LOUIS, LE LAY: TP MACHINE LEARNING

(MSE/ISMIN, 2A, 2023-2024)

Jérémy Louis, Gaël-Mehdi Le Lay
jeremy.louis@etu.emse.fr, gael-mehdi.lelay@etu.emse.fr

January 31, 2024

1 The datasets: CIFAR-3

1.1 Question 1

Le dataset CIFAR-3 est de la forme $(18000, 32, 32, 3)$. Cela correspond à 18000 images de 32 pixels par 32 pixels en couleurs, soit avec 3 composantes : une pour le niveau de rouge, une de bleu et une de vert. Chaque composante est un entier entre 0 et 255.

Le dataset CIFAR-3-GRAY est lui de la forme $(18000, 32, 32)$. Cela correspond à 18000 images de 32 pixels par 32 pixels en couleurs, avec une composante de 0 à 255 qui représente le niveau de gris du pixel.

Oui, il est préférable de normaliser les valeurs des pixels entre 0 et 1. Des valeurs de pixels normalisées limitent la variance initiale des poids et aident à stabiliser le gradient pendant la formation.

Voici ci-dessous une image de la classe 2, soit un cheval :

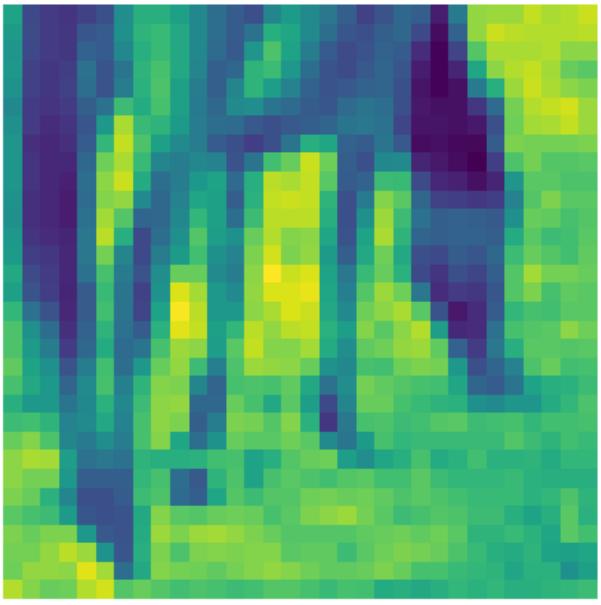

Figure 1: Image issue de la classe 2

1.2 Question 2

Pour séparer le dataset d'apprentissage du dataset de test, nous utilisons la fonction `train_test_split` de `sklearn.model_selection`. Cette fonction ressort 4 vecteurs différents : chaque dataset accompagné des labels qui lui sont associés. Le paramètre `test_size` permet de choisir la taille du dataset de test. On choisit généralement une valeur entre 20% et 30% de la taille du dataset initial. L'usage du paramètre `shuffle` est détaillé dans la question suivante.

Il est important de séparer le dataset d'apprentissage de celui de test car cela n'aurait pas de sens de mesurer la performance d'un modèle grâce à ses résultats sur le dataset d'apprentissage. Le but d'un modèle est qu'il apprenne afin d'obtenir de bons résultats sur des données inconnues, c'est pour cela qu'il est nécessaire d'utiliser des données "nouvelles" sur lesquelles l'évaluer.

Finalement, mesurer l'apprentissage d'un modèle sur ses résultats sur le dataset d'apprentissage ne permet pas de mesurer l'overfitting du modèle. En effet, un modèle peut avoir de très bons résultats sur son dataset d'entraînement car ce dernier apprend "par coeur" des paternes qui pourraient ne pas être présents dans des données réelles.

1.3 Question 3

La commande `shuffle = True` permet de mélanger les données du dataset afin que les trois classes soient équitablement présentes dans le dataset d'entraînement et celui de test.

2 Dimensionality reduction with the PCA

2.1 Question 1

La PCA a été réalisée, voir le Jupyter Notebook.

Afin de réaliser la PCA, il est nécessaire de changer la forme des données. En effet, comme vu lors de la Question 1.1, le dataset CIFAR-3-GRAY est de la forme $(18000, 32, 32)$. Cependant, la PCA nécessite un vecteur de dimension $(n, m) \in \mathbb{N}^2$.

