modelepdf

November 15, 2025

[254]:

```python
import sys
import numpy as np
import matplotlib.pyplot as plt

np.random.seed(2)
```

[ ]:

```python
import numpy as np

def extract_features_light(img):
    """
    Version légère: ~100 features au lieu de 337
    Garde les plus importantes
    """
    features = []
    img = img.astype(np.float32)
    gray = img.mean(axis=2)

    # 1. RGB stats (18)
    for c in range(3):
        channel = img[:, :, c]
        features.append(channel.mean())
        features.append(channel.std())
        features.append(np.percentile(channel, 10))
        features.append(np.percentile(channel, 50))
        features.append(np.percentile(channel, 90))
        features.append(channel.max())

    # 2. Grayscale (6)
    features.append(gray.mean())
    features.append(gray.std())
    features.append(np.percentile(gray, 25))
    features.append(np.percentile(gray, 75))
    features.append(gray.min())
    features.append(gray.max())

    # 3. Edges (3)
    gx = np.abs(gray[:, 1:] - gray[:, :-1]).mean()
    gy = np.abs(gray[1:, :] - gray[:-1, :]).mean()
    features.extend([gx, gy, np.sqrt(gx**2 + gy**2)])

    # 4. Quadrants (8)
    H, W = gray.shape
    h2, w2 = H//2, W//2
    for quad in [gray[:h2, :w2], gray[:h2, w2:], gray[h2:, :w2], gray[h2:, w2:
↪]]:
        features.append(quad.mean())
        features.append(quad.std())

    # 5. Color ratios (3)
    R, G, B = img[:,:,0], img[:,:,1], img[:,:,2]
    features.append(R.mean() / (G.mean() + 1e-8))
    features.append(R.mean() / (B.mean() + 1e-8))
    features.append(G.mean() / (B.mean() + 1e-8))

    # 6. LBP (64 bins au lieu de 256)
    def lbp(gray):
        H, W = gray.shape
        padded = np.pad(gray, ((1, 1), (1, 1)), mode="edge")
        lbp_img = np.zeros((H, W), dtype=np.uint8)
        offsets = [(-1,-1),(-1,0),(-1,1),(0,-1),(0,1),(1,-1),(1,0),(1,1)]
        for idx, (dy, dx) in enumerate(offsets):
            neigh = padded[1+dy:H+1+dy, 1+dx:W+1+dx]
            bit = (neigh >= gray).astype(np.uint8)
            lbp_img |= (bit << idx)
        return lbp_img

    lbp_img = lbp(gray)
    hist_lbp = np.histogram(lbp_img, bins=64, range=(0, 256))[0]
    features.extend(hist_lbp.tolist())


    return np.array(features, dtype=np.float32)
```

[256]:

```
class StandardScaler:
    def fit(self, X):
        self.mu = X.mean(axis=0)
        self.sigma = X.std(axis=0) + 1e-8
    def transform(self, X):
        return (X - self.mu) / self.sigma
    def fit_transform(self, X):
        self.fit(X)
        return self.transform(X)
```

[257]:

```
class SoftmaxClassifier:
    def __init__(self, input_dim, num_classes, reg=0.0, seed=None):
```

```python
        if seed is not None:
            np.random.seed(seed)

        self.W = 0.01 * np.random.randn(input_dim, num_classes).astype(np.float32)
        self.reg = reg  # L2
        self.b = np.zeros(num_classes)
        self.seed = seed

    def _softmax(self, scores):
        # scores: (N, K)
