Tout d'abord, nous avons importé les bibliothèques nécessaires à la construction et à l'entraînement d'un modèle de réseau de neurones convolutif (CNN), telles que Keras, Tensorflow et Matplotlib.

Ensuite, nous avons préparé notre ensemble de données en divisant les images en ensembles d'entraînement, de validation et de test. Nous avons également appliqué des techniques de prétraitement d'images telles que la normalisation des pixels et l'augmentation de données.

Ensuite, nous avons construit notre modèle CNN en ajoutant plusieurs couches convolutives et de pooling, suivies de couches de régularisation et de densité. Nous avons également compilé le modèle avec une fonction de perte et un optimiseur.

Nous avons ensuite entraîné le modèle en utilisant les ensembles d'entraînement et de validation et visualisé les résultats à l'aide de graphiques de précision et de perte pour les deux ensembles.

Finalement, nous avons utilisé le modèle pour prédire les classes d'images dans l'ensemble de test et avons évalué sa précision.

En ce qui concerne la méthode CNN, elle est particulièrement utile pour la reconnaissance d'images car elle est capable de détecter automatiquement des motifs et des caractéristiques dans les images à partir de filtres de convolution. Cette méthode est largement utilisée dans la vision par ordinateur pour la classification, la segmentation et la détection d'objets, entre autres applications. Les CNN ont également la capacité d'apprendre des caractéristiques à différents niveaux d'abstraction, ce qui les rend particulièrement efficaces pour la reconnaissance d'images.

Si on reprends point par point j’ai fait :

1 Importation des bibliothèques :

Tout d'abord, nous devons importer les bibliothèques nécessaires pour créer notre CNN. Les principales bibliothèques à importer sont Keras et TensorFlow.

2. Charger les données :

Nous avons déjà chargé les données dans le tutoriel précédent, mais pour rappel, nous utilisons l'objet ImageDataGenerator pour charger les données à partir du dossier.

3. Créer le modèle :

Maintenant que nous avons chargé les données, nous pouvons créer notre modèle CNN. Le modèle de base consiste en une série de couches convolutives et de pooling, suivies de couches entièrement connectées. Nous utiliserons le modèle de base de Keras, appelé Sequential, qui nous permet de créer un modèle en ajoutant des couches les unes après les autres.

4. Compiler le modèle :

Une fois que nous avons créé notre modèle, nous devons le compiler en spécifiant la fonction de perte, l'optimiseur et les métriques à utiliser. Dans ce cas, nous utiliserons la fonction de perte binaire de Keras, l'optimiseur Adam et la métrique accuracy pour évaluer la performance du modèle.

5. Entraîner le modèle :

Nous pouvons maintenant entraîner notre modèle en utilisant la fonction fit() de Keras. Nous spécifierons le générateur de données d'entraînement, le générateur de données de validation, le nombre d'époques et la taille du lot.

Entraîner le modèle :

Maintenant que nous avons notre modèle CNN, nous pouvons l'entraîner sur les données que nous avons préparées.

Nous allons utiliser l'algorithme d'optimisation Adam, qui est un algorithme d'optimisation stochastique basé sur le gradient, très efficace pour l'entraînement des réseaux de neurones.

Nous allons également utiliser la fonction de perte binaire binary\_crossentropy, qui est utilisée pour les problèmes de classification binaire.

Enfin, nous allons utiliser la métrique accuracy pour évaluer les performances du modèle.

Pour cela, nous allons compiler notre modèle avec les paramètres suivants :

ici, nous utilisons simplement la méthode evaluate sur test\_generator. Cette méthode renvoie deux valeurs : la perte (loss) et la précision (accuracy) du modèle sur l'ensemble de données de test. Nous imprimons ces deux valeurs avec un formatage approprié.

Ensuite, vous pouvez visualiser les performances d'apprentissage de votre modèle en traçant les courbes d'apprentissage :

Ici, nous traçons les courbes d'apprentissage pour l'exactitude (accuracy) et la perte (loss) à la fois pour l'ensemble d'entraînement et l'ensemble de validation. Nous utilisons la bibliothèque matplotlib pour dessiner les graphiques.

Cela devrait vous donner une idée de la façon dont votre modèle apprend et de son aptitude à généraliser à des données inédites.

Le premier graphique montre comment la précision du modèle varie au fil des époques pour les données d'entraînement et de validation. L'axe horizontal représente le nombre d'époques d'entraînement, tandis que l'axe vertical représente la précision (ou l'exactitude) du modèle. La courbe bleue montre la précision sur les données d'entraînement, et la courbe orange montre la précision sur les données de validation. L'objectif est d'avoir une précision élevée à la fois sur les données d'entraînement et de validation, ce qui signifie que le modèle peut généraliser efficacement à de nouvelles données.

Le deuxième graphique montre comment la perte (ou l'erreur) du modèle varie au fil des époques pour les données d'entraînement et de validation. L'axe horizontal représente le nombre d'époques d'entraînement, tandis que l'axe vertical représente la perte du modèle. La courbe bleue montre la perte sur les données d'entraînement, et la courbe orange montre la perte sur les données de validation. L'objectif est d'avoir une perte faible à la fois sur les données d'entraînement et de validation, ce qui signifie que le modèle peut prédire les étiquettes correctes avec une grande précision.