Modele3_cnn

November 19, 2025

[55]:
```python
import sys
import numpy as np
import matplotlib.pyplot as plt

np.random.seed(0)
```

[56]:
```python
import tensorflow as tf
print(tf.__version__)
```

```
2.20.0
```

[57]:
```python
import pickle
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.utils.class_weight import compute_class_weight

import tensorflow as tf
from keras import layers, models
from keras.callbacks import EarlyStopping, ReduceLROnPlateau


# -----------------------------
# Load Data
# -----------------------------
path_to_data = "ift-3395-6390-kaggle-2-competition-fall-2025/train_data.pkl"

with open(path_to_data, "rb") as f:
    train_data = pickle.load(f)



X = train_data["images"].astype("float32") / 255.0
y = train_data["labels"].reshape(-1)

X = X.astype(np.float32)
X = (X - X.min()) / (X.max() - X.min() + 1e-6)

X = (X - X.mean()) / (X.std() + 1e-6)
```

```python
# ------------------------------
# Train/Val Split
# ------------------------------
X_train, X_val, y_train, y_val = train_test_split(
    X, y, test_size=0.2, stratify=y, random_state=42
)

# One-hot encoding
y_train_oh = tf.keras.utils.to_categorical(y_train, num_classes=5)
y_val_oh   = tf.keras.utils.to_categorical(y_val, num_classes=5)


# ------------------------------
# Model Definition
# ------------------------------
def create_cnn_model(num_classes=5):
    model = models.Sequential([
        layers.Input(shape=(28, 28, 3)),

        layers.Conv2D(32, 3, activation='relu', padding='same'),
        layers.BatchNormalization(),
        layers.Conv2D(32, 3, activation='relu', padding='same'),
        layers.BatchNormalization(),
        layers.MaxPooling2D(),
        layers.Dropout(0.3),

        layers.Conv2D(64, 3, activation='relu', padding='same'),
        layers.BatchNormalization(),
        layers.Conv2D(64, 3, activation='relu', padding='same'),
        layers.BatchNormalization(),
        layers.MaxPooling2D(),
        layers.Dropout(0.3),

        layers.Conv2D(128, 3, activation='relu', padding='same'),
        layers.BatchNormalization(),
        layers.MaxPooling2D(),
        layers.Dropout(0.4),

        layers.Flatten(),
        layers.Dense(256, activation='relu'),
        layers.BatchNormalization(),
        layers.Dropout(0.5),

        layers.Dense(128, activation='relu'),
        layers.Dropout(0.4),
        layers.Dense(num_classes, activation='softmax')
    ])
    return model


model = create_cnn_model(5)


# -----------------------------
# Class Weights
# -----------------------------
classes = np.unique(y_train)
weights = compute_class_weight("balanced", classes=classes, y=y_train)
class_weights = dict(zip(classes, weights))

print("Class weights:", class_weights)


# -----------------------------
# Compile
# -----------------------------
model.compile(
    optimizer="adam",
    loss="categorical_crossentropy",
    metrics=[
        "accuracy",
        tf.keras.metrics.Recall(name="recall")
    ]
)


# -----------------------------
# TRAINING
# -----------------------------
history = model.fit(
    X_train, y_train_oh,
    validation_data=(X_val, y_val_oh),
    epochs=45,
    batch_size=32,
    class_weight=class_weights,
    callbacks=[
        EarlyStopping(
            monitor="balanced_accuracy",
            patience=6,
            restore_best_weights=True,
            mode="max"
```

```python
        ),
        ReduceLROnPlateau(
            monitor="balanced_accuracy",
            factor=0.5,
            patience=5,
            mode="max",
            min_lr=1e-6
        )
    ],
    shuffle=True
)


# -----------------------------
# Evaluation
# -----------------------------
from sklearn.metrics import classification_report, balanced_accuracy_score

y_pred = model.predict(X_val)
y_pred_classes = np.argmax(y_pred, axis=1)

print("Balanced accuracy:", balanced_accuracy_score(y_val, y_pred_classes))
print(classification_report(y_val, y_pred_classes))
```

```
Class weights: {np.uint8(0): np.float64(0.4442159383033419), np.uint8(1):
np.float64(1.6941176470588235), np.uint8(2): np.float64(1.0472727272727274),
np.uint8(3): np.float64(1.1148387096774193), np.uint8(4):
np.float64(3.260377358490566)}
Epoch 1/45
27/27                 2s 28ms/step -
accuracy: 0.2407 - loss: 2.4561 - recall: 0.1759 - val_accuracy: 0.1852 -
val_loss: 1.6543 - val_recall: 0.0000e+00 - learning_rate: 0.0010
Epoch 2/45
 7/27                 0s 21ms/step -
accuracy: 0.3387 - loss: 2.2077 - recall: 0.2526
```

```
/Users/yamira.poldosilva/Documents/UDEM/A25/IFT3395/kaggle2/kaggle2/lib/python3.
