer le modèle : un ***early-stopping*** a été introduit pour suspendre la calibration du modèle une fois l'objectif d'accuracy atteint.

```
if accuracy >= 0.97:
    print('Performance condition satisfied, stopping..')
    save_file = '/DSSP17_Project/resnet18.pt'
    torch.save(model_resnet.state_dict(), save_file)
    return running_train_loss, running_val_loss
```