

# cnn-model

March 20, 2024

1. Préparation des Données et Effectuer le prétraitement: Données de CIFAR-10

```
[17]: from tensorflow.keras.datasets import cifar10

# Charger les données CIFAR-10
(train_images, train_labels), (test_images, test_labels) = cifar10.load_data()

# Normaliser les images en les mettant à l'échelle entre 0 et 1
train_images = train_images.astype('float32') / 255.0
test_images = test_images.astype('float32') / 255.0
```

2. Création du Modèle : Création de notre propre CNN model

```
[18]: from tensorflow.keras import layers, models

# Créer le modèle CNN
def create_cnn_model(input_shape, num_classes):
    model = models.Sequential()

    # Première couche de convolution
    model.add(layers.Conv2D(32, (3, 3), activation='relu',
        ↪input_shape=input_shape))
    model.add(layers.MaxPooling2D((2, 2)))

    # Deuxième couche de convolution
    model.add(layers.Conv2D(64, (3, 3), activation='relu'))
    model.add(layers.MaxPooling2D((2, 2)))

    # Troisième couche de convolution
    model.add(layers.Conv2D(64, (3, 3), activation='relu'))

    # Applatissement des données pour les couches entièrement connectées
    model.add(layers.Flatten())

    # Première couche entièrement connectée
    model.add(layers.Dense(64, activation='relu'))

    # Couche de sortie
    model.add(layers.Dense(num_classes, activation='softmax'))
```

```

return model

# Définition des dimensions de l'entrée et du nombre de classes
input_shape = train_images.shape[1:]
num_classes = 10

# Création du modèle
model = create_cnn_model(input_shape, num_classes)

# Affichage de la structure du modèle
model.summary()

```

Model: "sequential\_3"

Layer (type)	Output Shape	Param #
conv2d_9 (Conv2D)	(None, 30, 30, 32)	896
max_pooling2d_6 (MaxPooling2D)	(None, 15, 15, 32)	0
conv2d_10 (Conv2D)	(None, 13, 13, 64)	18496
max_pooling2d_7 (MaxPooling2D)	(None, 6, 6, 64)	0
conv2d_11 (Conv2D)	(None, 4, 4, 64)	36928
flatten_3 (Flatten)	(None, 1024)	0
dense_6 (Dense)	(None, 64)	65600
dense_7 (Dense)	(None, 10)	650
Total params: 122570 (478.79 KB)		
Trainable params: 122570 (478.79 KB)		
Non-trainable params: 0 (0.00 Byte)		

### 3. Entraînement du Modèle :

```

[19]: from tensorflow.keras.utils import to_categorical
      from tensorflow.keras.optimizers import Adam

      # Prétraitement des étiquettes
      train_labels = to_categorical(train_labels, num_classes)

```

```
test_labels = to_categorical(test_labels, num_classes)
```

```
# Compilation du modèle
```

```
model.compile(optimizer=Adam(lr=0.001),  
              loss='categorical_crossentropy',  
              metrics=['accuracy'])
```

```
# Entraînement du modèle
```

```
history = model.fit(train_images, train_labels,  
                    epochs=10,  
                    batch_size=64,  
                    validation_split=0.1)
```

WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learning\_rate` or use the legacy optimizer, e.g.,`tf.keras.optimizers.legacy.Adam`.

Epoch 1/10

704/704 [=====] - 63s 88ms/step - loss: 1.5816 -  
accuracy: 0.4226 - val\_loss: 1.3058 - val\_accuracy: 0.5380

Epoch 2/10

704/704 [=====] - 51s 72ms/step - loss: 1.2429 -  
accuracy: 0.5620 - val\_loss: 1.1355 - val\_accuracy: 0.6028

Epoch 3/10

704/704 [=====] - 51s 72ms/step - loss: 1.0925 -  
accuracy: 0.6158 - val\_loss: 1.0668 - val\_accuracy: 0.6312

Epoch 4/10

704/704 [=====] - 51s 72ms/step - loss: 0.9844 -  
accuracy: 0.6556 - val\_loss: 0.9550 - val\_accuracy: 0.6736

