Modele2_cnn

November 19, 2025

[1]:
```
import sys
import numpy as np
import matplotlib.pyplot as plt

np.random.seed(0)
```

[2]:
```
import tensorflow as tf
print(tf.__version__)
```

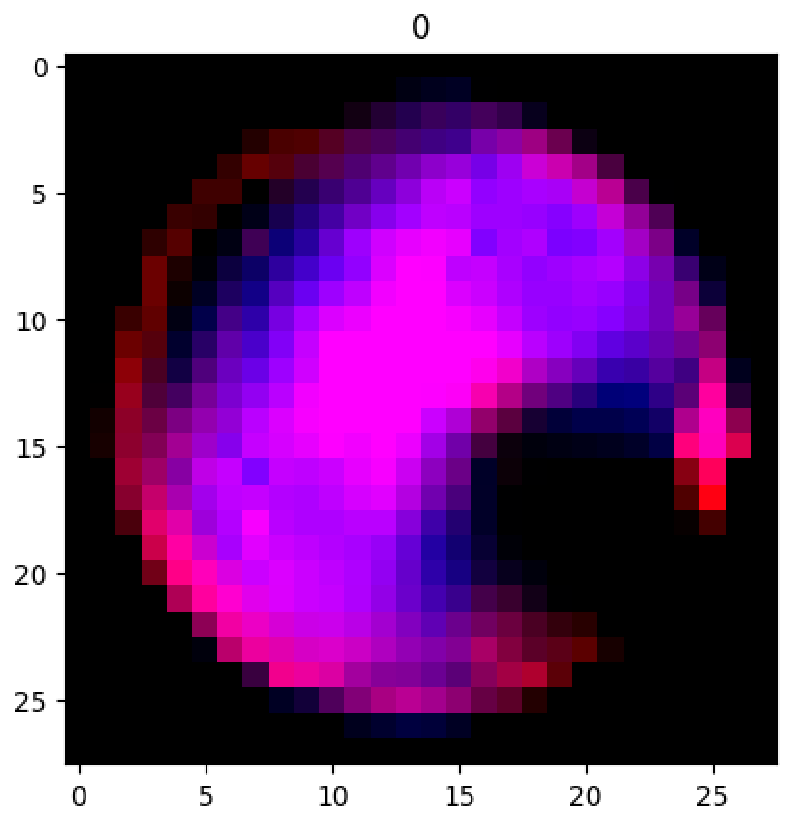
```
2.20.0
```

[3]:
```
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)
```

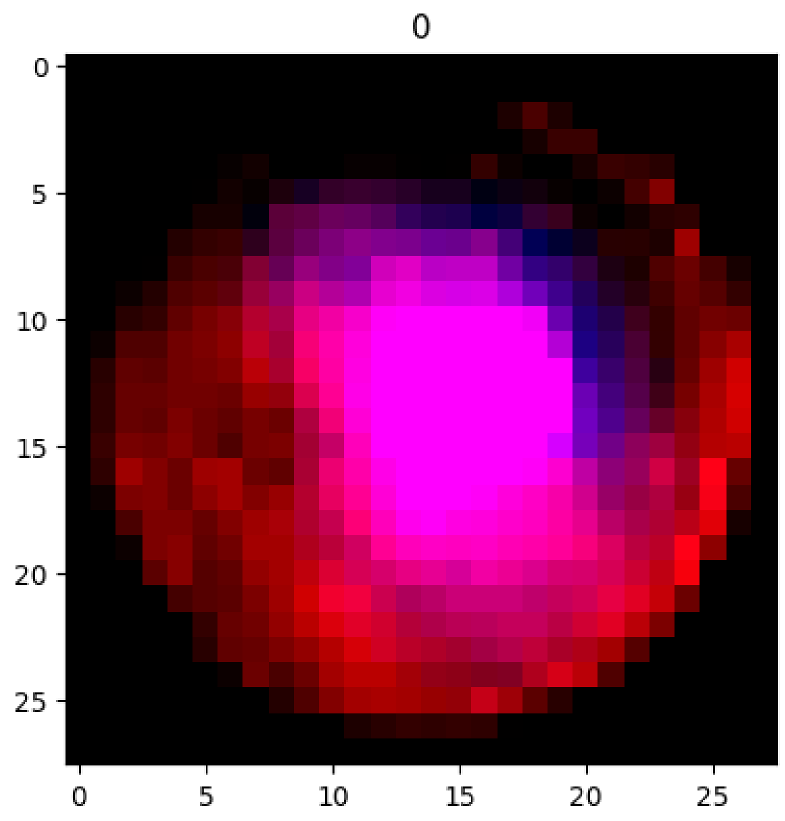
```
for i in range(10):
    plt.imshow(X[i])
    plt.title(y[i])
    plt.show()
# -----------------------------
# Train/Val Split
# -----------------------------
X_train, X_val, y_train, y_val = train_test_split(
    X, y, test_size=0.2, stratify=y, random_state=42
)

# 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)


# -----------------------------
# Data Augmentation block
# -----------------------------
data_augmentation = models.Sequential([
    layers.RandomFlip("horizontal"),
    layers.RandomRotation(0.1),
    layers.RandomZoom(0.1),
    layers.RandomTranslation(0.1, 0.1),
], name="data_augmentation")


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

        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=100,
    batch_size=32,
    #class_weight=class_weights,
    callbacks=[
        EarlyStopping(
            monitor="val_recall",
            patience=12,
            restore_best_weights=True,
            mode="max"
        ),
        ReduceLROnPlateau(
            monitor="val_recall",
            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))
```