        scores = scores - scores.max(axis=1, keepdims=True)
        exp_scores = np.exp(scores)
        return exp_scores / exp_scores.sum(axis=1, keepdims=True)

    def loss(self, X, y, sample_weights=None):
        N = X.shape[0]

        scores = X @ self.W + self.b
        probs = self._softmax(scores)

        correct_logprobs = -np.log(probs[np.arange(N), y] + 1e-12)

        if sample_weights is None:
            sample_weights = np.ones(N)

        loss = np.sum(sample_weights * correct_logprobs) / N
        loss += 0.5 * self.reg * np.sum(self.W**2)

        return loss, probs

    def grad(self, X, y, probs, sample_weights=None):
        N = X.shape[0]

        if sample_weights is None:
            sample_weights = np.ones(N)

        dscores = probs.copy()
        dscores[np.arange(N), y] -= 1
        dscores *= sample_weights[:, None]
        dscores /= N

        dW = X.T @ dscores + self.reg * self.W
        db = dscores.sum(axis=0)
        return dW, db
    def fit(self, X, y, lr=1e-4, n_steps=1000, sample_weights=None,␣
↪verbose=True):
        losses = []
        for step in range(n_steps):

            loss, probs = self.loss(X, y, sample_weights)
            dW, db = self.grad(X, y, probs, sample_weights)

            self.W -= lr * dW
            self.b -= lr * db

            losses.append(loss)

            if verbose and step % 100 == 0:
                print(f"Step {step}, loss = {loss:.4f}")

        return losses


    def predict_proba(self, X):
        scores = X @ self.W
        probs = self._softmax(scores)
        return probs

    def predict(self, X):
        probs = self.predict_proba(X)
        return probs.argmax(axis=1)
```

[258]:

```
def accuracy(y_true, y_pred):
    y_true = np.asarray(y_true)
    y_pred = np.asarray(y_pred)
    return np.mean(y_true == y_pred)
```

[259]:

```
def confusion_matrix_np(y_true, y_pred, num_classes=None):
    y_true = np.asarray(y_true).astype(int)
    y_pred = np.asarray(y_pred).astype(int)

    if num_classes is None:
        num_classes = max(y_true.max(), y_pred.max()) + 1

    cm = np.zeros((num_classes, num_classes), dtype=int)
    for t, p in zip(y_true, y_pred):
        cm[t, p] += 1
    return cm
```

[260]:

```
def balanced_accuracy(y_true, y_pred):
    cm = confusion_matrix_np(y_true, y_pred)
    TP = np.diag(cm)
    real_pos = cm.sum(axis=1)
    recall = np.where(real_pos > 0, TP / real_pos, 0.0)
    return recall.mean()
```

[261]:

```
def recall_per_class(cm):
    """
    cm : matrice de confusion (numpy array KxK)
    retourne un vecteur de recall par classe
    """
    TP = np.diag(cm)
    real_pos = cm.sum(axis=1)   # total de vrais échantillons par classe

    # recall par classe (évite division par zéro)
    recall = np.where(real_pos > 0, TP / real_pos, 0.0)
    return recall
```

[262]:

```
import pickle
path_to_data = 'ift-3395-6390-kaggle-2-competition-fall-2025/train_data.pkl'

# --- Load training data ---
with open(path_to_data, "rb") as f:
