



Machine Learning Avancé : Questions.

Auteurs : Sarra Madad & Manelle Nouar
Enseignant : Vincent DESPIEGEL

20 décembre 2022

1 | Questions

1.1 Quel type d'architecture avez-vous utilisé, pourquoi ?

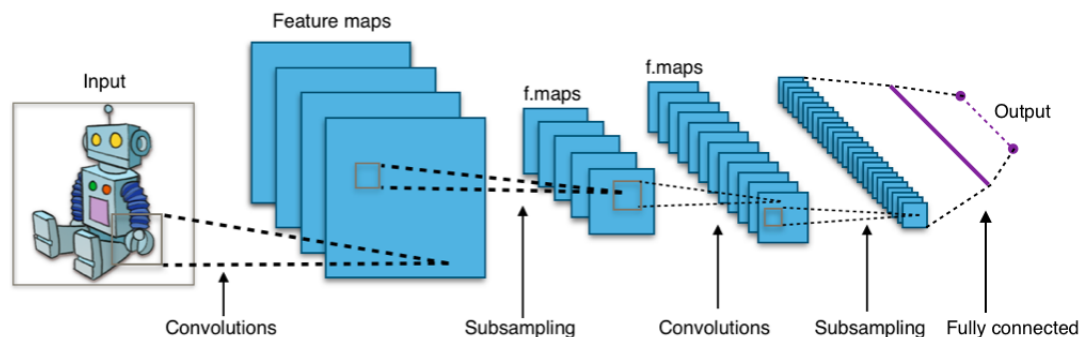


FIGURE 1 – CNN

Nous avons utilisé un CNN : réseaux de neurones à convolution, très utilisé pour la reconnaissance d'image.

Les images ont une taille de 130 x 130, que l'on a converti en matrice `np.array` pour introduire dans le réseau.

Nous sommes partis d'une architecture basique à 3 couches de convolution pour une classification binaire mutuellement exclusive (chien ou chat par exemple) et nous avons adapté les fonctions d'activation, l'optimiser ainsi que le nombre de couches et ajouté des paramètres supplémentaires.

1.2 Combien de convolutions contient votre architecture, pourquoi ?

Le nombre de couches de convolution est un hyperparamètre, à moduler en fonction de la complexité de la classification.

Par exemple, les modèles pré-entraînés ont parfois des quinzaines de couche de convolution pour la classification d'image.

Plus il y a de couches mieux c'est, mais parfois il est nécessaire de garder un modèle "simple" au lieu d'ajouter des couches pour gagner peu de pourcentage en accuracy.

En augmentant les couches petit à petit, on peut arriver à un nombre optimal pour notre problème en particulier.

Un minimum de 3 couches pour de la classification d'image binaire est néanmoins requis. En ajoutant une couche supplémentaire donc 4 couches en tout, nous arrivons à des résultats satisfaisants.

```

model = Sequential()
model.add(Conv2D(filters=16, kernel_size=(5,5), activation="relu", input_shape=(130, 130, 3)))
#relu car Label 0 ou 1 (tanh 1 ou -1)
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size = (2, 2)))
model.add(Dropout(0.2)) #éviter sur-apprentissage

model.add(Conv2D(filters=32, kernel_size=(5,5), activation="relu"))
model.add(MaxPooling2D(pool_size = (2, 2)))
model.add(BatchNormalization())
model.add(Dropout(0.2))

model.add(Conv2D(filters=64, kernel_size=(5,5), activation="relu"))
model.add(MaxPooling2D(pool_size = (2, 2)))
model.add(BatchNormalization())
model.add(Dropout(0.2))

model.add(Conv2D(filters=128, kernel_size=(5,5), activation="relu"))
model.add(MaxPooling2D(pool_size = (2, 2)))
model.add(BatchNormalization())
model.add(Dropout(0.2))

model.add(Flatten())
model.add(Dense(128, activation = 'relu'))
model.add(Dropout(0.2))
model.add(Dense(64, activation = 'relu', kernel_regularizer=l2(0.001)))
model.add(Flatten())
model.add(Dropout(0.2))
model.add(Dense(1, activation = 'sigmoid'))
model.add(Flatten())

model.compile(loss='binary_crossentropy', optimizer = 'RMSprop', metrics=['accuracy']) #optimizer = 'Adam' #adaptive learning rate

#il ne faut pas utiliser softmax pour la classification binaire, elle est plus intéressante pour les problèmes multiclassés
#mutuellement exclusives où chaque classe a une probabilité et la plus grande gagne

#Les sorties de la fonction Sigmoid sont des probabilités
#bien pour les problèmes binaires mutuellement exclusifs (chat ou pas chat)

```

FIGURE 2 – Modèle CNN

1.3 Comment vous êtes-vous préoccupé de l'over-fitting ?

Pour rappel :

- Overfitting = sous-apprentissage (pas assez de données pour s'entraîner donc mauvaise accuracy générale)
- Underfitting = sur-apprentissage (biais faible et variance trop forte donc mauvaise généralisation, accuracy en test qui s'écroule et loss en test très forte)

Pour éviter le sur-apprentissage, on ajoute des couches "Dropout" et on vérifie à chaque epoch grâce à la validation croisée.

Cela nous permet de voir, en comparant avec la perte, si le modèle généralise bien et n'est pas en sur-apprentissage.

On utilise aussi des régularisateurs de poids L1 et L2.