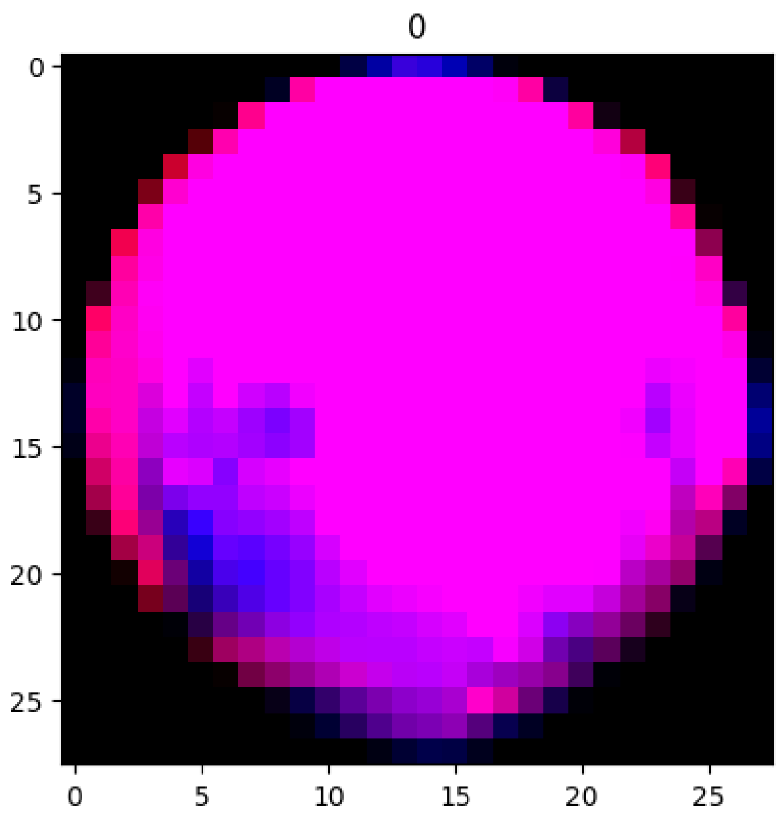
```
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers). Got range [-0.82880193..2.2015023].
```



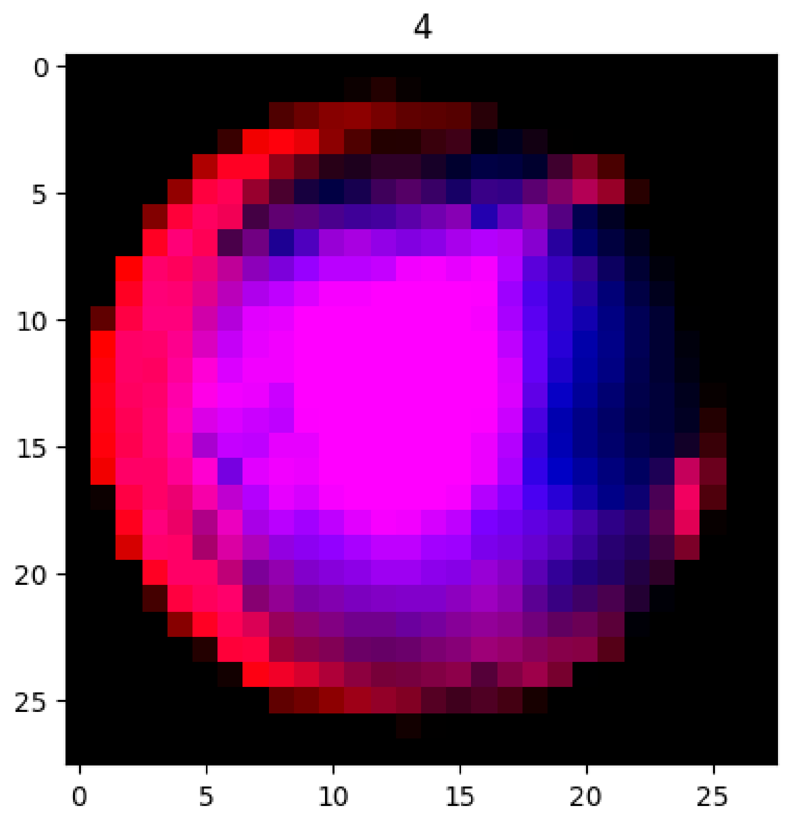
```
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers). Got range [-0.82880193..3.3653147].
```



