

Modele1__to__validate

November 18, 2025

```
[3203]: import sys
import numpy as np
import pickle
np.random.seed(0)
from scripts import (
    #train_test_split,
    StandardScaler,
    accuracy,
    confusion_matrix,
    recall_per_class,
    balanced_accuracy,
    KernelPerceptron,
    SVM,
    rbf_kernel,
    rbf_kernel_svm,
    hybrid_kernel
)

import pandas as pd

[3204]: from sklearn.model_selection import train_test_split

# --- Load training 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_imgs = train_data["images"].astype(np.float32)
y = train_data["labels"].reshape(-1)

X_train_imgs, X_val_imgs, y_train, y_val = train_test_split(
    X_imgs, y, test_size=0.2, random_state=0, stratify=y
)

train_min = X_train_imgs.min()
train_max = X_train_imgs.max()

train_mean = X_train_imgs.mean()
```

```

train_std = X_train_imgs.std()

X_train_imgs = (X_train_imgs - train_min) / (train_max - train_min + 1e-6)
X_train_imgs = (X_train_imgs - train_mean) / (train_std + 1e-6)

X_val_imgs = (X_val_imgs - train_min) / (train_max - train_min + 1e-6)
X_val_imgs = (X_val_imgs - train_mean) / (train_std + 1e-6)

```

```

[3205]: def radial_profile(img):
    """Calcule le profil radial moyen d'une image."""
    h, w = img.shape
    y, x = np.ogrid[:h, :w]
    r = np.sqrt((x - w//2)**2 + (y - h//2)**2).astype(int)
    profile = np.bincount(r.ravel(), img.ravel()) / np.bincount(r.ravel())
    return np.log(profile + 1e-6)

[3206]: def extract_simple_stats(img):
    gray = img.mean(axis=2)
    return np.array([gray.mean(), gray.std(), gray.min(), gray.max()], dtype=np.
    ↪float32)

[3207]: def simple_augment(images, labels):
    flips = images[:, :, ::-1, :]
    noise = images + 0.01*np.random.randn(*images.shape)
    aug_imgs = np.concatenate([images, flips, noise], axis=0)
    aug_labels = np.concatenate([labels, labels, labels])
    return aug_imgs, aug_labels

[3208]: def extract_channel_stats(img):
    # img.shape est (H, W, 3)
    features = []
    for c in range(img.shape[2]): # Itere sur les 3 canaux
        channel = img[:, :, c]
        features.extend([channel.mean(), channel.std(), channel.min(), channel.
    ↪max()])
    return np.array(features, dtype=np.float32)

[3209]: def fft_features(images):
    """Extrait les caractéristiques FFT d'images."""
    gray = images.mean(axis=3)
    F = np.fft.fft2(gray, axes=(1, 2))
    return np.abs(np.fft.fftshift(F, axes=(1, 2)))

[3210]: X_train_imgs, y_train = simple_augment(X_train_imgs, y_train)

fft_mag_train = fft_features(X_train_imgs)

```

```

X_fft_train = np.array([radial_profile(img) for img in fft_mag_train], dtype=np.
    ↪float32)
X_stats_train = np.array([extract_simple_stats(img) for img in X_train_imgs],
    ↪dtype=np.float32)
X_color_train = np.array([extract_channel_stats(img) for img in X_train_imgs],
    ↪dtype=np.float32)
X_train = np.hstack([X_fft_train,
                     X_stats_train, X_color_train
                     ])

fft_mag_val = fft_features(X_val_imgs)
X_fft_val = np.array([radial_profile(img) for img in fft_mag_val], dtype=np.
    ↪float32)
X_stats_val = np.array([extract_simple_stats(img) for img in X_val_imgs],
    ↪dtype=np.float32)
X_color_val = np.array([extract_channel_stats(img) for img in X_val_imgs],
    ↪dtype=np.float32)

X_val = np.hstack([X_fft_val,
                   X_stats_val, X_color_val
                   ])

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_val = scaler.transform(X_val)

```

```

[3211]: class_counts = np.bincount(y_val)
class_weights = (1.0 / class_counts)
class_weights /= class_weights.sum()
sample_weights = class_weights[y_train]
"""
model = SVM(sigma=2, max_iter=5, sample_weights=sample_weights)
model.fit(X_train, y_train)
"""

```

```

[3211]: '\nmodel = SVM(sigma=2, max_iter=5,
sample_weights=sample_weights)\nmodel.fit(X_train, y_train)\n'

```

```

[3212]: model = KernelPerceptron(kernel_fn=rbf_kernel, n_classes=5, sigma=1)
model.fit(X_train, y_train, max_epochs=50)

```

```

Training classifier for class 0...
Training classifier for class 1...
Training classifier for class 2...
Training classifier for class 3...
Training classifier for class 4...