    train_data = pickle.load(f)

X_imgs = train_data["images"]
y = train_data["labels"].reshape(-1)

# --- Feature extraction ---
X = np.array([extract_features(img) for img in X_imgs], dtype=np.float32)

# --- Normalize ---
scaler = StandardScaler()
X = scaler.fit_transform(X)

# --- Split ---
n_train = int(0.8 * len(X))
X_train, X_test = X[:n_train], X[n_train:]
y_train, y_test = y[:n_train], y[n_train:]

# --- Class weights ---
class_counts = np.bincount(y_train)
class_weights = (1.0 / class_counts)
class_weights /= class_weights.sum()
sample_weights = class_weights[y_train]
```

```python
# --- Train model ---
num_classes = len(np.unique(y))
model = SoftmaxClassifier(input_dim=X.shape[1], num_classes=num_classes, reg=0.05, seed=0)

model.fit(X_train, y_train, lr=0.0005, n_steps=50000, sample_weights=sample_weights)

# --- Evaluate ---
y_pred = model.predict(X_test)
acc = (y_pred == y_test).mean()
print("Test accuracy =", acc)

# --- Save model ---
pickle.dump((model, scaler), open("model_softmax.pkl", "wb"))


#print("Train accuracy:", acc)
print("Test accuracy:", acc)
cm = confusion_matrix_np(y_test, y_pred)
acc = accuracy(y_test, y_pred)
bal_acc = balanced_accuracy(y_test, y_pred)


rec = recall_per_class(cm)
print("Accuracy        :", acc)
print("Balanced acc    :", bal_acc)
print("Confusion matrix:\n", cm)
print("Recall par classe :", rec)
print("Recall moyen (macro):", rec.mean())
```

```
Step 0, loss = 0.2262
Step 100, loss = 0.2246
Step 200, loss = 0.2231
Step 300, loss = 0.2218
Step 400, loss = 0.2205
Step 500, loss = 0.2193
Step 600, loss = 0.2182
Step 700, loss = 0.2171
Step 800, loss = 0.2161
Step 900, loss = 0.2152
Step 1000, loss = 0.2143
Step 1100, loss = 0.2135
Step 1200, loss = 0.2127
Step 1300, loss = 0.2119
Step 1400, loss = 0.2112
Step 1500, loss = 0.2105
Step 1600, loss = 0.2099
Step 1700, loss = 0.2092
Step 1800, loss = 0.2086
Step 1900, loss = 0.2080
Step 2000, loss = 0.2075
Step 2100, loss = 0.2070
Step 2200, loss = 0.2064
Step 2300, loss = 0.2059
Step 2400, loss = 0.2054
Step 2500, loss = 0.2050
Step 2600, loss = 0.2045
Step 2700, loss = 0.2041
Step 2800, loss = 0.2036
Step 2900, loss = 0.2032
Step 3000, loss = 0.2028
Step 3100, loss = 0.2024
Step 3200, loss = 0.2020
Step 3300, loss = 0.2016
Step 3400, loss = 0.2012
Step 3500, loss = 0.2009
Step 3600, loss = 0.2005
Step 3700, loss = 0.2002
Step 3800, loss = 0.1998
Step 3900, loss = 0.1995
Step 4000, loss = 0.1992
Step 4100, loss = 0.1988
Step 4200, loss = 0.1985
Step 4300, loss = 0.1982
Step 4400, loss = 0.1979
Step 4500, loss = 0.1976
Step 4600, loss = 0.1973
Step 4700, loss = 0.1970
Step 4800, loss = 0.1967
Step 4900, loss = 0.1965
Step 5000, loss = 0.1962
Step 5100, loss = 0.1959
Step 5200, loss = 0.1956
Step 5300, loss = 0.1954
Step 5400, loss = 0.1951
Step 5500, loss = 0.1949
Step 5600, loss = 0.1946
Step 5700, loss = 0.1944
Step 5800, loss = 0.1941
Step 5900, loss = 0.1939
Step 6000, loss = 0.1936
Step 6100, loss = 0.1934
Step 6200, loss = 0.1932
Step 6300, loss = 0.1929