13/site-packages/keras/src/callbacks/early_stopping.py:99: UserWarning: Early
stopping conditioned on metric `balanced_accuracy` which is not available.
Available metrics are: accuracy,loss,recall,val_accuracy,val_loss,val_recall
  current = self.get_monitor_value(logs)
/Users/yamira.poldosilva/Documents/UDEM/A25/IFT3395/kaggle2/kaggle2/lib/python3.
13/site-packages/keras/src/callbacks/callback_list.py:171: UserWarning: Learning
rate reduction is conditioned on metric `balanced_accuracy` which is not
available. Available metrics are:
accuracy,loss,recall,val_accuracy,val_loss,val_recall,learning_rate.
  callback.on_epoch_end(epoch, logs)
```

```
27/27                 1s 27ms/step -
```

```
accuracy: 0.2616 - loss: 2.2682 - recall: 0.1944 - val_accuracy: 0.1157 -
val_loss: 1.7381 - val_recall: 0.0000e+00 - learning_rate: 0.0010
Epoch 3/45
27/27            1s 34ms/step -
accuracy: 0.2859 - loss: 2.1275 - recall: 0.1968 - val_accuracy: 0.2176 -
val_loss: 1.7172 - val_recall: 0.0139 - learning_rate: 0.0010
Epoch 4/45
27/27            1s 39ms/step -
accuracy: 0.3229 - loss: 1.9205 - recall: 0.2095 - val_accuracy: 0.2639 -
val_loss: 1.6392 - val_recall: 0.1389 - learning_rate: 0.0010
Epoch 5/45
27/27            1s 36ms/step -
accuracy: 0.3102 - loss: 1.8863 - recall: 0.2037 - val_accuracy: 0.2685 -
val_loss: 1.6630 - val_recall: 0.1574 - learning_rate: 0.0010
Epoch 6/45
27/27            1s 34ms/step -
accuracy: 0.3194 - loss: 1.7643 - recall: 0.1910 - val_accuracy: 0.2500 -
val_loss: 1.6596 - val_recall: 0.1111 - learning_rate: 0.0010
Epoch 7/45
27/27            1s 33ms/step -
accuracy: 0.3067 - loss: 1.7193 - recall: 0.1736 - val_accuracy: 0.2731 -
val_loss: 1.5432 - val_recall: 0.0880 - learning_rate: 0.0010
Epoch 8/45
27/27            1s 33ms/step -
accuracy: 0.3160 - loss: 1.7536 - recall: 0.1725 - val_accuracy: 0.3565 -
val_loss: 1.4101 - val_recall: 0.1620 - learning_rate: 0.0010
Epoch 9/45
27/27            1s 33ms/step -
accuracy: 0.3356 - loss: 1.7106 - recall: 0.1910 - val_accuracy: 0.4167 -
val_loss: 1.3745 - val_recall: 0.1759 - learning_rate: 0.0010
Epoch 10/45
27/27            1s 34ms/step -
accuracy: 0.3206 - loss: 1.6874 - recall: 0.1655 - val_accuracy: 0.3009 -