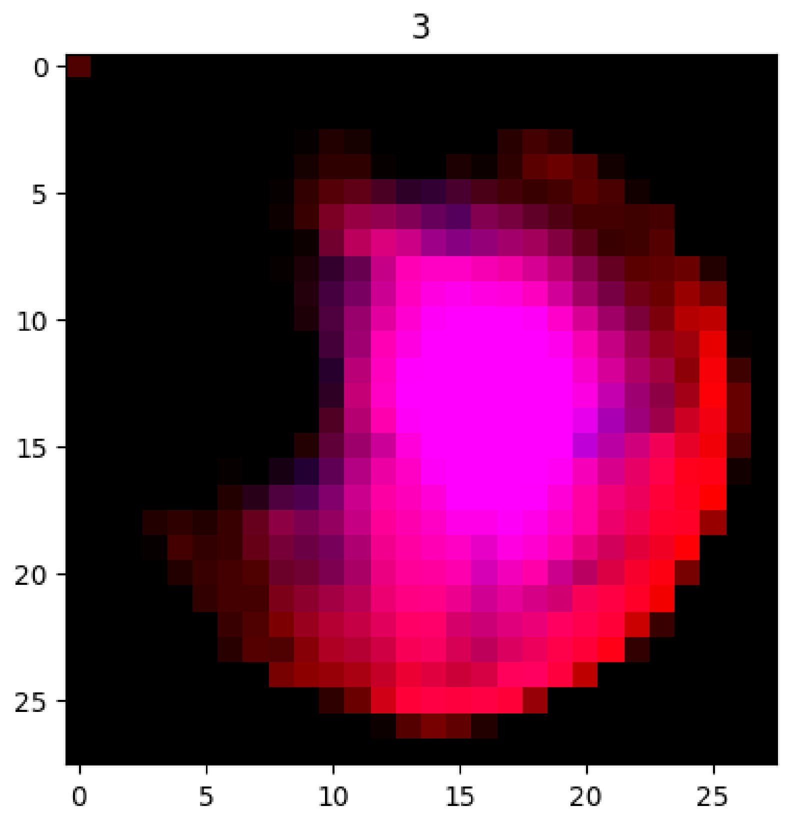
```
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers). Got range [-0.82880193..3.8703656].
```



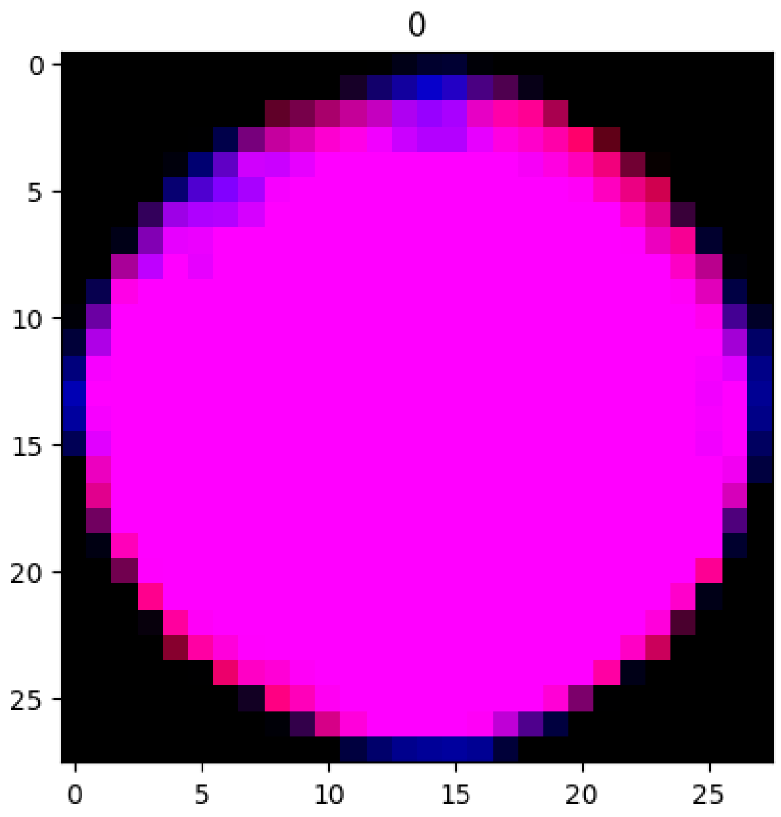
```
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers). Got range [-0.82880193..3.1457276].
```