Il faut donc redimensionner le dataset sur lequel appliquer la PCA. Pour cela, nous utilisons la commande `X_gray_train = np.reshape(X_gray_train,(np.shape(X_gray_train)[0],32*32))`. Cette commande permet de redimensionner chaque image sous la forme d'un vecteur de dimension $32 \times 32 = 1024$.

2.2 Question 2

La valeur `PCA.explained_variance_ratio_` est un tableau permettant de connaître le ratio de participation de chaque composante initiale à la variance initiale d'une donnée. Naturellement, plus on utilise une valeur élevée pour `n_components`, plus la somme des valeurs du tableau `PCA.explained_variance_ratio_` se rapproche de 1. Pour une valeur `n_components = 1024`, la somme des valeurs de `PCA.explained_variance_ratio_` vaut 1.

Afin de trouver la meilleure valeur de `n_components`, nous avons utilisé une boucle qui permet de trouver la plus petite valeur permettant d'obtenir une variance totale de 95%. Nous trouvons une valeur `n_components = 174` pour obtenir cette valeur de variance.

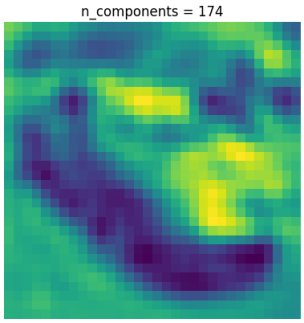

Figure 2: Image avec une variance totale de 95%

2.3 Question 3

Voici ci-dessous une image de la classe 0, i.e. une voiture, pour 5 valeurs différentes de `n_components` :

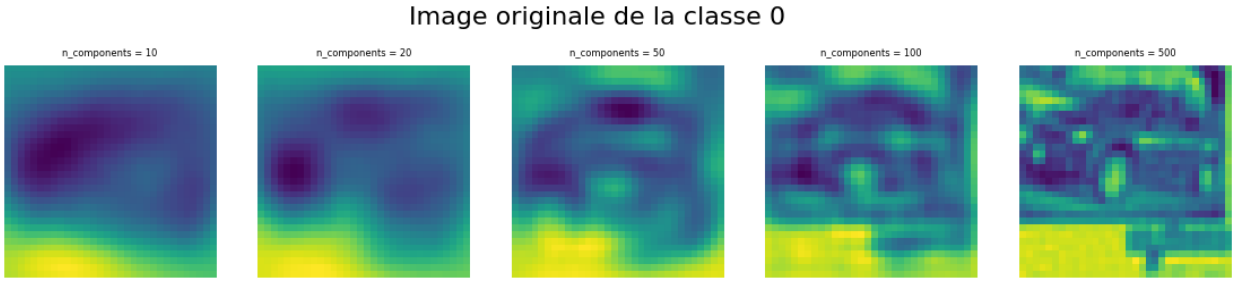

Figure 3: Image de la classe 0 pour 5 valeurs différentes de n_components

3 Supervised Machine Learning

3.1 Logistic Regression, Gaussian Naive Bayes Classifier

3.1.1 Question 1

Avec une régression logistique, nous estimons la quantité $P(Y|X)$ alors qu'avec une classification naïve bayésienne, nous estimons $P(X|Y)$ afin d'obtenir $P(Y|X)$ à l'aide de la formule de Bayes.

3.1.2 Question 2

Le seul paramètre que nous pouvons modifier est le ratio entre la taille du dataset d'entraînement et celui de test.

Les résultats de la régression logistique sont légèrement meilleurs que ceux de la classification naïve bayésienne. En effet, avec une taille de 20% pour le dataset de test nous obtenons un taux de réussite de 58% avec la classification naïve bayésienne et 60% avec la régression logistique.

3.1.3 Question 3

Les résultats ont été affichés. Il est nécessaire d'utiliser les résultats d'entraînement et de test afin de mesurer l'overfitting.

3.1.4 Question 4

La classification naïve bayésienne et la régression logistique ont été réalisées avec le dataset passé par la PCA. Nous avons utilisé la PCA qui conserve une variance de 95% de la variance initiale.