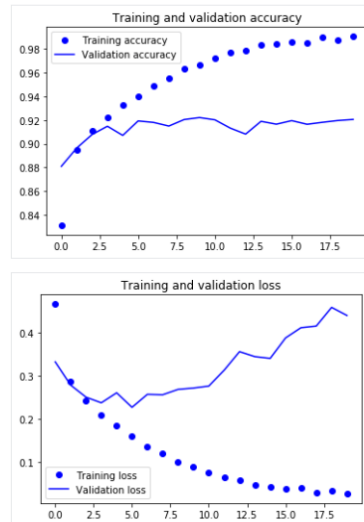


FIGURE 3 – Courbes d’accuracy et de loss d’un modèle CNN en overfitting

Exemple de sur-apprentissage :

D’après les deux graphiques ci-dessus, on peut voir que l’accuracy en validation a stagné après 4-5 epochs et a rarement augmenté à certaines epochs. Au début, l’accuracy en validation augmentait linéairement avec la perte, mais ensuite elle n’a pas beaucoup augmenté.

La perte (loss) en validation montre que c’est le signe d’un sur-apprentissage, comme l’accuracy en validation, elle a diminué linéairement mais après 4-5 epochs, elle a commencé à augmenter. Cela signifie que le modèle a essayé de mémoriser les données et a réussi.

1.4 Est-ce que les labels bruités ont posé problème. Comment l’avez-vous géré ?

Le bruit sur des images sont toujours gênants, il ajoute un flou.

Il peut être nettoyé à l’aide de filtres médians ou par un simple flou gaussien de l’image. La transformation de Fourier peut également être utilisée en cartographiant l’image bruyante dans un domaine de fréquences, en supprimant les fréquences indésirables, puis en la cartographiant à nouveau dans le domaine spatial.

Le bruit peut être éliminé en utilisant de très nombreuses données d’entraînement afin que le CNN puisse déterminer quelles sont les caractéristiques importantes ou non. En bref, les grands ensembles de données présentent un rapport signal/bruit de bonne qualité dans l’ensemble.

On peut introduire des méthodes de régularisation telles que :

- Dropout
- Fonctions de coût robustes
- Batch normalization
- Garder le modèle simple

1.5 Est-ce que l'asymétrie de la base d'apprentissage est un problème? si oui, comment avez traité?

Oui c'est un problème, cela signifie que le modèle risque d'apprendre à mieux différencier une classe que l'autre.

Dans notre cas, nous avons 2 fois plus de classe 1 que de classe -1, ce qui signifie que le modèle aurait mieux appris à classer la classe 1 que la classe -1.

On gère l'asymétrie de classe avec la data augmentation pour compenser la classe manquante, et le F1 score.

Nous avons également visualiser la matrice de confusion afin de visualiser la répartition des classes par le modèle, et nous avons sorti un rapport de classification. (visibles dans le Jupyter Notebook)

1.6 Pour les images de la base de test, à partir de la sortie de votre réseau, comment attribuez-vous le label?

Nous avons choisi la fonction d'activation Sigmoid : parfaite pour les problèmes binaires mutuellement exclusifs (chat ou pas chat).

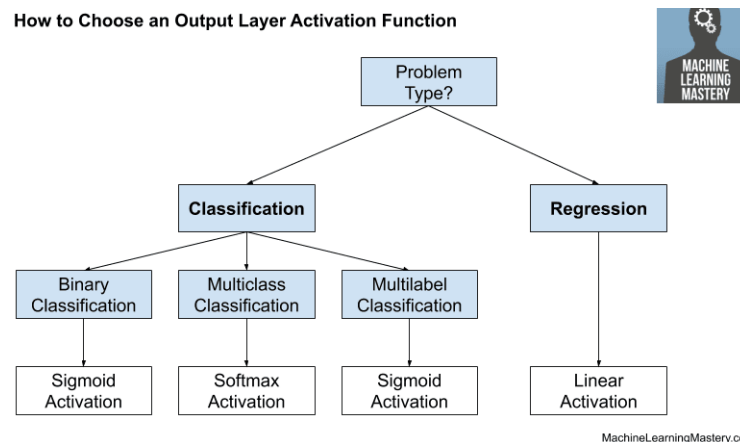


FIGURE 4 – Choix de la fonction d'activation de sortie

Il ne faut pas utiliser Softmax pour la classification binaire, elle est plus intéressante pour les problèmes multiclassés mutuellement exclusifs où chaque classe a une probabilité et la plus grande gagne.

Les sorties de la fonction Sigmoid sont des probabilités, nous avons donc établi un seuil de 50% :

- Classe 0 (ou -1) si la probabilité est $\leq 50\%$
- Classe 1 si la probabilité est $> 50\%$

1.7 Avec un mois supplémentaire pour travailler, que feriez-vous pour améliorer vos résultats?

Avec un mois supplémentaire, nous aurions pu :

- Faire une PCA sur nos images (compression)

- Calculer le temps d'inférence sur le modèle pour comparer les performances + l'accuracy en ajoutant/retirant des couches
- Utiliser du Transfer Learning (VGG 16, ResNet 50, Inception V3) pour réduire le temps et la capacité de calcul
- Mettre en place un TensorBoard
- Passage en GPU (bug sur mon PC donc cette étape a été écartée)

1.8 Question non notée : à quoi correspond le label recherché ?

En se basant sur cette base de données : <https://www.kaggle.com/datasets/jessicali9530/celeba-dataset>

On peut donc penser que les labels 1 et -1 correspondent à OUI ou NON pour une feature donnée.




▲ image_id	# 5_o_Clock_Shadow	# Arched_Eyebrows	# Attractive
202599 unique values			
000001.jpg	-1	1	1
000002.jpg	-1	-1	-1
000003.jpg	-1	-1	-1
000004.jpg	-1	-1	1
000005.jpg	-1	1	1

FIGURE 5 – Extrait de la base de donnée