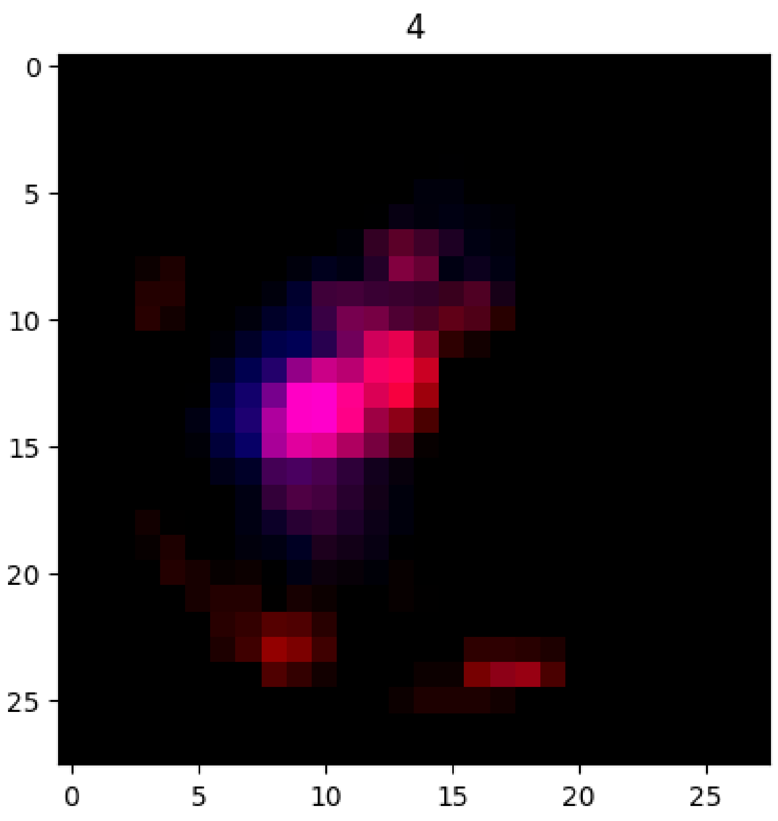
```
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers). Got range [-0.82880193..3.189645].
```



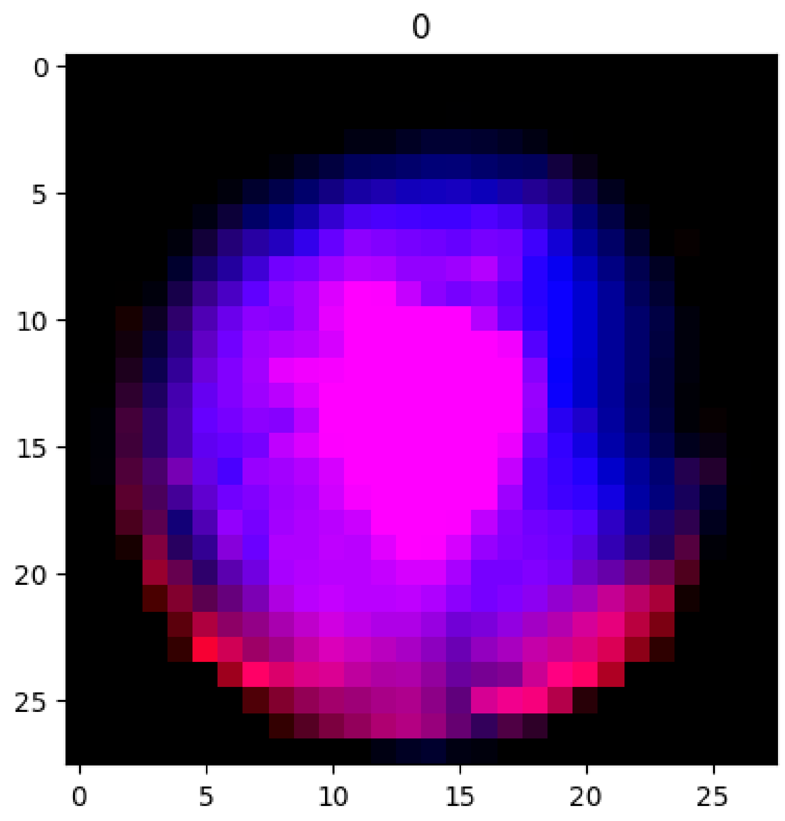
```
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers). Got range [-0.82880193..2.9041815].
```



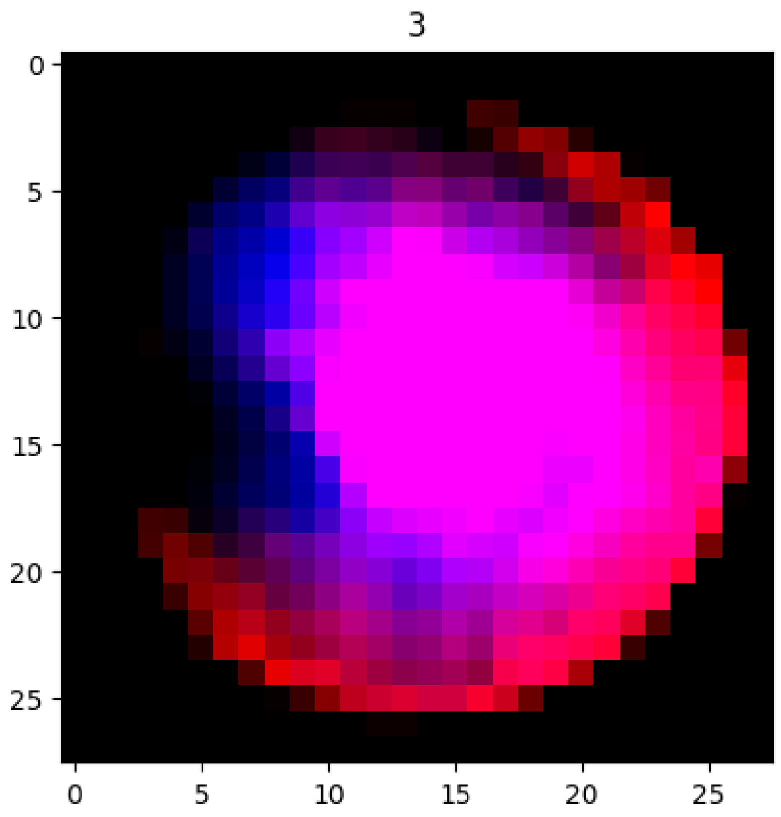
```
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers). Got range [-0.82880193..1.5866578].
```



