МІНІСТЕРСТВО ОСВІТИ І НАУКИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»
Інститут комп'ютерних наук та інформаційних технологій
Кафедра систем штучного інтелекту

Звіт до лабораторної №3

з дисципліни
“Обробка зображень методами штучного інтелекту”

Виконав:
ст. гр. КН-410
Романишин Микола

Викладач:
Пелешко Д. Д.

Львів – 2022

Лабораторна робота №3

Класифікація зображень. Застосування нейромереж для пошуку подібних зображень

Варіант 10

Мета – набути практичних навіків у розв’язанні задачі пошуку подібних зображень на прикладі організації CNN класифікації.

Завдання - Побудувати CNN на основі Inception для класифікації зображень на основі датасету fashion-mnist. Зробити налаштування моделі для досягнення необхідної точності. На базі Siamese networks побудувати систему для пошуку подібних зображень в датасеті fashion-mnist. Візуалізувати отримані результати t-SNE.

Хід роботи

Для початку, після проведення підготовки даних, подивимось на зображення, схожість яких потрібно шукати та на ідеальнізовані результати роботи мережі:

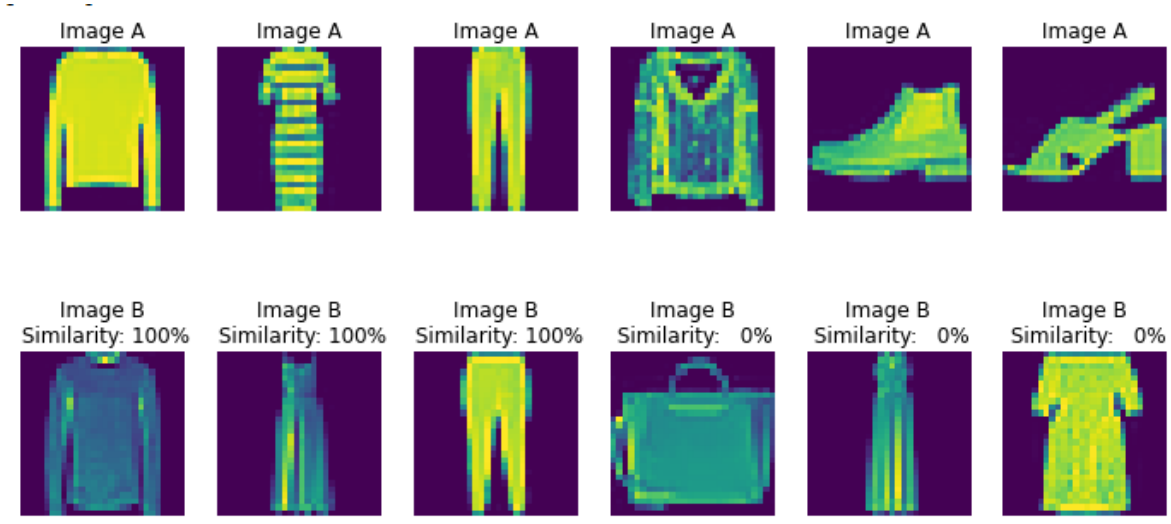

Рис. 1. Ідеальний результат роботи мережі

Далі, поглянемо на роботу не тренованої мережі (рис 2).