```

Training done.

```
[3213]: """
model = SoftmaxClassifier(input_dim=X.shape[1], num_classes=num_classes, reg=0.
    ↪05, seed=0)
model.fit(X_train, y_train, lr=2, n_steps=5000, sample_weights=sample_weights)
"""
```

```
[3213]: '\nmodel = SoftmaxClassifier(input_dim=X.shape[1], num_classes=num_classes,
reg=0.05, seed=0)\nmodel.fit(X_train, y_train, lr=2, n_steps=5000,
sample_weights=sample_weights)\n'
```

Understanding Deep Learning Requires Rethinking Generalization (Zhang et al., ICLR 2017) → importance de la fréquence dans image classification. Fourier Transform in Image Processing — Gonzalez & Woods (exemples de séparation de classes via magnitude spectrale) FOURIER CNNs (Rippel et al., 2015) → montre qu'une représentation FFT est plus stable et expressive que pixels bruts. Spectral Representations for Image Classification (several IEEE papers) → radial power spectra suffisent à classer des textures complexes. Why Do Deep Neural Networks Learn High-Frequency Patterns? (Xu et al., NeurIPS 2019)

```
[3214]: y_pred_val = model.predict(X_val)
acc = (y_pred_val == y_val).mean()
print("Test accuracy =", acc)

cm = confusion_matrix(y_val, y_pred_val)
bal_acc = balanced_accuracy(y_val, y_pred_val)
rec = recall_per_class(cm)

print("Balanced acc :", bal_acc)
print("Recall par classe :", rec)
print("Recall moyen :", rec.mean())
print(cm)
```

```
Test accuracy = 0.4537037037037037
Balanced acc : 0.33001618279465905
Recall par classe : [0.68041237 0.19230769 0.34146341 0.28205128 0.15384615]
Recall moyen : 0.33001618279465905
[[66  4 17  5  5]
 [ 8  5  6  7  0]
 [11  1 14 13  2]
 [13  3 10 11  2]
 [ 4  1  4  2  2]]
```

```
[3215]: model_pkg = {
    'model': model,
    'scaler': scaler,
    'train_min': train_min,
    'train_max': train_max,
    'train_mean': train_mean,
```

```

        'train_std': train_std
    }

pickle.dump(model_pkg, open("model_perceptron.pkl", "wb"))

```

```

[ ]: # -----
# 1. Charger le modèle entraîné
# -----
model_pkg_pred = pickle.load(open("model_perceptron.pkl", "rb"))
model = model_pkg_pred['model']
scaler = model_pkg_pred['scaler']
train_min = model_pkg_pred['train_min']
train_max = model_pkg_pred['train_max']
train_mean = model_pkg_pred['train_mean']
train_std = model_pkg_pred['train_std']

# -----
# 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"].astype(np.float32)

X_test_imgs = (X_test_imgs - train_min) / (train_max - train_min + 1e-6)
X_test_imgs = (X_test_imgs - train_mean) / (train_std + 1e-6)

# -----
# 4. Normaliser avec les stats du train
# -----
fft_mag_test = fft_features(X_test_imgs)
X_fft_test = np.array([radial_profile(img) for img in fft_mag_test], dtype=np.
    float32)
X_stats_test = np.array([extract_simple_stats(img) for img in X_test_imgs],
    dtype=np.float32)
X_color_test = np.array([extract_channel_stats(img) for img in X_test_imgs],
    dtype=np.float32)

X_test = np.hstack([X_fft_test, X_stats_test, X_color_test])

X_test = scaler.transform(X_test)

# -----
# 5. Prédire
# -----

```

```

y_pred = model.predict(X_test).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_YAPS_MCS_V14.csv", index=False)

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

#print(df.head())

df1 = pd.read_csv("ift3395_YAPS_MCS_V14.csv")
df2 = pd.read_csv("ift3395_YamirPoldoSilvaV6_good.csv")

comparison = df1.compare(df2)
print(comparison)
print("Nombre de différences :", len(comparison))

```

Fichier 'submission.csv' généré !

	Label	
	self	other
1	3.0	0.0
3	0.0	2.0
4	0.0	3.0
6	4.0	0.0
7	0.0	2.0
..
393	4.0	1.0
395	0.0	2.0
396	0.0	2.0
397	1.0	2.0
399	0.0	3.0

[212 rows x 2 columns]

Nombre de différences : 212