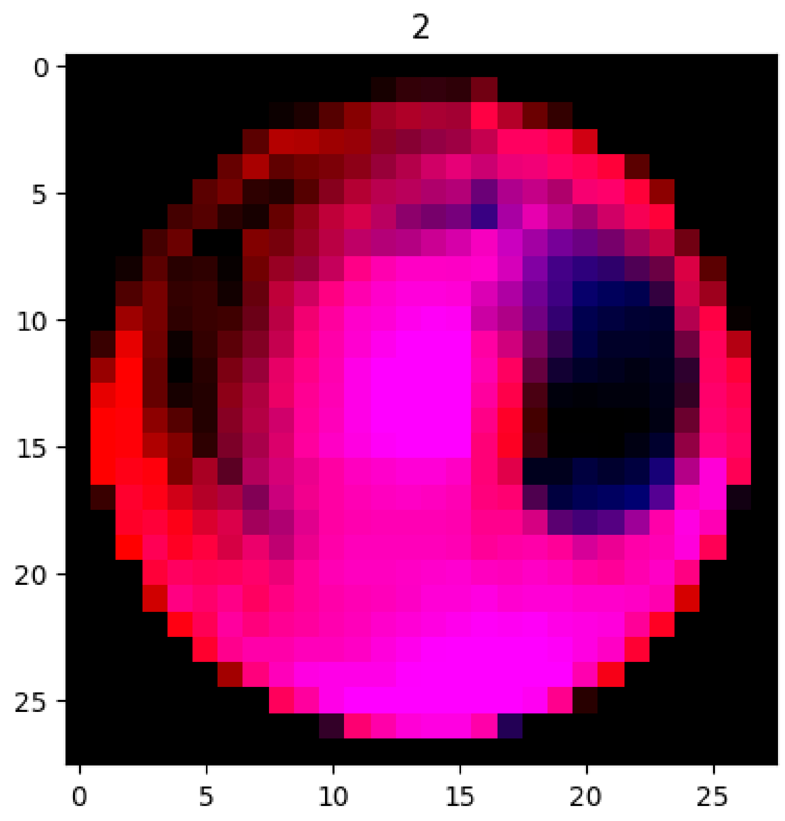
```
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers). Got range [-0.82880193..2.6406767].
```



```
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers). Got range [-0.82880193..3.1018102].
```



```
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers). Got range [-0.82880193..3.431191].
```