Les résultats de ces deux processus sont quasiment similaires au résultats précédents, c'est la preuve de l'utilité de la PCA : nous avons gardé une grande variance permettant d'appliquer des algorithmes de machine learning pour réaliser certaines tâches, le tout en réduisant grandement la quantité de données à traiter. Cela permet de gagner en temps de calcul ainsi qu'en quantité mémoire.

Voici les résultats obtenus :

Table 1: Résultats de la régression logistique et de la classification naïve bayésienne

	Classification naïve bayésienne		Régression logistique	
	accuracy_train	accuracy_test	accuracy_train	accuracy_test
Sans PCA	58,7%	57,7%	64,0%	58,7%
Avec PCA	58,1%	59,2%	62,6%	62,6%

3.2 Deep Learning: MLP

3.2.1 Question 1

Le tenseur d'entrée est de taille $32 \times 32 = 1024$.

Le tenseur de sortie est de taille 3 car il y a 3 classes différentes dans notre dataset.

3.2.2 Question 2

Nous utilisons 10 époques. Le nombre d'époque correspond au nombre de passage dans le dataset d'entraînement au cours de l'exécution du modèle.

Le `batch_size` correspond au nombre d'échantillons du dataset d'entrée utilisé pendant chaque phase de calcul des coefficients. Par exemple, pour un dataset d'entrainement de taille $18000 \times 0.8 = 1440$, si le `batch_size` vaut 100, chaque passage dans les données d'entraînement se fera par bloc de 100 échantillons. Les 13 premiers passages utiliseront 100 données et le quatorzième passage en utilisera 40.

3.2.3 Question 3

Le dataset de validation permet d'avoir une mesure de l'overfitting du modèle.

3.2.4 Question 4

Il y a plusieurs paramètres sur lesquels nous pouvons jouer pour modifier l'apprentissage de notre modèle.

Le premier est l'assemblage de couches utilisé dans notre modèle. Dans un premier temps nous avons utilisé l'assemblage suivant :

```
model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=(32, 32)),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(3, activation='Softmax')
])
```

Cela permet de définir un modèle avec 3 couches :

- Une couche "plate" d'entrée.
- Une couche dense de 128 paramètres avec le mode "relu". Cela permet d'introduire la non-linéarité nécessaire au bon fonctionnement du modèle.
- Une couche "plate" de sortie de taille 3, comme vu question 3.2.1.

Aussi, il est possible de modifier le taux d'apprentissage de l'optimiseur. Ici, nous avons utilisé l'optimiseur Adam, avec un taux d'apprentissage de $0,001$, c'est ce que permet la ligne `optimizer=tf.keras.optimizers.Adam(0.001)`. Le taux d'apprentissage permet de régler la vitesse d'apprentissage du modèle. Cependant, un taux d'apprentissage trop élevé peut rendre le modèle instable.

Finalement, nous pouvons modifier le nombre d'époques et le `batch_size` pour améliorer les performances du modèle.

3.2.5 Question 5

Voici ci-dessous les résultats que nous obtenons avec ce modèle initial :

Dans un premier temps, nous pouvons remarquer une chose simple : notre modèle apprend. En effet, plus le nombre d'époques augmente, i.e. plus notre modèle balaie les données d'entraînements, plus son taux de réussite est élevé. On remarque également que, globalement, la fonction de loss du modèle diminue avec le nombre d'époque, ce qui montre également que notre modèle apprend.

Nous remarquons qu'après de 10 époques, le taux de réussite du modèle sur le dataset de test est supérieur à 70%. On peut également remarquer que ce taux de réussite est obtenu au bout de 8 époques, et qu'il diminue sur les époques 9 et 10.