val_loss: 1.5467 - val_recall: 0.0463 - learning_rate: 0.0010
Epoch 11/45
27/27            1s 32ms/step -
accuracy: 0.3426 - loss: 1.6436 - recall: 0.1690 - val_accuracy: 0.2917 -
val_loss: 1.4989 - val_recall: 0.1250 - learning_rate: 0.0010
Epoch 12/45
27/27            1s 32ms/step -
accuracy: 0.3715 - loss: 1.5748 - recall: 0.1956 - val_accuracy: 0.3056 -
val_loss: 1.4934 - val_recall: 0.1435 - learning_rate: 0.0010
Epoch 13/45
27/27            1s 33ms/step -
accuracy: 0.3600 - loss: 1.5674 - recall: 0.1921 - val_accuracy: 0.3148 -
val_loss: 1.5044 - val_recall: 0.1343 - learning_rate: 0.0010
Epoch 14/45
27/27            1s 34ms/step -
```

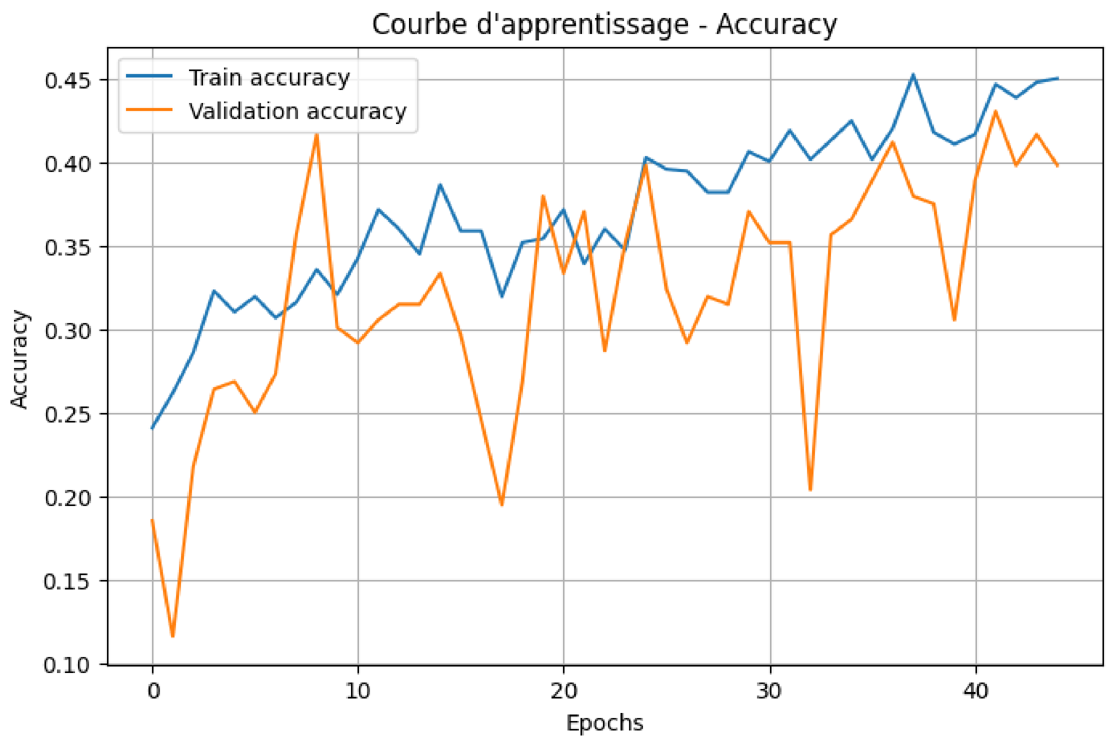
```
accuracy: 0.3449 - loss: 1.6296 - recall: 0.1748 - val_accuracy: 0.3148 -
val_loss: 1.4861 - val_recall: 0.1481 - learning_rate: 0.0010
Epoch 15/45
27/27            1s 32ms/step -
accuracy: 0.3866 - loss: 1.5552 - recall: 0.1944 - val_accuracy: 0.3333 -
val_loss: 1.4626 - val_recall: 0.1481 - learning_rate: 0.0010
Epoch 16/45
27/27            1s 33ms/step -
accuracy: 0.3588 - loss: 1.6120 - recall: 0.1979 - val_accuracy: 0.2963 -
val_loss: 1.5076 - val_recall: 0.1111 - learning_rate: 0.0010
Epoch 17/45
27/27            1s 33ms/step -
accuracy: 0.3588 - loss: 1.5717 - recall: 0.1875 - val_accuracy: 0.2454 -
val_loss: 1.6131 - val_recall: 0.0833 - learning_rate: 0.0010
Epoch 18/45
27/27            1s 32ms/step -
accuracy: 0.3194 - loss: 1.5868 - recall: 0.1586 - val_accuracy: 0.1944 -
val_loss: 1.6989 - val_recall: 0.0000e+00 - learning_rate: 0.0010
Epoch 19/45
27/27            1s 33ms/step -
accuracy: 0.3519 - loss: 1.6123 - recall: 0.1562 - val_accuracy: 0.2685 -
val_loss: 1.5785 - val_recall: 0.0509 - learning_rate: 0.0010
Epoch 20/45
27/27            1s 33ms/step -
accuracy: 0.3542 - loss: 1.5709 - recall: 0.1574 - val_accuracy: 0.3796 -
val_loss: 1.4416 - val_recall: 0.1620 - learning_rate: 0.0010
Epoch 21/45
27/27            1s 33ms/step -
accuracy: 0.3715 - loss: 1.5491 - recall: 0.1898 - val_accuracy: 0.3333 -
val_loss: 1.4788 - val_recall: 0.1574 - learning_rate: 0.0010
Epoch 22/45
27/27            1s 34ms/step -
accuracy: 0.3391 - loss: 1.5110 - recall: 0.1736 - val_accuracy: 0.3704 -