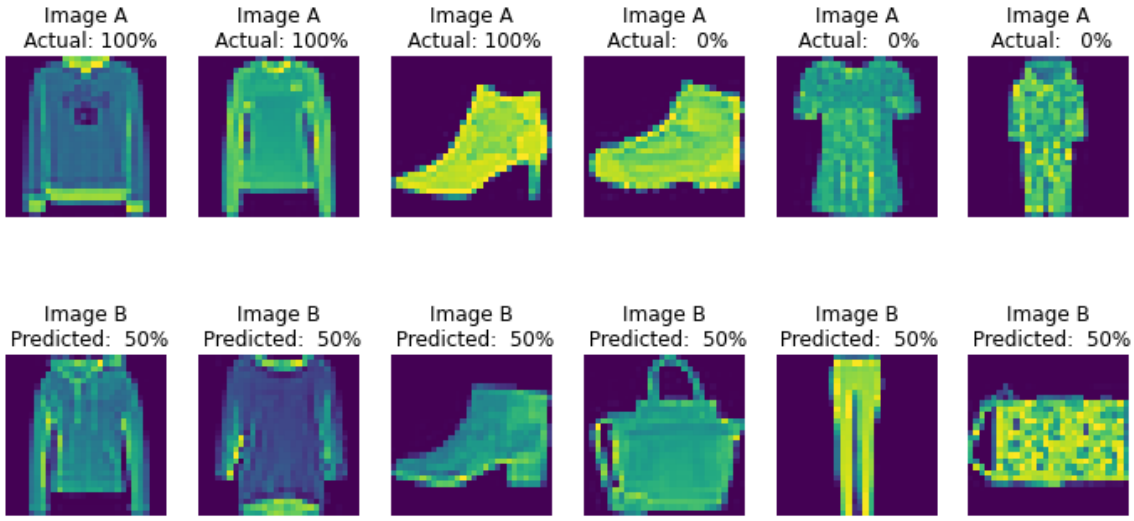

Рис. 2. Результат роботи мережі до тренування

Проведемо тренування та тестування мережі та поглянемо на результати після цього:

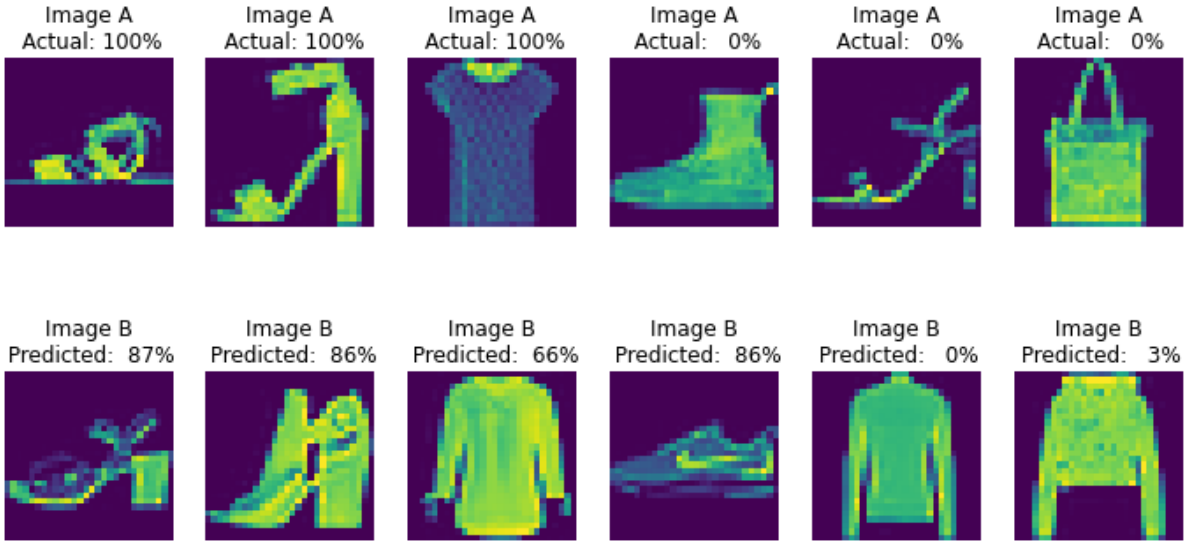

Рис. 3. Результат роботи мережі після тренування

Тепер, застосуємо TSNE, для зменшення розмірності отриманих векторів фіч, та візуалізуємо отримані кластери (рис 4).