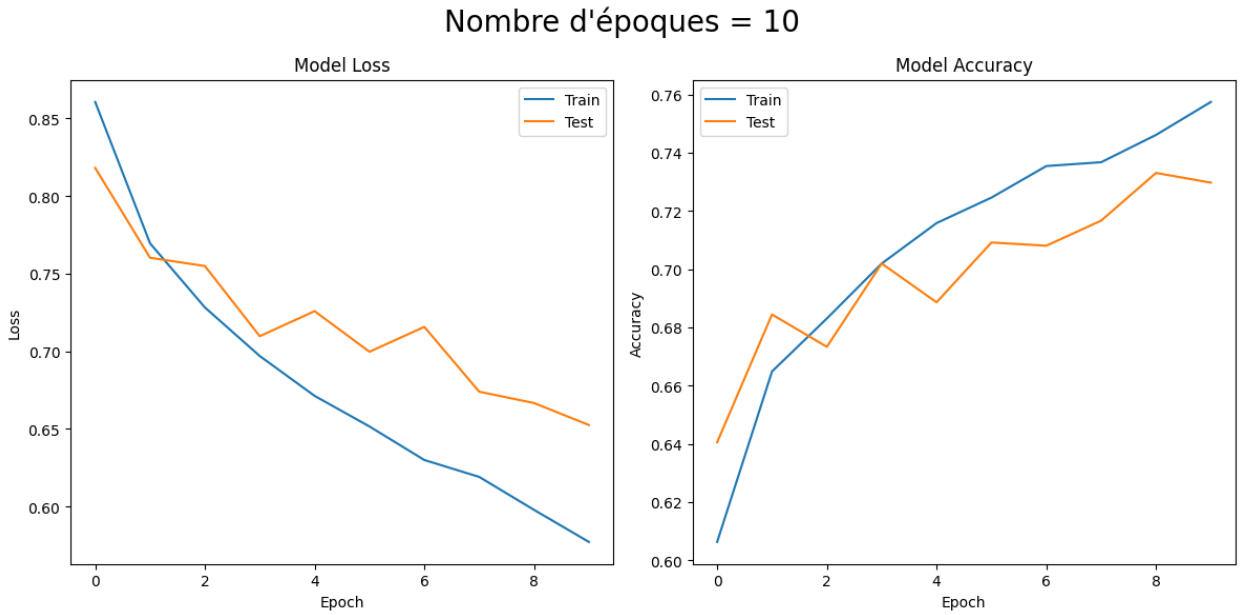

Figure 4: Model Loss et Model Accuracy de notre modèle MLP initial

3.2.6 Question 6

Nous remarquons que nous avons un très grand écart entre l'apprentissage de notre modèle sur le dataset d'entraînement et eclui sur le dataset de test. Cela montre que nous sommes en présence d'overfitting de notre modèle. Celui-ci apprend "par cœur" les paternes permettant de distinguer les trois classes sur le dataset d'entraînement mais n'est pas capable de généraliser afin d'apprendre sur de nouvelles données.

3.2.7 Question 7

Pour obtenir ces nouveaux résultats, nous avons modifier l'archtecture de notre réseau de neurones MLP. Nous l'avons construit de manière suivante :

```
model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=(32, 32)),
    tf.keras.layers.Dense(256, activation='relu'),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(3, activation='Softmax')
])
```

La couche de dropout permet de prévenir l'overfitting en éteignant de manière aléatoire 20% des neuronnes de la couche qui la précède.