Step 6400, loss = 0.1927
Step 6500, loss = 0.1925
Step 6600, loss = 0.1923
Step 6700, loss = 0.1921
Step 6800, loss = 0.1918
Step 6900, loss = 0.1916
Step 7000, loss = 0.1914
Step 7100, loss = 0.1912
Step 7200, loss = 0.1910
Step 7300, loss = 0.1908
Step 7400, loss = 0.1906
Step 7500, loss = 0.1904
Step 7600, loss = 0.1902
Step 7700, loss = 0.1900
Step 7800, loss = 0.1898
Step 7900, loss = 0.1896
Step 8000, loss = 0.1895
Step 8100, loss = 0.1893
Step 8200, loss = 0.1891
Step 8300, loss = 0.1889
Step 8400, loss = 0.1887
Step 8500, loss = 0.1886
Step 8600, loss = 0.1884
Step 8700, loss = 0.1882
Step 8800, loss = 0.1880
Step 8900, loss = 0.1879
Step 9000, loss = 0.1877
Step 9100, loss = 0.1875
Step 9200, loss = 0.1874
Step 9300, loss = 0.1872
Step 9400, loss = 0.1871
Step 9500, loss = 0.1869
Step 9600, loss = 0.1868
Step 9700, loss = 0.1866
Step 9800, loss = 0.1864
Step 9900, loss = 0.1863
Step 10000, loss = 0.1861
Step 10100, loss = 0.1860
Step 10200, loss = 0.1858
Step 10300, loss = 0.1857
Step 10400, loss = 0.1856
Step 10500, loss = 0.1854
Step 10600, loss = 0.1853
Step 10700, loss = 0.1851
Step 10800, loss = 0.1850
Step 10900, loss = 0.1849
Step 11000, loss = 0.1847
Step 11100, loss = 0.1846
Step 11200, loss = 0.1845
Step 11300, loss = 0.1843
Step 11400, loss = 0.1842
Step 11500, loss = 0.1841
Step 11600, loss = 0.1839
Step 11700, loss = 0.1838
Step 11800, loss = 0.1837
Step 11900, loss = 0.1836
Step 12000, loss = 0.1834
Step 12100, loss = 0.1833
Step 12200, loss = 0.1832
Step 12300, loss = 0.1831
Step 12400, loss = 0.1830
Step 12500, loss = 0.1828
Step 12600, loss = 0.1827
Step 12700, loss = 0.1826
Step 12800, loss = 0.1825
Step 12900, loss = 0.1824
Step 13000, loss = 0.1823
Step 13100, loss = 0.1822
Step 13200, loss = 0.1821
Step 13300, loss = 0.1819
Step 13400, loss = 0.1818
Step 13500, loss = 0.1817
Step 13600, loss = 0.1816
Step 13700, loss = 0.1815
Step 13800, loss = 0.1814
Step 13900, loss = 0.1813
Step 14000, loss = 0.1812
Step 14100, loss = 0.1811
Step 14200, loss = 0.1810
Step 14300, loss = 0.1809
Step 14400, loss = 0.1808
Step 14500, loss = 0.1807
Step 14600, loss = 0.1806
Step 14700, loss = 0.1805
Step 14800, loss = 0.1804
Step 14900, loss = 0.1803
Step 15000, loss = 0.1802
Step 15100, loss = 0.1801
Step 15200, loss = 0.1801
Step 15300, loss = 0.1800
Step 15400, loss = 0.1799
Step 15500, loss = 0.1798
Step 15600, loss = 0.1797
Step 15700, loss = 0.1796
Step 15800, loss = 0.1795
Step 15900, loss = 0.1794
Step 16000, loss = 0.1793
Step 16100, loss = 0.1793
Step 16200, loss = 0.1792
Step 16300, loss = 0.1791
Step 16400, loss = 0.1790
Step 16500, loss = 0.1789
Step 16600, loss = 0.1789
Step 16700, loss = 0.1788
Step 16800, loss = 0.1787
Step 16900, loss = 0.1786
Step 17000, loss = 0.1785
Step 17100, loss = 0.1785
Step 17200, loss = 0.1784
Step 17300, loss = 0.1783
Step 17400, loss = 0.1782
Step 17500, loss = 0.1782
Step 17600, loss = 0.1781
Step 17700, loss = 0.1780
Step 17800, loss = 0.1779
Step 17900, loss = 0.1779