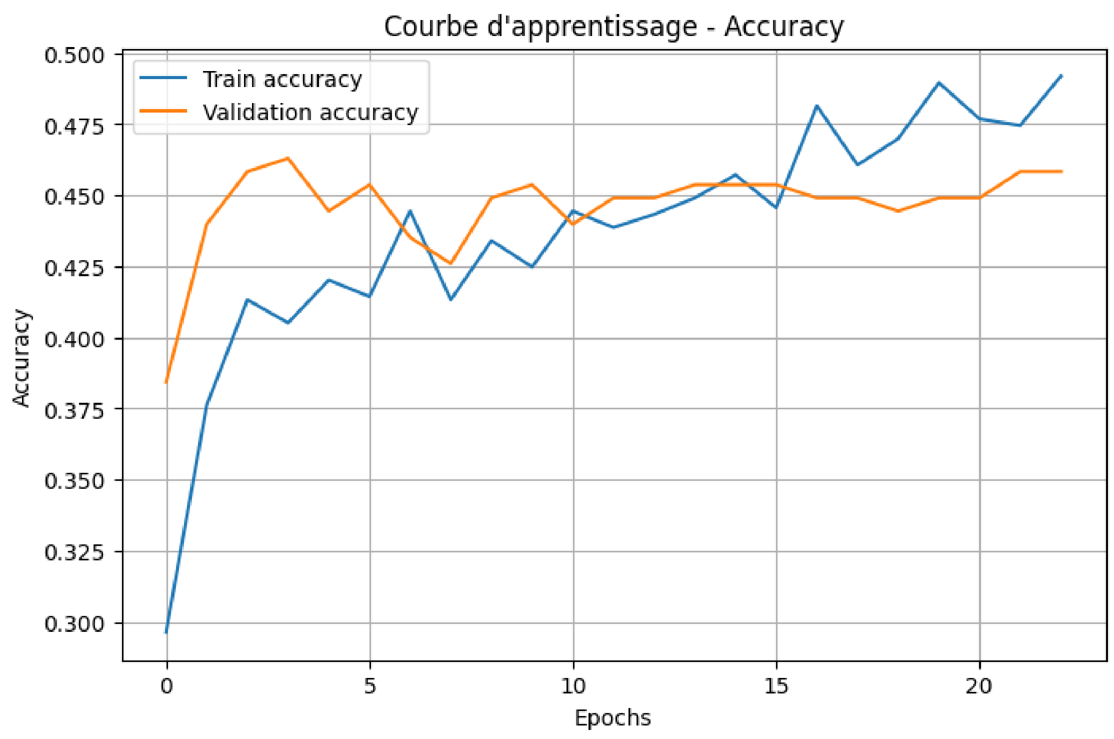
```
Class weights: {np.uint8(0): np.float64(0.4442159383033419), np.uint8(1):
np.float64(1.6941176470588235), np.uint8(2): np.float64(1.0472727272727274),
np.uint8(3): np.float64(1.1148387096774193), np.uint8(4):
np.float64(3.260377358490566)}
Epoch 1/100
27/27             2s 27ms/step -
accuracy: 0.2963 - loss: 2.2645 - recall: 0.2407 - val_accuracy: 0.3843 -
val_loss: 1.4910 - val_recall: 0.0000e+00 - learning_rate: 0.0010
Epoch 2/100
27/27             1s 21ms/step -
accuracy: 0.3762 - loss: 1.8915 - recall: 0.2963 - val_accuracy: 0.4398 -
val_loss: 1.4103 - val_recall: 0.0000e+00 - learning_rate: 0.0010
Epoch 3/100
27/27             1s 20ms/step -
accuracy: 0.4132 - loss: 1.7187 - recall: 0.3275 - val_accuracy: 0.4583 -
val_loss: 1.3454 - val_recall: 0.1898 - learning_rate: 0.0010
Epoch 4/100
27/27             1s 20ms/step -
accuracy: 0.4051 - loss: 1.6750 - recall: 0.2859 - val_accuracy: 0.4630 -
val_loss: 1.3567 - val_recall: 0.1759 - learning_rate: 0.0010
Epoch 5/100
```