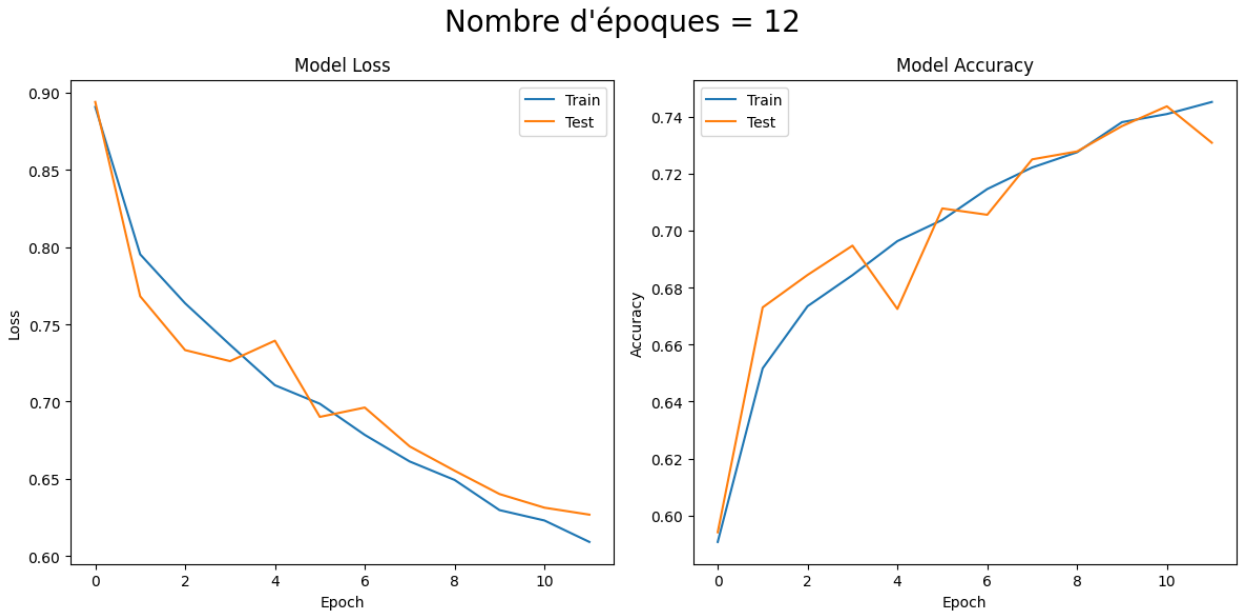

Figure 5: Model Loss et Model Accuracy de notre modèle MLP modifié

Nous pouvons voir que l'écart entre nos deux courbes est plus faible, ce qui signifie que nous avons réduit l'overfitting de notre modèle.

3.3 Deep Learning: CNN

3.3.1 Question 1

Ici, nous utilisons le dataset CIFAR-3, soit les images en couleurs. La taille du tenseur d'entrée est donc $(32, 32, 3)$.

3.3.2 Question 2

Voici dans un premier temps l'architecture que nous utilisons pour notre modèle :

```
model = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)),
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(3, activation='softmax')
])
```

Voici ci-dessous les résultats que nous obtenons :

