train_model

December 24, 2025

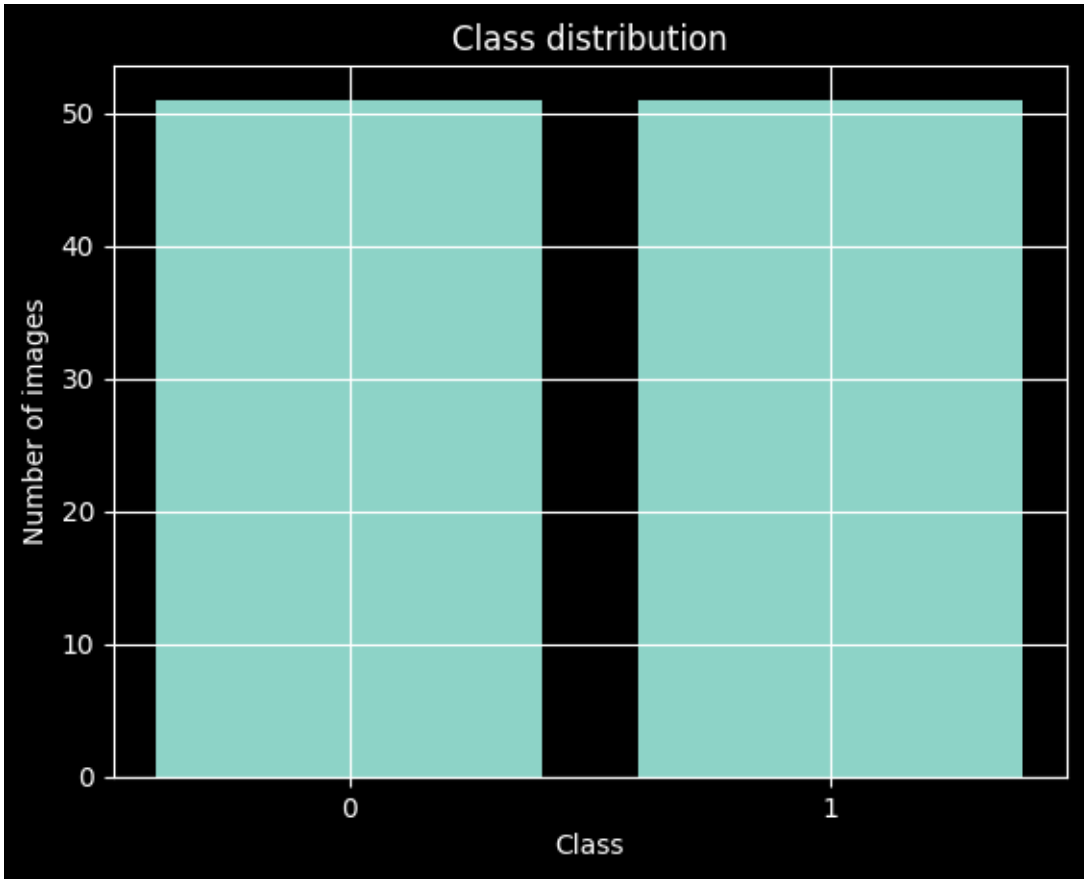
[20]:
```python
import os
import cv2
import numpy as np

X = []
y = []

dataset_dir = "captured_images"
IMG_SIZE = 32   # ← dùng 32x32 cho MLP

for label in os.listdir(dataset_dir):
    label_path = os.path.join(dataset_dir, label)

    if not os.path.isdir(label_path):
        continue

    for img_name in os.listdir(label_path):
        img_path = os.path.join(label_path, img_name)

        img = cv2.imread(img_path)
        if img is None:
            continue

        # resize về 32x32
        img = cv2.resize(img, (IMG_SIZE, IMG_SIZE))

        # normalize
        img = img / 255.0

        X.append(img)
        y.append(int(label))   # label = 0 hoặc 1

# chuyển sang numpy
X = np.array(X)
y = np.array(y).reshape(-1, 1)   # (N, 1)

# FLATTEN cho mạng 1 hidden layer
```

```
X = X.reshape(len(X), -1)

print("X shape:", X.shape)
print("y shape:", y.shape)
```

```
X shape: (102, 3072)
y shape: (102, 1)
```