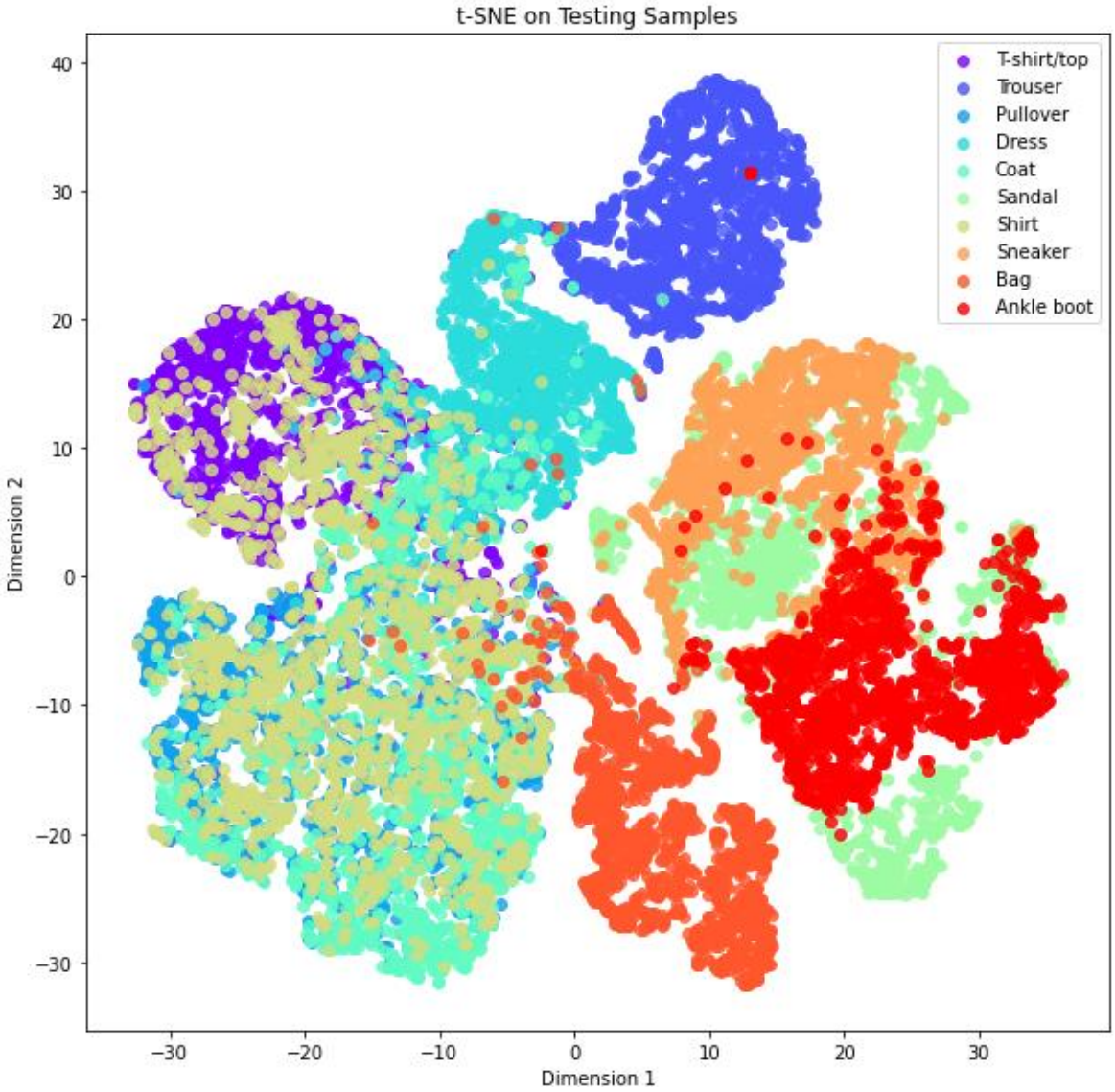

Рис. 4. Візуалізація отриманих кластерів

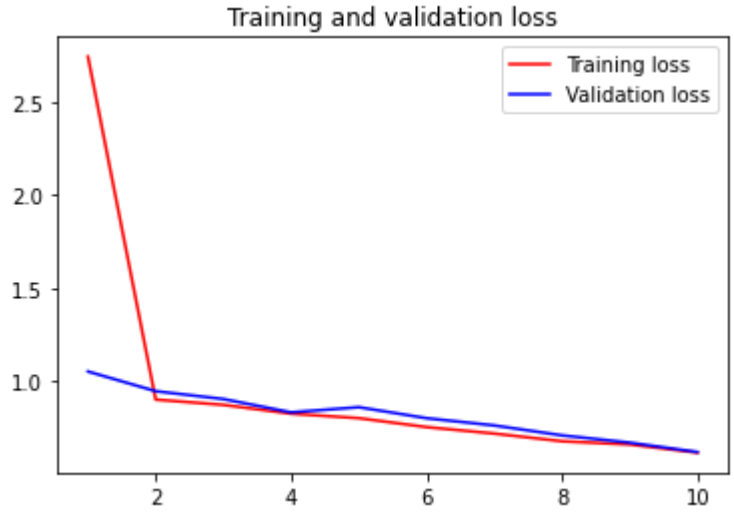

Рис. 5 Результати тренування мережі на основі ResNeXt-50

Висновок: у ході виконання даної лабораторної роботи я набув практичних навіків у розв’язанні задачі пошуку подібних зображень на прикладі організації CNN класифікації.