Epoch 5/10

704/704 [=====] - 49s 70ms/step - loss: 0.9144 -  
accuracy: 0.6798 - val\_loss: 0.9565 - val\_accuracy: 0.6708

Epoch 6/10

704/704 [=====] - 51s 72ms/step - loss: 0.8569 -  
accuracy: 0.7012 - val\_loss: 0.8811 - val\_accuracy: 0.7020

Epoch 7/10

704/704 [=====] - 50s 70ms/step - loss: 0.8085 -  
accuracy: 0.7164 - val\_loss: 0.8482 - val\_accuracy: 0.7064

Epoch 8/10

704/704 [=====] - 50s 72ms/step - loss: 0.7648 -  
accuracy: 0.7330 - val\_loss: 0.8440 - val\_accuracy: 0.7136

Epoch 9/10

704/704 [=====] - 49s 70ms/step - loss: 0.7222 -  
accuracy: 0.7484 - val\_loss: 0.9022 - val\_accuracy: 0.6988

Epoch 10/10

704/704 [=====] - 49s 70ms/step - loss: 0.6841 -  
accuracy: 0.7606 - val\_loss: 0.8191 - val\_accuracy: 0.7164

4. Affichage des Courbes d'Apprentissage :

```
[20]: import matplotlib.pyplot as plt

# Extraire les données d'entraînement et de validation de l'historique
train_loss = history.history['loss']
train_accuracy = history.history['accuracy']
val_loss = history.history['val_loss']
val_accuracy = history.history['val_accuracy']
epochs = range(1, len(train_loss) + 1)

# Tracer la perte d'entraînement et de validation
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(epochs, train_loss, 'b-', label='Training Loss')
plt.plot(epochs, val_loss, 'r-', label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

# Tracer la précision d'entraînement et de validation
plt.subplot(1, 2, 2)
plt.plot(epochs, train_accuracy, 'b-', label='Training Accuracy')
plt.plot(epochs, val_accuracy, 'r-', label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

plt.tight_layout()
plt.show()
```



5. Prédiction : L'ensemble de données CIFAR-10 permet de classer les types d'objets suivants : Avion , Automobile , Oiseau , Chat , Cerf , Chien , Grenouille , Cheval , Navire , Camion

```
[22]: import numpy as np
from tensorflow.keras.preprocessing import image
import matplotlib.pyplot as plt

# Chemin de l'image à tester
image_path = "/content/horse.jpg"
img1 = image.load_img(image_path)
# Charger l'image et la prétraiter
img = image.load_img(image_path, target_size=(32, 32))
img_array = image.img_to_array(img) / 255.0
img_array = np.expand_dims(img_array, axis=0)

# Définition des classes CIFAR-10
class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer',
               'dog', 'frog', 'horse', 'ship', 'truck']

# Faire la prédiction avec le modèle
prediction = model.predict(img_array)
predicted_class = np.argmax(prediction)
class_name = class_names[predicted_class]

# Afficher l'image avec sa prédiction
plt.imshow(img1)
plt.title(f"Predicted Class: {class_name}")
plt.axis('off')
plt.show()
```

1/1 [=====] - 0s 64ms/step

Predicted Class: horse



#### 6. Évaluation du Modèle :

```
[23]: from sklearn.metrics import accuracy_score, f1_score

# Faire des prédictions sur l'ensemble de test
y_pred = model.predict(test_images)
y_pred_classes = np.argmax(y_pred, axis=1)

# Convertir les étiquettes de l'ensemble de test en classes prédites
y_true_classes = np.argmax(test_labels, axis=1)

# Calculer l'exactitude (accuracy) et le score F1
accuracy = accuracy_score(y_true_classes, y_pred_classes)
f1 = f1_score(y_true_classes, y_pred_classes, average='weighted')

print("Exactitude (Accuracy) :", accuracy)
print("Score F1 :", f1)
```

```
313/313 [=====] - 7s 23ms/step
Exactitude (Accuracy) : 0.7041
Score F1 : 0.7021114885038244
```