[21]:
```
import matplotlib.pyplot as plt
import random
import os
import cv2

dataset_dir = "captured_images"
IMG_SIZE = 32

def show_images(label, n=5):
    folder = os.path.join(dataset_dir, str(label))
    images = random.sample(os.listdir(folder), n)

    plt.figure(figsize=(n * 2, 2))
    for i, img_name in enumerate(images):
        img_path = os.path.join(folder, img_name)
        img = cv2.imread(img_path)
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

        plt.subplot(1, n, i + 1)
        plt.imshow(img)
        plt.axis("off")
        plt.title(f"Class {label}")

    plt.show()

show_images(0)
show_images(1)
```

[22]:
```
import matplotlib.pyplot as plt
import os

labels = []
counts = []

for label in os.listdir(dataset_dir):
    label_path = os.path.join(dataset_dir, label)
    if os.path.isdir(label_path):
        labels.append(label)
        counts.append(len(os.listdir(label_path)))

plt.figure()
plt.bar(labels, counts)
plt.xlabel("Class")
plt.ylabel("Number of images")
plt.grid()
plt.title("Class distribution")
plt.show()
```



[23]:
```
np.random.seed(42)

idx = np.random.permutation(len(X))
X = X[idx]
y = y[idx]
```

[24]:
```
split = int(0.8 * len(X))

X_train = X[:split]
y_train = y[:split]

X_val = X[split:]
y_val = y[split:]
```

[25]:
```
print("Train:", X_train.shape, y_train.shape)
print("Val  :", X_val.shape, y_val.shape)
```

```
Train: (81, 3072) (81, 1)
Val  : (21, 3072) (21, 1)
```

[26]:
```
from model import *
print("y_train unique:", np.unique(y_train, return_counts=True))
print("y_val unique:", np.unique(y_val, return_counts=True))
```

```
y_train unique: (array([0, 1]), array([43, 38]))
y_val unique: (array([0, 1]), array([ 8, 13]))
```