val_loss: 1.4644 - val_recall: 0.1713 - learning_rate: 0.0010
Epoch 23/45
27/27            1s 33ms/step -
accuracy: 0.3600 - loss: 1.5190 - recall: 0.1921 - val_accuracy: 0.2870 -
val_loss: 1.5246 - val_recall: 0.1019 - learning_rate: 0.0010
Epoch 24/45
27/27            1s 33ms/step -
accuracy: 0.3472 - loss: 1.5495 - recall: 0.1644 - val_accuracy: 0.3519 -
val_loss: 1.4442 - val_recall: 0.1296 - learning_rate: 0.0010
Epoch 25/45
27/27            1s 33ms/step -
accuracy: 0.4028 - loss: 1.4547 - recall: 0.1875 - val_accuracy: 0.3981 -
val_loss: 1.4144 - val_recall: 0.1667 - learning_rate: 0.0010
Epoch 26/45
27/27            1s 33ms/step -
```

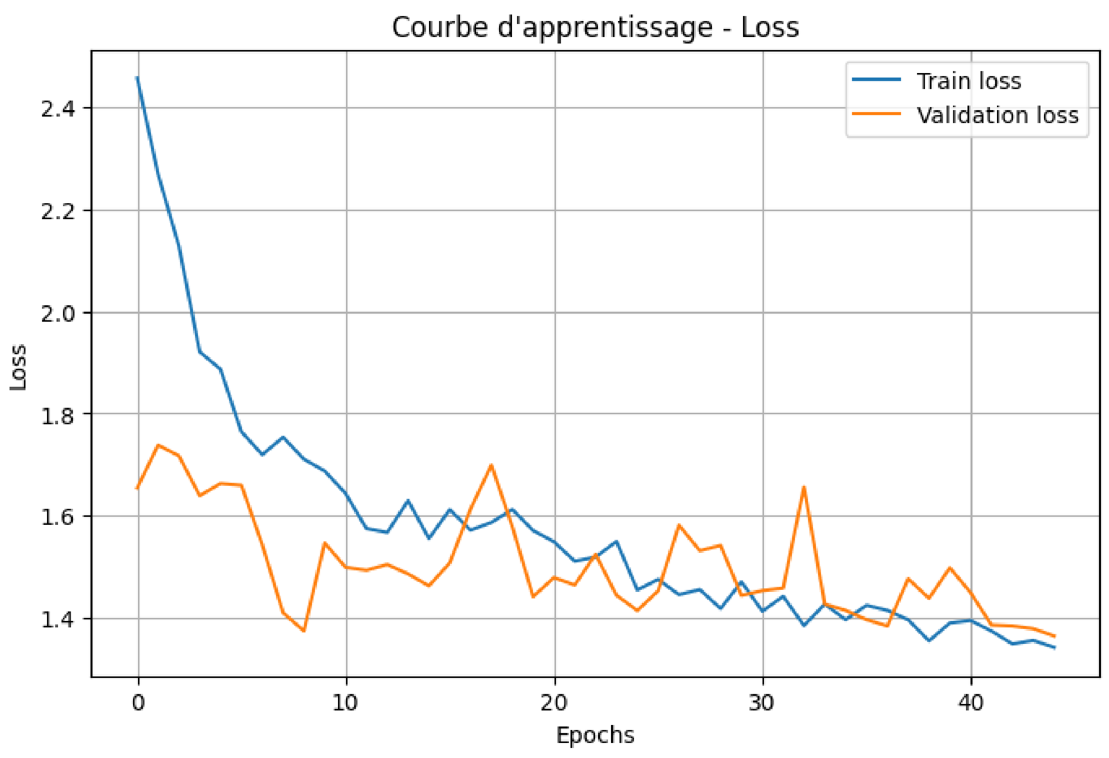
```
accuracy: 0.3958 - loss: 1.4754 - recall: 0.1944 - val_accuracy: 0.3241 -
val_loss: 1.4529 - val_recall: 0.1528 - learning_rate: 0.0010
Epoch 27/45
27/27           1s 33ms/step -
accuracy: 0.3947 - loss: 1.4454 - recall: 0.2118 - val_accuracy: 0.2917 -
val_loss: 1.5815 - val_recall: 0.0509 - learning_rate: 0.0010
Epoch 28/45
27/27           1s 33ms/step -
accuracy: 0.3819 - loss: 1.4555 - recall: 0.1921 - val_accuracy: 0.3194 -
val_loss: 1.5316 - val_recall: 0.0694 - learning_rate: 0.0010
Epoch 29/45
27/27           1s 33ms/step -
accuracy: 0.3819 - loss: 1.4183 - recall: 0.2269 - val_accuracy: 0.3148 -
val_loss: 1.5417 - val_recall: 0.1157 - learning_rate: 0.0010
Epoch 30/45
27/27           1s 33ms/step -
accuracy: 0.4062 - loss: 1.4704 - recall: 0.2280 - val_accuracy: 0.3704 -
val_loss: 1.4438 - val_recall: 0.1713 - learning_rate: 0.0010
Epoch 31/45
27/27           1s 33ms/step -
accuracy: 0.4005 - loss: 1.4131 - recall: 0.2222 - val_accuracy: 0.3519 -
val_loss: 1.4534 - val_recall: 0.1343 - learning_rate: 0.0010
Epoch 32/45
27/27           1s 34ms/step -
accuracy: 0.4190 - loss: 1.4422 - recall: 0.2234 - val_accuracy: 0.3519 -
val_loss: 1.4583 - val_recall: 0.1389 - learning_rate: 0.0010
Epoch 33/45
27/27           1s 33ms/step -
accuracy: 0.4016 - loss: 1.3850 - recall: 0.2037 - val_accuracy: 0.2037 -
val_loss: 1.6558 - val_recall: 0.0231 - learning_rate: 0.0010
Epoch 34/45
27/27           1s 34ms/step -
accuracy: 0.4132 - loss: 1.4270 - recall: 0.2245 - val_accuracy: 0.3565 -