Код програми

lab3_1.py

```
import numpy as np
import os
import pandas as pd
from keras.preprocessing.image import ImageDataGenerator
from keras.utils.np_utils import to_categorical
import matplotlib.pyplot as plt

from google.colab import drive
drive.mount('/content/drive')

from sklearn.model_selection import train_test_split
data_train = pd.read_csv('/content/drive/My Drive/fashion-mnist_train.csv')
X_full = data_train.iloc[:,1:]
y_full = data_train.iloc[:,:1]
x_train, x_test, y_train, y_test = train_test_split(X_full, y_full,
test_size=0.3)

x_train = x_train.values.reshape(-1, 28, 28, 1).astype('float32') / 255.
x_test = x_test.values.reshape(-1, 28, 28, 1).astype('float32') / 255.
y_train = y_train.values.astype('int')
y_test = y_test.values.astype('int')
print('Training', x_train.shape, x_train.max())
print('Testing', x_test.shape, x_test.max())

# reorganize by groups
train_groups = [x_train[np.where(y_train==i)[0]] for i in np.unique(y_train)]
test_groups = [x_test[np.where(y_test==i)[0]] for i in np.unique(y_train)]
print('train groups:', [x.shape[0] for x in train_groups])
print('test groups:', [x.shape[0] for x in test_groups])

def gen_random_batch(in_groups, batch_halfsize=8):
    out_img_a, out_img_b, out_score = [], [], []
    all_groups = list(range(len(in_groups)))
    for match_group in [True, False]:
        group_idx = np.random.choice(all_groups, size=batch_halfsize)
        print(group_idx)
        out_img_a +=
[in_groups[c_idx][np.random.choice(range(in_groups[c_idx].shape[0]))] for
c_idx in group_idx]
        if match_group:
            b_group_idx = group_idx
            out_score += [1]*batch_halfsize
        else:
            # anything but the same group
            non_group_idx = [np.random.choice([i for i in all_groups if i !=
c_idx]) for c_idx in group_idx]
            b group idx = non group idx
            out_score += [0]*batch_halfsize
        out_img_b +=
[in_groups[c_idx][np.random.choice(range(in_groups[c_idx].shape[0]))] for
c_idx in b_group_idx]
    return np.stack(out_img_a,0), np.stack(out_img_b,0),
```

```
np.stack(out_score,0)

pv_a, pv_b, pv_sim = gen_random_batch(train_groups, 3)
fig, m_axs = plt.subplots(2, pv_a.shape[0], figsize=(12, 6))
for c_a, c_b, c_d, (ax1, ax2) in zip(pv_a, pv_b, pv_sim, m_axs.T):
    ax1.imshow(c_a[:,:,0])
    ax1.set_title('Image A')
    ax1.axis('off')
    ax2.imshow(c_b[:,:,0])
    ax2.set_title('Image B\n Similarity: %3.0f%%' % (100*c_d))
    ax2.axis('off')

from keras.models import Model
from keras.layers import Input, Conv2D, BatchNormalization, MaxPool2D,
Activation, Flatten, Dense, Dropout

img_in = Input(shape=x_train.shape[1:], name='FeatureNet_ImageInput')
n_layer = img_in
for i in range(2):
    n_layer = Conv2D(8*2**i, kernel_size=(3,3), activation='linear')(n_layer)
    n_layer = BatchNormalization()(n_layer)
    n_layer = Activation('relu')(n_layer)
    n_layer = Conv2D(16*2**i, kernel_size=(3,3),
activation='linear')(n_layer)
    n_layer = BatchNormalization()(n_layer)
    n_layer = Activation('relu')(n_layer)
    n_layer = MaxPool2D((2,2))(n_layer)
n_layer = Flatten()(n_layer)
n_layer = Dense(32, activation='linear')(n_layer)
n_layer = Dropout(0.5)(n_layer)
n_layer = BatchNormalization()(n_layer)
n_layer = Activation('relu')(n_layer)
feature_model = Model(inputs=[img_in], outputs=[n_layer],
name='FeatureGenerationModel')
feature_model.summary()

from keras.layers import concatenate
img_a_in = Input(shape=x_train.shape[1:], name='ImageA_Input')
img_b_in = Input(shape=x_train.shape[1:], name='ImageB_Input')
img_a_feat = feature_model(img_a_in)
img_b_feat = feature_model(img_b_in)
combined_features = concatenate([img_a_feat, img_b_feat],
name='merge_features')
combined_features = Dense(16, activation='linear')(combined_features)
combined_features = BatchNormalization()(combined_features)
combined_features = Activation('relu')(combined_features)
combined_features = Dense(4, activation='linear')(combined_features)
combined_features = BatchNormalization()(combined_features)
combined_features = Activation('relu')(combined_features)
combined_features = Dense(1, activation='sigmoid')(combined_features)
similarity_model = Model(inputs=[img_a_in, img_b_in],