Step 18000, loss = 0.1778
Step 18100, loss = 0.1777
Step 18200, loss = 0.1776
Step 18300, loss = 0.1776
Step 18400, loss = 0.1775
Step 18500, loss = 0.1774
Step 18600, loss = 0.1774
Step 18700, loss = 0.1773
Step 18800, loss = 0.1772
Step 18900, loss = 0.1772
Step 19000, loss = 0.1771
Step 19100, loss = 0.1770
Step 19200, loss = 0.1770
Step 19300, loss = 0.1769
Step 19400, loss = 0.1768
Step 19500, loss = 0.1768
Step 19600, loss = 0.1767
Step 19700, loss = 0.1767
Step 19800, loss = 0.1766
Step 19900, loss = 0.1765
Step 20000, loss = 0.1765
Step 20100, loss = 0.1764
Step 20200, loss = 0.1763
Step 20300, loss = 0.1763
Step 20400, loss = 0.1762
Step 20500, loss = 0.1762
Step 20600, loss = 0.1761
Step 20700, loss = 0.1761
Step 20800, loss = 0.1760
Step 20900, loss = 0.1759
Step 21000, loss = 0.1759
Step 21100, loss = 0.1758
Step 21200, loss = 0.1758
Step 21300, loss = 0.1757
Step 21400, loss = 0.1757
Step 21500, loss = 0.1756
Step 21600, loss = 0.1756
Step 21700, loss = 0.1755
Step 21800, loss = 0.1755
Step 21900, loss = 0.1754
Step 22000, loss = 0.1754
Step 22100, loss = 0.1753
Step 22200, loss = 0.1753
Step 22300, loss = 0.1752
Step 22400, loss = 0.1752
Step 22500, loss = 0.1751
Step 22600, loss = 0.1751
Step 22700, loss = 0.1750
Step 22800, loss = 0.1750
Step 22900, loss = 0.1749
Step 23000, loss = 0.1749
Step 23100, loss = 0.1748
Step 23200, loss = 0.1748
Step 23300, loss = 0.1747
Step 23400, loss = 0.1747
Step 23500, loss = 0.1746
Step 23600, loss = 0.1746
Step 23700, loss = 0.1745
Step 23800, loss = 0.1745
Step 23900, loss = 0.1744
Step 24000, loss = 0.1744
Step 24100, loss = 0.1744
Step 24200, loss = 0.1743
Step 24300, loss = 0.1743
Step 24400, loss = 0.1742
Step 24500, loss = 0.1742
Step 24600, loss = 0.1741
Step 24700, loss = 0.1741
Step 24800, loss = 0.1741
Step 24900, loss = 0.1740
Step 25000, loss = 0.1740
Step 25100, loss = 0.1739
Step 25200, loss = 0.1739
Step 25300, loss = 0.1739
Step 25400, loss = 0.1738
Step 25500, loss = 0.1738
Step 25600, loss = 0.1737
Step 25700, loss = 0.1737
Step 25800, loss = 0.1737
Step 25900, loss = 0.1736
Step 26000, loss = 0.1736
Step 26100, loss = 0.1736
Step 26200, loss = 0.1735
Step 26300, loss = 0.1735
Step 26400, loss = 0.1734
Step 26500, loss = 0.1734
Step 26600, loss = 0.1734
Step 26700, loss = 0.1733
Step 26800, loss = 0.1733
Step 26900, loss = 0.1733
Step 27000, loss = 0.1732
Step 27100, loss = 0.1732
Step 27200, loss = 0.1732
Step 27300, loss = 0.1731
Step 27400, loss = 0.1731
Step 27500, loss = 0.1731
Step 27600, loss = 0.1730
Step 27700, loss = 0.1730
Step 27800, loss = 0.1730
Step 27900, loss = 0.1729
Step 28000, loss = 0.1729
Step 28100, loss = 0.1729
Step 28200, loss = 0.1728
Step 28300, loss = 0.1728
Step 28400, loss = 0.1728
Step 28500, loss = 0.1727
Step 28600, loss = 0.1727
Step 28700, loss = 0.1727
Step 28800, loss = 0.1726
Step 28900, loss = 0.1726
Step 29000, loss = 0.1726
Step 29100, loss = 0.1726
Step 29200, loss = 0.1725
Step 29300, loss = 0.1725
Step 29400, loss = 0.1725