val_loss: 1.4269 - val_recall: 0.2083 - learning_rate: 0.0010
Epoch 35/45
27/27           1s 33ms/step -
accuracy: 0.4248 - loss: 1.3965 - recall: 0.2153 - val_accuracy: 0.3657 -
val_loss: 1.4148 - val_recall: 0.2130 - learning_rate: 0.0010
Epoch 36/45
27/27           1s 33ms/step -
accuracy: 0.4016 - loss: 1.4245 - recall: 0.2338 - val_accuracy: 0.3889 -
val_loss: 1.3967 - val_recall: 0.2269 - learning_rate: 0.0010
Epoch 37/45
27/27           1s 33ms/step -
accuracy: 0.4201 - loss: 1.4148 - recall: 0.2245 - val_accuracy: 0.4120 -
val_loss: 1.3841 - val_recall: 0.2639 - learning_rate: 0.0010
Epoch 38/45
27/27           1s 33ms/step -
```

```
accuracy: 0.4525 - loss: 1.3964 - recall: 0.2396 - val_accuracy: 0.3796 -
val_loss: 1.4766 - val_recall: 0.2454 - learning_rate: 0.0010
Epoch 39/45
27/27               1s 33ms/step -
accuracy: 0.4178 - loss: 1.3550 - recall: 0.2535 - val_accuracy: 0.3750 -
val_loss: 1.4383 - val_recall: 0.1991 - learning_rate: 0.0010
Epoch 40/45
27/27               1s 33ms/step -
accuracy: 0.4109 - loss: 1.3900 - recall: 0.2176 - val_accuracy: 0.3056 -
val_loss: 1.4981 - val_recall: 0.1481 - learning_rate: 0.0010
Epoch 41/45
27/27               1s 33ms/step -
accuracy: 0.4167 - loss: 1.3949 - recall: 0.2245 - val_accuracy: 0.3889 -
val_loss: 1.4493 - val_recall: 0.2222 - learning_rate: 0.0010
Epoch 42/45
27/27               1s 33ms/step -
accuracy: 0.4468 - loss: 1.3742 - recall: 0.2315 - val_accuracy: 0.4306 -
val_loss: 1.3860 - val_recall: 0.2500 - learning_rate: 0.0010
Epoch 43/45
27/27               1s 33ms/step -
accuracy: 0.4387 - loss: 1.3490 - recall: 0.2407 - val_accuracy: 0.3981 -
val_loss: 1.3842 - val_recall: 0.2361 - learning_rate: 0.0010
Epoch 44/45
27/27               1s 37ms/step -
accuracy: 0.4479 - loss: 1.3559 - recall: 0.2454 - val_accuracy: 0.4167 -
val_loss: 1.3790 - val_recall: 0.2685 - learning_rate: 0.0010
Epoch 45/45
27/27               1s 35ms/step -
accuracy: 0.4502 - loss: 1.3424 - recall: 0.2488 - val_accuracy: 0.3981 -
val_loss: 1.3647 - val_recall: 0.2315 - learning_rate: 0.0010
7/7               0s 23ms/step
Balanced accuracy: 0.3257848010676776
              precision    recall  f1-score   support