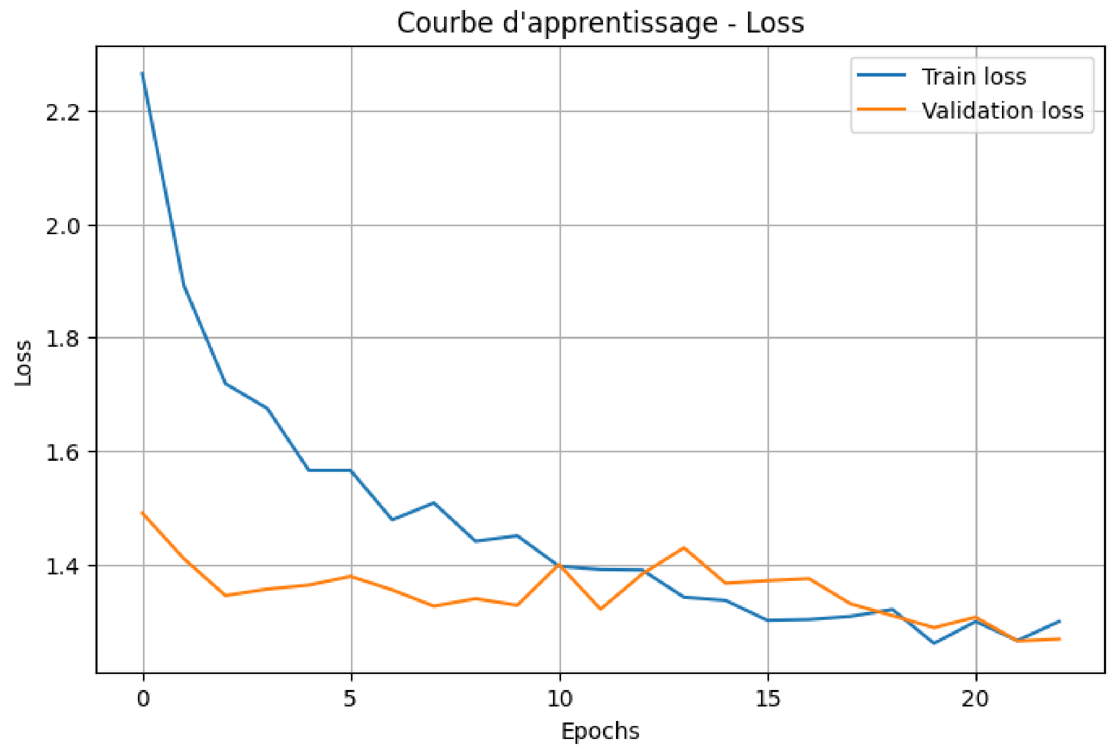
```
27/27             1s 20ms/step -
accuracy: 0.4201 - loss: 1.5658 - recall: 0.2963 - val_accuracy: 0.4444 -
val_loss: 1.3639 - val_recall: 0.1991 - learning_rate: 0.0010
Epoch 6/100
27/27             1s 21ms/step -
accuracy: 0.4144 - loss: 1.5658 - recall: 0.2870 - val_accuracy: 0.4537 -
val_loss: 1.3792 - val_recall: 0.2037 - learning_rate: 0.0010
Epoch 7/100
27/27             1s 20ms/step -
accuracy: 0.4444 - loss: 1.4789 - recall: 0.2859 - val_accuracy: 0.4352 -
val_loss: 1.3553 - val_recall: 0.2130 - learning_rate: 0.0010
Epoch 8/100
27/27             1s 20ms/step -
accuracy: 0.4132 - loss: 1.5087 - recall: 0.2755 - val_accuracy: 0.4259 -
val_loss: 1.3267 - val_recall: 0.1898 - learning_rate: 0.0010
Epoch 9/100
27/27             1s 20ms/step -
accuracy: 0.4340 - loss: 1.4411 - recall: 0.2581 - val_accuracy: 0.4491 -
val_loss: 1.3399 - val_recall: 0.3194 - learning_rate: 0.0010
Epoch 10/100
27/27             1s 20ms/step -
accuracy: 0.4248 - loss: 1.4508 - recall: 0.2581 - val_accuracy: 0.4537 -
val_loss: 1.3282 - val_recall: 0.2315 - learning_rate: 0.0010
Epoch 11/100
27/27             1s 20ms/step -
accuracy: 0.4444 - loss: 1.3971 - recall: 0.2951 - val_accuracy: 0.4398 -
val_loss: 1.4002 - val_recall: 0.3935 - learning_rate: 0.0010
Epoch 12/100
27/27             1s 20ms/step -
accuracy: 0.4387 - loss: 1.3913 - recall: 0.2650 - val_accuracy: 0.4491 -
val_loss: 1.3213 - val_recall: 0.2963 - learning_rate: 0.0010
Epoch 13/100
27/27             1s 20ms/step -
accuracy: 0.4433 - loss: 1.3904 - recall: 0.2708 - val_accuracy: 0.4491 -
val_loss: 1.3835 - val_recall: 0.3843 - learning_rate: 0.0010
Epoch 14/100
27/27             1s 30ms/step -
accuracy: 0.4491 - loss: 1.3425 - recall: 0.2812 - val_accuracy: 0.4537 -
val_loss: 1.4296 - val_recall: 0.3843 - learning_rate: 0.0010
Epoch 15/100
27/27             1s 21ms/step -
accuracy: 0.4572 - loss: 1.3368 - recall: 0.2847 - val_accuracy: 0.4537 -
val_loss: 1.3671 - val_recall: 0.3519 - learning_rate: 0.0010
Epoch 16/100
27/27             1s 24ms/step -
accuracy: 0.4456 - loss: 1.3017 - recall: 0.2812 - val_accuracy: 0.4537 -
val_loss: 1.3718 - val_recall: 0.3194 - learning_rate: 0.0010
Epoch 17/100
```

```
27/27            1s 25ms/step -
accuracy: 0.4815 - loss: 1.3032 - recall: 0.2882 - val_accuracy: 0.4491 -
val_loss: 1.3753 - val_recall: 0.3565 - learning_rate: 5.0000e-04
Epoch 18/100
27/27            1s 25ms/step -
accuracy: 0.4606 - loss: 1.3087 - recall: 0.2755 - val_accuracy: 0.4491 -
val_loss: 1.3308 - val_recall: 0.3472 - learning_rate: 5.0000e-04
Epoch 19/100
27/27            1s 21ms/step -
accuracy: 0.4699 - loss: 1.3209 - recall: 0.2743 - val_accuracy: 0.4444 -
val_loss: 1.3100 - val_recall: 0.3472 - learning_rate: 5.0000e-04
Epoch 20/100
27/27            1s 22ms/step -
accuracy: 0.4896 - loss: 1.2614 - recall: 0.2894 - val_accuracy: 0.4491 -
val_loss: 1.2889 - val_recall: 0.3333 - learning_rate: 5.0000e-04
Epoch 21/100
27/27            1s 20ms/step -
accuracy: 0.4769 - loss: 1.2997 - recall: 0.2870 - val_accuracy: 0.4491 -
val_loss: 1.3073 - val_recall: 0.3657 - learning_rate: 5.0000e-04
Epoch 22/100
27/27            1s 21ms/step -
accuracy: 0.4745 - loss: 1.2664 - recall: 0.2905 - val_accuracy: 0.4583 -
val_loss: 1.2656 - val_recall: 0.3148 - learning_rate: 2.5000e-04
Epoch 23/100
27/27            1s 20ms/step -
accuracy: 0.4919 - loss: 1.2996 - recall: 0.2882 - val_accuracy: 0.4583 -
val_loss: 1.2689 - val_recall: 0.3194 - learning_rate: 2.5000e-04
7/7            0s 13ms/step
Balanced accuracy: 0.20150674068199842
              precision    recall  f1-score   support

           0       0.46      0.97      0.62        97
           1       0.11      0.04      0.06        26
           2       0.00      0.00      0.00        41
           3       0.00      0.00      0.00        39
           4       0.00      0.00      0.00        13

    accuracy                           0.44       216
   macro avg       0.11      0.20      0.14       216
weighted avg       0.22      0.44      0.29       216


/Users/yamira.poldosilva/Documents/UDEM/A25/IFT3395/kaggle2/kaggle2/lib/python3.
13/site-packages/sklearn/metrics/_classification.py:1731:
UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels
with no predicted samples. Use `zero_division` parameter to control this
behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", result.shape[0])
```

```
/Users/yamira.poldosilva/Documents/UDEM/A25/IFT3395/kaggle2/kaggle2/lib/python3.
13/site-packages/sklearn/metrics/_classification.py:1731:
UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels
with no predicted samples. Use `zero_division` parameter to control this
behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", result.shape[0])
/Users/yamira.poldosilva/Documents/UDEM/A25/IFT3395/kaggle2/kaggle2/lib/python3.
13/site-packages/sklearn/metrics/_classification.py:1731:
UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels
with no predicted samples. Use `zero_division` parameter to control this
behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", result.shape[0])
```