Step 29500, loss = 0.1724
Step 29600, loss = 0.1724
Step 29700, loss = 0.1724
Step 29800, loss = 0.1724
Step 29900, loss = 0.1723
Step 30000, loss = 0.1723
Step 30100, loss = 0.1723
Step 30200, loss = 0.1722
Step 30300, loss = 0.1722
Step 30400, loss = 0.1722
Step 30500, loss = 0.1722
Step 30600, loss = 0.1721
Step 30700, loss = 0.1721
Step 30800, loss = 0.1721
Step 30900, loss = 0.1721
Step 31000, loss = 0.1720
Step 31100, loss = 0.1720
Step 31200, loss = 0.1720
Step 31300, loss = 0.1720
Step 31400, loss = 0.1719
Step 31500, loss = 0.1719
Step 31600, loss = 0.1719
Step 31700, loss = 0.1719
Step 31800, loss = 0.1718
Step 31900, loss = 0.1718
Step 32000, loss = 0.1718
Step 32100, loss = 0.1718
Step 32200, loss = 0.1717
Step 32300, loss = 0.1717
Step 32400, loss = 0.1717
Step 32500, loss = 0.1717
Step 32600, loss = 0.1716
Step 32700, loss = 0.1716
Step 32800, loss = 0.1716
Step 32900, loss = 0.1716
Step 33000, loss = 0.1716
Step 33100, loss = 0.1715
Step 33200, loss = 0.1715
Step 33300, loss = 0.1715
Step 33400, loss = 0.1715
Step 33500, loss = 0.1714
Step 33600, loss = 0.1714
Step 33700, loss = 0.1714
Step 33800, loss = 0.1714
Step 33900, loss = 0.1714
Step 34000, loss = 0.1713
Step 34100, loss = 0.1713
Step 34200, loss = 0.1713
Step 34300, loss = 0.1713
Step 34400, loss = 0.1713
Step 34500, loss = 0.1712
Step 34600, loss = 0.1712
Step 34700, loss = 0.1712
Step 34800, loss = 0.1712
Step 34900, loss = 0.1712
Step 35000, loss = 0.1711
Step 35100, loss = 0.1711
Step 35200, loss = 0.1711
Step 35300, loss = 0.1711
Step 35400, loss = 0.1711
Step 35500, loss = 0.1710
Step 35600, loss = 0.1710
Step 35700, loss = 0.1710
Step 35800, loss = 0.1710
Step 35900, loss = 0.1710
Step 36000, loss = 0.1710
Step 36100, loss = 0.1709
Step 36200, loss = 0.1709
Step 36300, loss = 0.1709
Step 36400, loss = 0.1709
Step 36500, loss = 0.1709
Step 36600, loss = 0.1709
Step 36700, loss = 0.1708
Step 36800, loss = 0.1708
Step 36900, loss = 0.1708
Step 37000, loss = 0.1708
Step 37100, loss = 0.1708
Step 37200, loss = 0.1708
Step 37300, loss = 0.1707
Step 37400, loss = 0.1707
Step 37500, loss = 0.1707
Step 37600, loss = 0.1707
Step 37700, loss = 0.1707
Step 37800, loss = 0.1707
Step 37900, loss = 0.1706
Step 38000, loss = 0.1706
Step 38100, loss = 0.1706
Step 38200, loss = 0.1706
Step 38300, loss = 0.1706
Step 38400, loss = 0.1706
Step 38500, loss = 0.1705
Step 38600, loss = 0.1705
Step 38700, loss = 0.1705
Step 38800, loss = 0.1705
Step 38900, loss = 0.1705
Step 39000, loss = 0.1705
Step 39100, loss = 0.1705
Step 39200, loss = 0.1704
Step 39300, loss = 0.1704
Step 39400, loss = 0.1704
Step 39500, loss = 0.1704
Step 39600, loss = 0.1704
Step 39700, loss = 0.1704
Step 39800, loss = 0.1704
Step 39900, loss = 0.1703
Step 40000, loss = 0.1703
Step 40100, loss = 0.1703
Step 40200, loss = 0.1703
Step 40300, loss = 0.1703
Step 40400, loss = 0.1703
Step 40500, loss = 0.1703
Step 40600, loss = 0.1702
Step 40700, loss = 0.1702
Step 40800, loss = 0.1702
Step 40900, loss = 0.1702