[27]:
```
para = train(X_train, y_train, 100, 0.001)
```

```
epoch: 0, loss: 0.6930217121455131
epoch: 1, loss: 0.6930129412407114
epoch: 2, loss: 0.6930041552556414
epoch: 3, loss: 0.6929953535541423
epoch: 4, loss: 0.6929865354979142
epoch: 5, loss: 0.6929777004464547
epoch: 6, loss: 0.6929688105262345
epoch: 7, loss: 0.6929597559831603
epoch: 8, loss: 0.6929506811628838
epoch: 9, loss: 0.6929415853598886
epoch: 10, loss: 0.6929324678657447
epoch: 11, loss: 0.6929233279690329
epoch: 12, loss: 0.6929141649552649
epoch: 13, loss: 0.6929049781068071
epoch: 14, loss: 0.6928957667028003
epoch: 15, loss: 0.6928865220239355
epoch: 16, loss: 0.6928770705263119
epoch: 17, loss: 0.6928675906804781
epoch: 18, loss: 0.6928580816899272
epoch: 19, loss: 0.6928485427540308
epoch: 20, loss: 0.6928389730679438
epoch: 21, loss: 0.6928293450665227
epoch: 22, loss: 0.6928195062899754
epoch: 23, loss: 0.6928095457800763
epoch: 24, loss: 0.6927994220833525
epoch: 25, loss: 0.6927892593873707
epoch: 26, loss: 0.6927790567056795
epoch: 27, loss: 0.6927688318373972
epoch: 28, loss: 0.692758629229762
epoch: 29, loss: 0.6927484671662658
epoch: 30, loss: 0.6927387554219707
epoch: 31, loss: 0.6927291397519341
epoch: 32, loss: 0.692719771574706
epoch: 33, loss: 0.6927103932317044
epoch: 34, loss: 0.6927008759545136
epoch: 35, loss: 0.6926916325402116
epoch: 36, loss: 0.6926825330975916
epoch: 37, loss: 0.6926736999551029
epoch: 38, loss: 0.6926650830440952
```

```
epoch: 39, loss: 0.6926566559830724
epoch: 40, loss: 0.6926482670267884
epoch: 41, loss: 0.6926398485194885
epoch: 42, loss: 0.6926315415543373
epoch: 43, loss: 0.6926233051149662
epoch: 44, loss: 0.6926151048786933
epoch: 45, loss: 0.6926069073070971
epoch: 46, loss: 0.6925987088778054
epoch: 47, loss: 0.6925904955884903
epoch: 48, loss: 0.6925822671605256
epoch: 49, loss: 0.6925740139718393
epoch: 50, loss: 0.6925657356482766
epoch: 51, loss: 0.6925574318144835
epoch: 52, loss: 0.692549102093889
epoch: 53, loss: 0.6925407522493497
epoch: 54, loss: 0.69253238553851
epoch: 55, loss: 0.6925239919925001
epoch: 56, loss: 0.6925155777665106
epoch: 57, loss: 0.6925071403525144
epoch: 58, loss: 0.6924986750923213
epoch: 59, loss: 0.6924901816037015
epoch: 60, loss: 0.6924816595030857
epoch: 61, loss: 0.6924731084055484
epoch: 62, loss: 0.6924645279247876
epoch: 63, loss: 0.6924559176731082
epoch: 64, loss: 0.6924472794197056
epoch: 65, loss: 0.6924386151163309
epoch: 66, loss: 0.6924299199528615
epoch: 67, loss: 0.6924211935365934
epoch: 68, loss: 0.6924124354733403
epoch: 69, loss: 0.6924036453674142
epoch: 70, loss: 0.6923948228216071
epoch: 71, loss: 0.692385967437172
epoch: 72, loss: 0.6923770788138028
epoch: 73, loss: 0.6923681565496175
epoch: 74, loss: 0.6923592002411373
epoch: 75, loss: 0.6923502094832682
epoch: 76, loss: 0.6923411838692821
epoch: 77, loss: 0.6923321229907969
epoch: 78, loss: 0.6923230264377576
epoch: 79, loss: 0.6923138937984171
epoch: 80, loss: 0.6923047246593164
epoch: 81, loss: 0.6922955186052654
epoch: 82, loss: 0.6922862752193232
epoch: 83, loss: 0.6922769940827784
epoch: 84, loss: 0.6922676747751301
epoch: 85, loss: 0.6922583168740669
epoch: 86, loss: 0.6922489199554486
```

```
epoch: 87, loss: 0.6922394835932849
epoch: 88, loss: 0.6922300073597167
epoch: 89, loss: 0.6922204908249951
epoch: 90, loss: 0.6922109335574621
epoch: 91, loss: 0.6922013351235298
epoch: 92, loss: 0.6921916950876605
epoch: 93, loss: 0.6921820130123467
epoch: 94, loss: 0.69217228845809
epoch: 95, loss: 0.6921625209833816
epoch: 96, loss: 0.6921527101446808
epoch: 97, loss: 0.6921428554963954
epoch: 98, loss: 0.69213295659086
epoch: 99, loss: 0.6921230129783164
```

[28]:
```
y_pred = predict(X_val, para)
print("Accuracy:", accuracy(y_val, y_pred))
```

```
Accuracy: 0.6190476190476191
```

[29]:
```
y_pred = predict(X_train, para)
print("Accuracy:", accuracy(y_train, y_pred))
```

```
Accuracy: 0.4691358024691358
```