           0       0.72      0.57      0.64        97
           1       0.29      0.38      0.33        26
           2       0.27      0.29      0.28        41
           3       0.33      0.15      0.21        39
           4       0.07      0.23      0.11        13

    accuracy                           0.40       216
   macro avg       0.34      0.33      0.31       216
weighted avg       0.48      0.40      0.42       216
```

[58]:
```python
import matplotlib.pyplot as plt
```

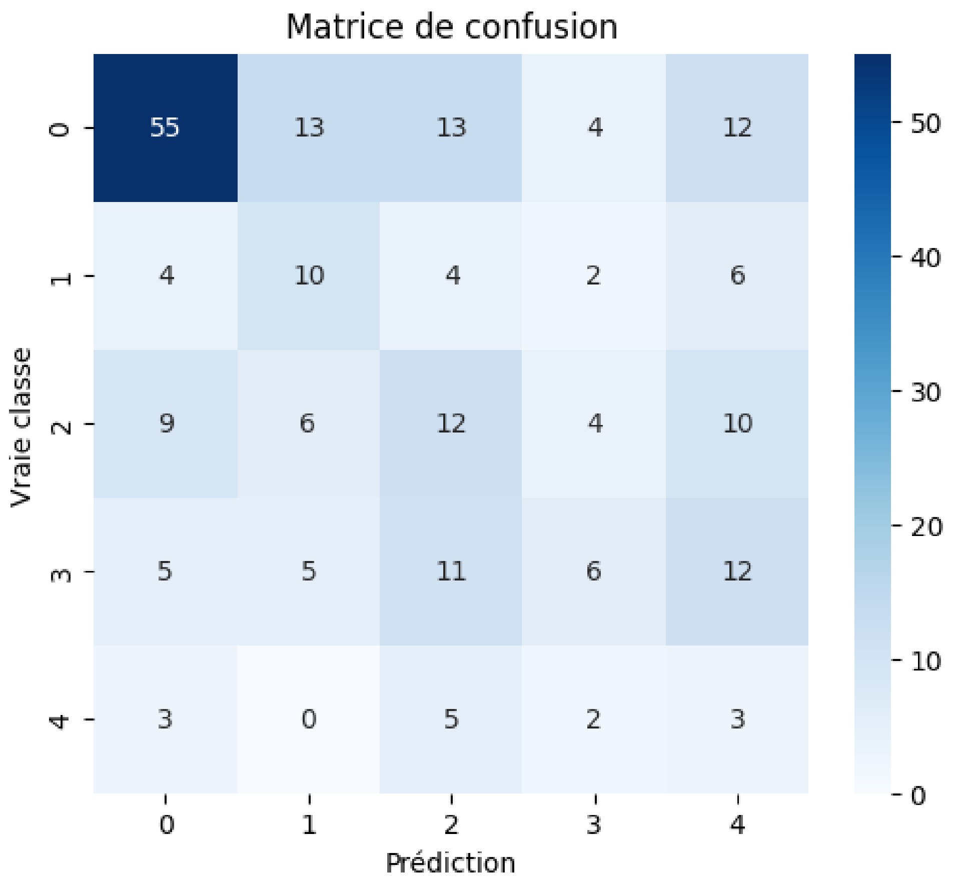
```python
# Accuracy
plt.figure(figsize=(8,5))
plt.plot(history.history["accuracy"], label="Train accuracy")
plt.plot(history.history["val_accuracy"], label="Validation accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.title("Courbe d'apprentissage - Accuracy")
plt.legend()
plt.grid(True)
plt.show()

# Loss
plt.figure(figsize=(8,5))
plt.plot(history.history["loss"], label="Train loss")
plt.plot(history.history["val_loss"], label="Validation loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.title("Courbe d'apprentissage - Loss")
plt.legend()
plt.grid(True)
plt.show()
```





```
[59]: y_val_pred = model.predict(X_val)
      y_val_pred_classes = y_val_pred.argmax(axis=1)
      from sklearn.metrics import confusion_matrix, accuracy_score
      import seaborn as sns

      cm = confusion_matrix(y_val, y_val_pred_classes)

      plt.figure(figsize=(6,5))
      sns.heatmap(cm, annot=True, cmap="Blues", fmt="d")
      plt.title("Matrice de confusion")
      plt.xlabel("Prédiction")
      plt.ylabel("Vraie classe")
      plt.show()
      from sklearn.metrics import recall_score

      recall_per_class = recall_score(y_val, y_val_pred_classes, average=None)
      recall_macro = recall_score(y_val, y_val_pred_classes, average="macro")
      acc = accuracy_score(y_val, y_val_pred_classes)
      print("Recall par classe :", recall_per_class)
      print('acc:', acc)
      print("Recall macro :", recall_macro)
```

```
7/7                0s 7ms/step
```



```
Recall par classe : [0.56701031 0.38461538 0.29268293 0.15384615 0.23076923]
acc: 0.39814814814814814
Recall macro : 0.3257848010676776
```

[60]:
```
import pickle