outputs=[combined_features], name='Similarity_Model')
similarity_model.summary()

# setup the optimization process
similarity_model.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['mae'])

def find_simmilar(dataset, image, n_candidates=1024, finalists=4):
    candidates, preds = [], []
    for i in range(n_candidates):
        candidates.append(dataset[np.random.choice(range(len(dataset)))])
    compiled = [np.tile(image, (n_candidates, 1, 1, 1)),
np.stack(candidates)]
```

```
    print(compiled[0].shape, compiled[1].shape)
    preds = similarity_model.predict(compiled).reshape(-1)
    print(preds.shape)
    top = np.argpartition(preds, -finalists)[-finalists:]
    for i in top:
        plt.imshow(candidates[i][:, :, 0])
        plt.show()
        print(preds[i])

def show_model_output(nb_examples = 3):
    pv_a, pv_b, pv_sim = gen_random_batch(test_groups, nb_examples)
    pred_sim = similarity_model.predict([pv_a, pv_b])
    fig, m_axs = plt.subplots(2, pv_a.shape[0], figsize=(12, 6))
    for c_a, c_b, c_d, p_d, (ax1, ax2) in zip(pv_a, pv_b, pv_sim, pred_sim,
m_axs.T):
        ax1.imshow(c_a[:,:,0])
        ax1.set_title('Image A\n Actual: %3.0f%%' % (100*c_d))
        ax1.axis('off')
        ax2.imshow(c_b[:,:,0])
        ax2.set_title('Image B\n Predicted: %3.0f%%' % (100*p_d))
        ax2.axis('off')
    return fig
# a completely untrained model
_ = show_model_output()

def siam_gen(in_groups, batch_size=32):
    while True:
        pv_a, pv_b, pv_sim = gen_random_batch(train_groups, batch_size//2)
        yield [pv_a, pv_b], pv_sim

valid_a, valid_b, valid_sim = gen_random_batch(test_groups, 1024)
loss_history = similarity_model.fit(
    siam_gen(train_groups),
    steps_per_epoch=500,
    validation_data=([valid_a, valid_b], valid_sim),
    epochs=10,
    verbose=True
)

_ = show_model_output()

image = x_train[999]

plt.imshow(image[:, :, 0])

find_simmilar(x_train, image)

t_shirt_vec = np.stack([train_groups[0][0]]*x_test.shape[0], 0)
t_shirt_score = similarity_model.predict([t_shirt_vec, x_test], verbose=True,
batch_size=128)
ankle_boot_vec = np.stack([train_groups[-1][0]]*x_test.shape[0], 0)
ankle_boot_score = similarity_model.predict([ankle_boot_vec, x_test],
verbose=True, batch_size=128)

obj_categories = [
    'T-shirt/top','Trouser','Pullover','Dress',
    'Coat','Sandal','Shirt','Sneaker','Bag','Ankle boot'
]
colors = plt.cm.rainbow(np.linspace(0, 1, 10))
plt.figure(figsize=(10, 10))
for c_group, (c_color, c_label) in enumerate(zip(colors, obj_categories)):
    plt.scatter(
        t_shirt_score[np.where(y_test == c_group), 0],
        ankle_boot_score[np.where(y_test == c_group), 0],
```

```
        marker='.',
        color=c_color,
        linewidth='1',
        alpha=0.8,
        label=c_label
    )
plt.xlabel('T-Shirt Dimension')
plt.ylabel('Ankle-Boot Dimension')
plt.title('T-Shirt and Ankle-Boot Dimension')
plt.legend(loc='best')
plt.savefig('tshirt-boot-dist.png')
plt.show(block=False)

x_test_features = feature_model.predict(
    x_test,
    verbose=True,
    batch_size=128
)

# Commented out IPython magic to ensure Python compatibility.
# %%time
# from sklearn.manifold import TSNE
# tsne_obj = TSNE(
#     n_components=2,
#     init='pca',
#     random_state=101,
#     method='barnes_hut',
#     n_iter=500,
#     verbose=2
# )
# tsne_features = tsne_obj.fit_transform(x_test_features)

obj_categories = [
    'T-shirt/top','Trouser','Pullover','Dress',
    'Coat','Sandal','Shirt','Sneaker','Bag','Ankle boot'
]
colors = plt.cm.rainbow(np.linspace(0, 1, 10))
plt.figure(figsize=(10, 10))

for c_group, (c_color, c_label) in enumerate(zip(colors, obj_categories)):
    plt.scatter(
        tsne_features[np.where(y_test == c_group), 0],
        tsne_features[np.where(y_test == c_group), 1],
        marker='o',
        color=c_color,
        linewidth='1',
        alpha=0.8,
        label=c_label
    )
plt.xlabel('Dimension 1')
plt.ylabel('Dimension 2')
plt.title('t-SNE on Testing Samples')
plt.legend(loc='best')
plt.savefig('clothes-dist.png')
plt.show(block=False)
```

lab3_2.py

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from keras.preprocessing.image import img_to_array, array_to_img
from tensorflow.keras.utils import to_categorical
```