[30]:
```
# khao sat cac yeu to
```

[31]:
```
from model1 import *
```

[32]:
```
layer_configs = [
    [X.shape[1], 1],            # no hidden
    [X.shape[1], 8, 1],          # 1 hidden
    [X.shape[1], 8, 8, 1],       # 2 hidden
    [X.shape[1], 8, 8, 8, 1]     # 3 hidden
]

for layers in layer_configs:
    para = train_model(
        X_train, y_train,
        layers=layers, lr=0.01
    )
    y_pred = predict(X_val, para)
    y_hat = predict(X_train, para)
    print(f"layer {layers} accuracy train:", accuracy(y_train, y_hat))
    print(f"layer {layers} accuracy val:", accuracy(y_val, y_pred))
    print("\n")
```

```
layer [3072, 1] accuracy train: 1.0
layer [3072, 1] accuracy val: 1.0
```

```
layer [3072, 8, 1] accuracy train: 1.0
layer [3072, 8, 1] accuracy val: 1.0


layer [3072, 8, 8, 1] accuracy train: 0.5308641975308642
layer [3072, 8, 8, 1] accuracy val: 0.38095238095238093


layer [3072, 8, 8, 8, 1] accuracy train: 0.5308641975308642
layer [3072, 8, 8, 8, 1] accuracy val: 0.38095238095238093
```

[33]:
```python
neurons = [4, 8, 16, 32]

for n in neurons:
    layers = [X.shape[1], n, 1]
    para = train_model(
    X_train, y_train,
    layers=layers, lr=0.01
    )

    y_hat_train = predict(X_train, para)
    y_hat_val   = predict(X_val, para)

    print(f"neuron {n} accuracy train:", accuracy(y_train, y_hat_train))
    print(f"neuron {n} accuracy val:", accuracy(y_val, y_hat_val))
    print()
```

```
neuron 4 accuracy train: 1.0
neuron 4 accuracy val: 1.0

neuron 8 accuracy train: 1.0
neuron 8 accuracy val: 1.0

neuron 16 accuracy train: 1.0
neuron 16 accuracy val: 1.0

neuron 32 accuracy train: 1.0
neuron 32 accuracy val: 1.0
```

[34]:
```python
lrs = [0.0001, 0.001, 0.01, 0.1]

for lr in lrs:
```

```
    layers = [X.shape[1], 8, 1]
    para = train_model(
        X_train, y_train,
        layers=layers, lr=0.01
    )
    y_pred = predict(X_val, para)
    y_hat = predict(X_train, para)
    print(f"lr {lr} accuracy train:", accuracy(y_train, y_hat))
    print(f"lr {lr} accuracy val:", accuracy(y_val, y_pred))
    print("\n")
```

```
lr 0.0001 accuracy train: 1.0
lr 0.0001 accuracy val: 1.0


lr 0.001 accuracy train: 1.0
lr 0.001 accuracy val: 1.0


lr 0.01 accuracy train: 1.0
lr 0.01 accuracy val: 1.0


lr 0.1 accuracy train: 1.0
lr 0.1 accuracy val: 1.0
```

[35]:
```
import matplotlib.pyplot as plt
import random

def visualize_predictions(X, y_true, para, img_shape=(32, 32, 3), n_samples=5):
    """
    X: (N, D) flattened images
    y_true: (N, 1)
    """
    idxs = random.sample(range(X.shape[0]), n_samples)

    y_pred = predict(X[idxs], para)

    plt.figure(figsize=(15, 3))

    for i, idx in enumerate(idxs):
        img = X[idx].reshape(img_shape)

        true_label = int(y_true[idx][0])
        pred_label = int(y_pred[i][0])
```

```
        plt.subplot(1, n_samples, i + 1)
        plt.imshow(img.astype(np.uint8))
        plt.axis("off")
        plt.title(f"True: {true_label}\nPred: {pred_label}",
                  color="green" if true_label == pred_label else "red")

    plt.show()
```

[36]:
```
visualize_predictions(X_val, y_val, para, img_shape=(32, 32, 3), n_samples=5)
```