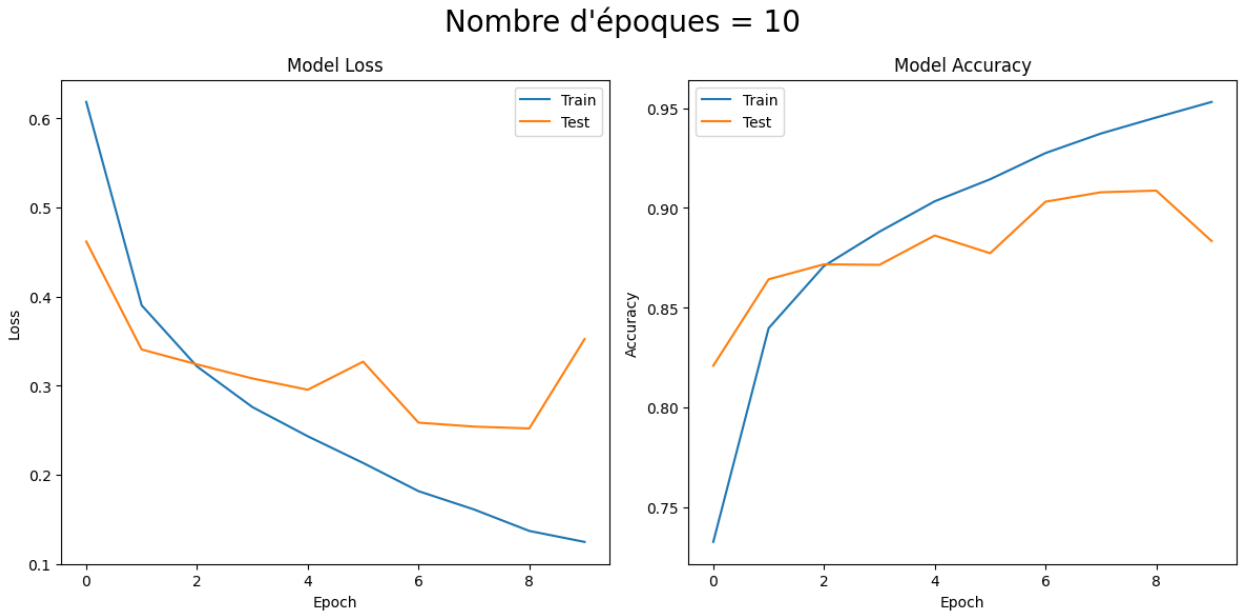

Figure 6: Model Loss et Model Accuracy de notre modèle CNN initial

Premièrement, nous pouvons remarquer que les résultats de ce nouveau modèle sont bien meilleurs que ceux du modèle MLP. On remarque un taux de réussite supérieur à 85% sur le dataset de test à la fin du modèle, ce qui est bien meilleur que précédemment.

Nous pouvons cependant observer un fort écart entre la fonction de loss de notre modèle sur le dataset d'entraînement et celui de test. De plus, le pourcentage de réussite de notre modèle baisse avec le nombre d'époques sur le dataset de test. Tous ces signes montrent un overfitting du modèle au dataset d'entraînement. Il est donc possible d'améliorer encore ce modèle.

3.3.3 Question 3

Comme expliqué à la question précédente, on remarque une forte présence d'overfitting dans notre modèle actuel.

3.3.4 Question 4

Pour améliorer le modèle, nous avons modifié plusieurs points du modèle:

- Nous avons rajouté des couches de dropouts entre les couches denses de notre modèle. Ce dropout a pour objectif de régulariser le modèle, et de diminuer l'overfitting.
- Nous avons modifié le taux d'apprentissage du modèle : nous sommes passés d'une valeur de $0,001$ à $0,0005$.
- Nous avons modifié le nombre d'époques en pasant de 10 à 12.

Voici l'architecture de notre modèle ainsi modifié :

```
model = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)),
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dropout(0.7),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dropout(0.7),
    tf.keras.layers.Dense(3, activation='softmax')
```

])

Le placement des couches de dropout nous a été suggéré par ChatGPT, c'est l'emplacement qui permettait de diminuer au mieux l'overfitting.

De plus, il a fallu mettre un très fort dropout pour réduire l'overfitting, c'est pourquoi nous éteignons 70% des neurones lors des différentes couches de dropout.

Nous avons essayé d'augmenter le nombre d'époque, cependant cela a créé encore plus d'overfitting à notre modèle. Voici ci-dessous un essai avec 20 époques.