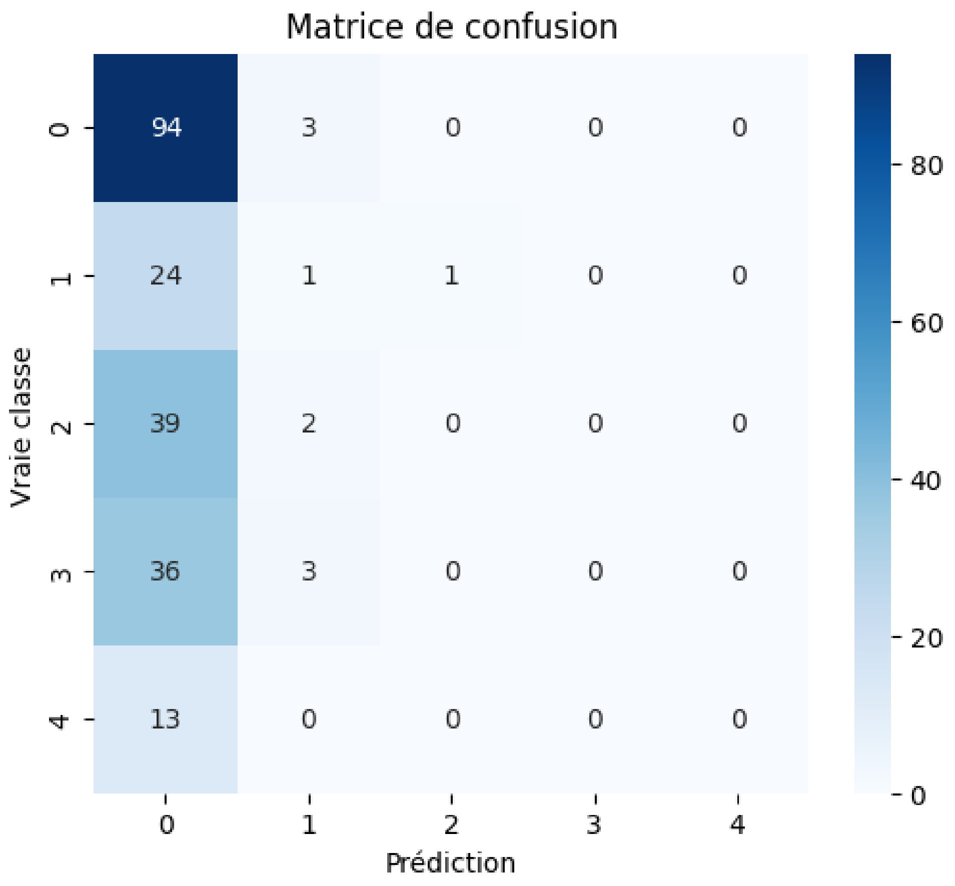
[4]:
```python
import matplotlib.pyplot as plt

# 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()
```


Courbe d'apprentissage - Accuracy
Train accuracy
Validation accuracy
Accuracy
Epochs
0.500
0.475
0.450
0.425
0.400
0.375
0.350
0.325
0.300
0
5
10
15
20


Courbe d'apprentissage - Loss
Train loss
Validation loss
Loss
Epochs
2.2
2.0
1.8
1.6
1.4
0
5
10
15
20

[5]:
```python
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 6ms/step
```



```
Recall par classe : [0.96907216 0.03846154 0.         0.         0.        ]
acc: 0.4398148148148148
Recall macro : 0.20150674068199842
```

[6]:
```
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 !")
"""
```

```
[6]: '\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'
```