Step 41000, loss = 0.1702
Step 41100, loss = 0.1702
Step 41200, loss = 0.1702
Step 41300, loss = 0.1702
Step 41400, loss = 0.1701
Step 41500, loss = 0.1701
Step 41600, loss = 0.1701
Step 41700, loss = 0.1701
Step 41800, loss = 0.1701
Step 41900, loss = 0.1701
Step 42000, loss = 0.1701
Step 42100, loss = 0.1701
Step 42200, loss = 0.1701
Step 42300, loss = 0.1700
Step 42400, loss = 0.1700
Step 42500, loss = 0.1700
Step 42600, loss = 0.1700
Step 42700, loss = 0.1700
Step 42800, loss = 0.1700
Step 42900, loss = 0.1700
Step 43000, loss = 0.1700
Step 43100, loss = 0.1699
Step 43200, loss = 0.1699
Step 43300, loss = 0.1699
Step 43400, loss = 0.1699
Step 43500, loss = 0.1699
Step 43600, loss = 0.1699
Step 43700, loss = 0.1699
Step 43800, loss = 0.1699
Step 43900, loss = 0.1699
Step 44000, loss = 0.1699
Step 44100, loss = 0.1698
Step 44200, loss = 0.1698
Step 44300, loss = 0.1698
Step 44400, loss = 0.1698
Step 44500, loss = 0.1698
Step 44600, loss = 0.1698
Step 44700, loss = 0.1698
Step 44800, loss = 0.1698
Step 44900, loss = 0.1698
Step 45000, loss = 0.1697
Step 45100, loss = 0.1697
Step 45200, loss = 0.1697
Step 45300, loss = 0.1697
Step 45400, loss = 0.1697
Step 45500, loss = 0.1697
Step 45600, loss = 0.1697
Step 45700, loss = 0.1697
Step 45800, loss = 0.1697
Step 45900, loss = 0.1697
Step 46000, loss = 0.1697
Step 46100, loss = 0.1696
Step 46200, loss = 0.1696
Step 46300, loss = 0.1696
Step 46400, loss = 0.1696
Step 46500, loss = 0.1696
Step 46600, loss = 0.1696
Step 46700, loss = 0.1696
Step 46800, loss = 0.1696
Step 46900, loss = 0.1696
Step 47000, loss = 0.1696
Step 47100, loss = 0.1696
Step 47200, loss = 0.1695
Step 47300, loss = 0.1695
Step 47400, loss = 0.1695
Step 47500, loss = 0.1695
Step 47600, loss = 0.1695
Step 47700, loss = 0.1695
Step 47800, loss = 0.1695
Step 47900, loss = 0.1695
Step 48000, loss = 0.1695
Step 48100, loss = 0.1695
Step 48200, loss = 0.1695
Step 48300, loss = 0.1695
Step 48400, loss = 0.1694
Step 48500, loss = 0.1694
Step 48600, loss = 0.1694
Step 48700, loss = 0.1694
Step 48800, loss = 0.1694
Step 48900, loss = 0.1694
Step 49000, loss = 0.1694
Step 49100, loss = 0.1694
Step 49200, loss = 0.1694
Step 49300, loss = 0.1694
Step 49400, loss = 0.1694
Step 49500, loss = 0.1694
Step 49600, loss = 0.1694
Step 49700, loss = 0.1694
Step 49800, loss = 0.1693
Step 49900, loss = 0.1693
Test accuracy = 0.3611111111111111
Test accuracy: 0.3611111111111111
Accuracy        : 0.3611111111111111
Balanced acc    : 0.31154700854700856
Confusion matrix:
 [[54 19 11  6 10]
 [ 2  8  2  3  9]
 [ 6  7  6  7 13]
 [ 4  5  4  7 25]
 [ 0  2  1  2  3]]
Recall par classe : [0.54       0.33333333 0.15384615 0.15555556 0.375     ]
Recall moyen (macro): 0.31154700854700856
```

[263]:

```python
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.predict(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_YamirPoldoSilvaV3.csv", index=False)

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

```
Fichier 'submission.csv' généré !
```