import numpy as np
import pandas as pd
"""
# ---------------------------
# 1. Charger le modèle entraîné
# ---------------------------
model, scaler = pickle.load(open("model_softmax.pkl", "rb"))

# ---------------------------
# 2. Charger le test_data.pkl
# ---------------------------
with open("ift-3395-6390-kaggle-2-competition-fall-2025/test_data.pkl", "rb")␣
  ↪as f:
    test_data = pickle.load(f)
```

```
X_test_imgs = test_data["images"]


# Apply to test set
X_test_feats = np.array([extract_features(img) for img in X_test_imgs],␣
 ↪dtype=np.float32)

# --------------------------
# 4. Normaliser avec les stats du train
# --------------------------
X_test_norm = scaler.transform(X_test_feats)

# --------------------------
# 5. Prédire
# --------------------------
y_pred = model.compute_output(X_test_norm).astype(int)

# --------------------------
# 6. Générer le CSV Kaggle
# --------------------------
df = pd.DataFrame({
    "ID": np.arange(1, len(y_pred)+1),
    "Label": y_pred
})

df.to_csv("ift3395_YamirPoldoSilvaV7.csv", index=False)

print("Fichier 'submission.csv' généré !")
"""
```

```
[60]: '\n# --------------------------\n# 1. Charger le modèle entraîné\n#
--------------------------\nmodel, scaler =
pickle.load(open("model_softmax.pkl", "rb"))\n\n# --------------------------\n#
2. Charger le test_data.pkl\n# --------------------------\nwith
open("ift-3395-6390-kaggle-2-competition-fall-2025/test_data.pkl", "rb") as f:\n
test_data = pickle.load(f)\n\nX_test_imgs = test_data["images"]\n\n\n# Apply to
test set\nX_test_feats = np.array([extract_features(img) for img in
X_test_imgs], dtype=np.float32)\n\n# --------------------------\n# 4. Normaliser
avec les stats du train\n# --------------------------\nX_test_norm =
scaler.transform(X_test_feats)\n\n# --------------------------\n# 5. Prédire\n#
--------------------------\ny_pred =
model.compute_output(X_test_norm).astype(int)\n\n# --------------------------\n#
6. Générer le CSV Kaggle\n# --------------------------\ndf = pd.DataFrame({\n
"ID": np.arange(1, len(y_pred)+1),\n    "Label":
y_pred\n})\n\ndf.to_csv("ift3395_YamirPoldoSilvaV7.csv",
index=False)\n\nprint("Fichier \'submission.csv\' généré !")\n'
```