```
from google.colab import drive
drive.mount('/content/drive')

train_data = pd.read_csv('/content/drive/My Drive/fashion-mnist_train.csv')
test_data = pd.read_csv('/content/drive/My Drive/fashion-mnist_test.csv')

train_X = np.array(train_data.iloc[:,1:])
test_X = np.array(test_data.iloc[:,1:])
train_Y = np.array (train_data.iloc[:,0])
test_Y = np.array(test_data.iloc[:,0])

train_X=np.dstack([train_X] * 3)
test_X=np.dstack([test_X] * 3)
train_X.shape, test_X.shape

train_X = train_X.reshape(-1, 28, 28, 3)
test_X= test_X.reshape(-1, 28, 28, 3)
train_X.shape, test_X.shape

train_X = np.asarray([img_to_array(array_to_img(im, scale=False).resize((48,
48))) for im in train_X])
test_X = np.asarray([img_to_array(array_to_img(im, scale=False).resize((48,
48))) for im in test_X])
train_X.shape, test_X.shape

train_X = train_X.astype('float32') / 255.
test_X = test_X.astype('float32') / 255.
train_Y_one_hot = to_categorical(train_Y)
test_Y_one_hot = to_categorical(test_Y)

train_X, valid_X, train_label, valid_label = train_test_split(
    train_X,
    train_Y_one_hot,
    test_size=0.2,
    random_state=13
)

train_X.shape, valid_X.shape, train_label.shape, valid_label.shape

!pip install git+https://github.com/qubvel/classification_models.git

# for keras
from classification_models.keras import Classifiers

# for tensorflow keras
from classification_models.tfkeras import Classifiers

Classifiers.models_names()

SeResNeXT, preprocess_input = Classifiers.get('seresnext50')
model = SeResNeXT(include_top = False, input_shape=(48, 48, 3),
weights='imagenet')
model.summary()

train_features = model.predict(np.array(train_X), batch_size=64, verbose=1)
test_features = model.predict(np.array(test_X), batch_size=64, verbose=1)
val_features = model.predict(np.array(valid_X), batch_size=64, verbose=1)

train_features.shape, test_features.shape, val_features.shape

train_features_flat = np.reshape(train_features, (48000, -1))
test_features_flat = np.reshape(test_features, (10000, -1))
val_features_flat = np.reshape(val_features, (12000, -1))
```

```
from keras import models
from keras import layers
from tensorflow.keras import optimizers
from keras import callbacks
from keras.regularizers import l2

model = models.Sequential()
model.add(layers.Dense(512, activation='relu', kernel_regularizer=l2(0.01)))
model.add(layers.BatchNormalization())
model.add(layers.Activation('relu'))
model.add(layers.Dense(256, activation='relu', kernel_regularizer=l2(0.01)))
model.add(layers.BatchNormalization())
model.add(layers.Activation('relu'))
model.add(layers.Dense(128, activation='relu', kernel_regularizer=l2(0.01)))
model.add(layers.BatchNormalization())
model.add(layers.Activation('relu'))
model.add(layers.Dense(10, activation='softmax'))

model.compile(
    loss='categorical_crossentropy',
    optimizer=optimizers.Adam(),
    metrics=['accuracy']
)

reduce_lr = callbacks.ReduceLROnPlateau(
    factor=0.2,
    patience=2,
    verbose=1,
    cooldown=2
)

history = model.fit(
    train_features_flat,
    train_label,
    epochs=10,
    steps_per_epoch=500,
    validation_data=(val_features_flat, valid_label),
    callbacks=[reduce_lr]
)

import matplotlib.pyplot as plt


loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(accuracy) + 1)



plt.figure()
plt.title('Training and validation loss')
plt.plot(epochs, loss, 'red', label='Training loss')
plt.plot(epochs, val_loss, 'blue', label='Validation loss')

plt.legend()

plt.show()
```