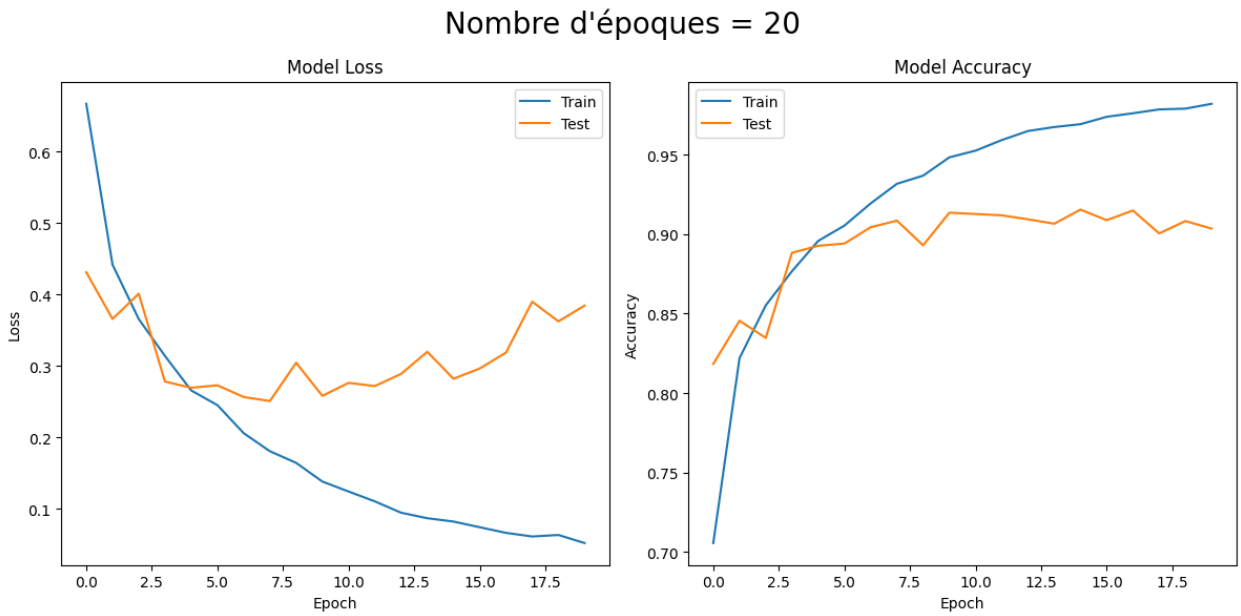

Figure 7: Model Loss et Model Accuracy de notre modèle CNN avec 20 époques

Après modification de la structure de notre modèle ainsi que des hyper-paramètres, voici ci-dessous les meilleurs résultats que nous avons obtenus. La structure finale anisi que les valeurs des hyper-paramètres sont ceux présents dans le fichier Python. Voici ci-dessous les résultats obtenus :

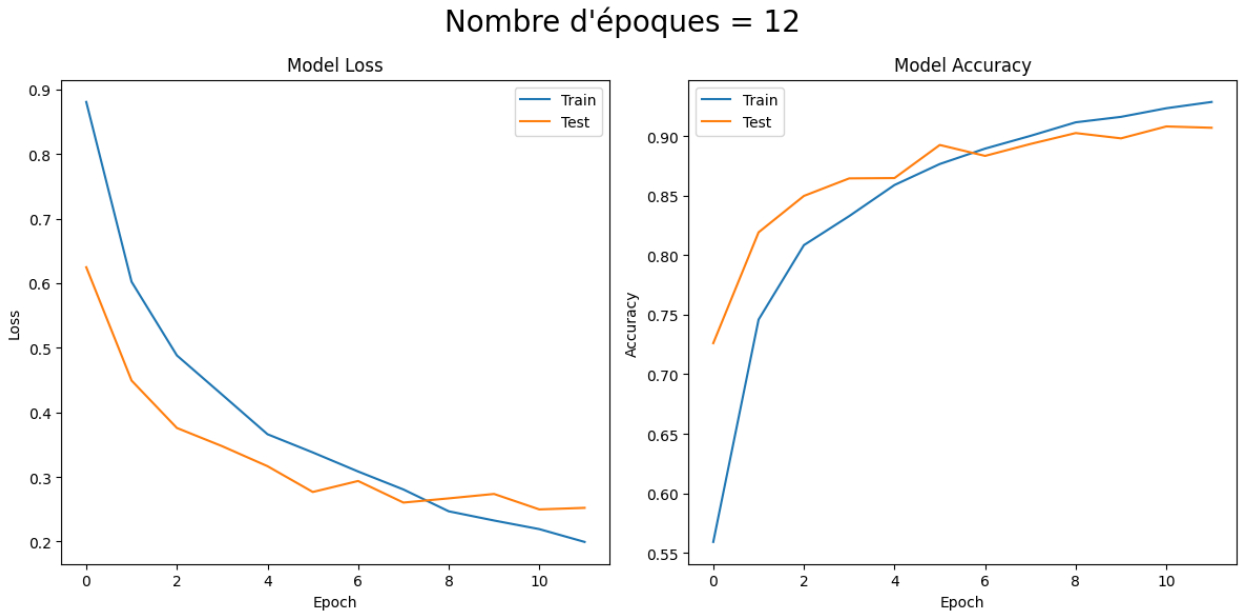

Figure 8: Model Loss et Model Accuracy de notre modèle CNN amélioré

3.3.5 Question 5

Nous avons eu de meilleurs résultats avec le modèle CNN dès la première utilisation de celui-ci, sans l'avoir amélioré. Le taux de réussite du modèle était supérieur à 85%, tandis que nous avons eu des difficultés à dépasser un score de 72% avec le modèle MLP.

Il a cependant été plus compliqué d'optimiser celui-ci et de réduire l'overfitting.