Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)

Department of Computer Science and Engineering (Data Science)

Subject: Image Processing and Computer Vision - II Laboratory (DJ19DSL702)

AY: 2024-25

Experiment 5

(Transfer Learning on Image Classification)

Name:Jhanvi Parekh

SAP ID: 60009210033

Batch: D11

Aim: To compare the performance of different transfer learning strategies on an image classification task.

Theory:

Transfer learning is a machine learning technique where a model developed for one task is reused as the starting point for a model on a second, related task. Instead of training a model from scratch, transfer learning allows leveraging the knowledge a model has gained from a large, diverse dataset (like ImageNet) to apply it to a smaller, more specific dataset. This is especially useful when the new dataset is limited in size or the task is closely related to the original problem. By reusing the learned features from a pre-trained model, transfer learning reduces computational resources, training time, and often improves the performance of the new task. It is widely used in areas like image classification, natural language processing, and speech recognition.

Datasets:

1. Pre-trained model: Use a pre-trained model like VGG16, ResNet50, or MobileNet, trained on ImageNet.
2. Custom dataset: A smaller image dataset specific to the problem (e.g., classifying images of specific objects, animals, or scenes).

Approach:

The experiment will test three different transfer learning strategies:

3. Fine-Tuning the Entire Model: The pre-trained model is fully retrained on the new dataset.
4. Freezing Some Layers: Some layers of the pre-trained model are frozen (not updated), while others are trained on the new dataset.
5. Using the Model as a Feature Extractor: The pre-trained model is used to extract features, and only a new classifier is trained on top.

Experiment Design:

Step 1: Preprocessing the Data

- Preprocess the custom dataset (resize, normalize, augment).
- Split the dataset into training, validation, and test sets.

Step 2: Transfer Learning Methods

Method 1: Fine-Tuning the Entire Model

- Load the pre-trained model (e.g., VGG16 or ResNet50).
- Replace the final classification layer to match the number of classes in your dataset.
- Train all layers of the model using the new dataset.

Method 2: Freezing Some Layers

- Load the pre-trained model.
- Freeze the initial layers (e.g., first 10 layers in a 50-layer network).
- Replace the final classification layer.
- Train only the non-frozen layers on the new dataset.

Method 3: Using the Model as a Feature Extractor

- Load the pre-trained model.
- Freeze all the layers except the final classification layer.
- Replace the final classification layer.
- Train only the classifier on the new dataset.

Step 3: Training

- Train each of the three models using appropriate loss functions (e.g., cross-entropy for classification).
- Use a learning rate schedule or optimizer like Adam or SGD.

Step 4: Evaluation

- Evaluate the models on the test set using metrics like accuracy, precision, recall, and F1-score.
- Record the training time and performance for each method.

Step 5: Analysis

- Compare the performance of the three methods in terms of: Model accuracy on the test set.

Expected Outcomes:

- **Fine-tuning the entire model** might yield the best performance but could be computationally expensive.
- **Freezing some layers** allows faster training while retaining some learning from the pre-trained model.
- **Feature extraction** might be the quickest but may offer slightly lower performance depending on the dataset.

Transfer Learning Applied on MNIST DataSet

```python
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import tensorflow as tf
from sklearn.utils import shuffle
from sklearn.model_selection import cross_val_score
from tensorflow import keras
from tensorflow.keras.wrappers.scikit_learn import KerasClassifier
from tensorflow.keras.layers import *
from tensorflow.keras.models import *
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.callbacks import EarlyStopping

train_data_path = '../input/fashionmnist/fashion-mnist_test.csv'
test_data_path = '../input/fashionmnist/fashion-mnist_test.csv'

train_data = pd.read_csv(train_data_path, dtype=np.float)
test_data = pd.read_csv(test_data_path, dtype=np.float)


# 1 Condition

nine_train = train_data[train_data["label"] !=5]
nine_test= test_data[test_data["label"] != 5]

eight_train = nine_train[nine_train["label"] !=6]
eight_test = nine_test[nine_test["label"] !=6]

#Application TL

five_train = train_data[train_data["label"]==5]
five_test = test_data[test_data["label"]==5]
six_train = train_data[train_data["label"]==6]
six_test = test_data[test_data["label"]==6]

train_frames=[five_train, six_train]
test_frames=[five_test, six_test]

two_train = pd.concat(train_frames)
two_test = pd.concat(test_frames)


full_train = train_data
full_test = test_data
```

```
def info (arg):
    return print("Shape y unique",arg.shape,arg['label'].unique(),"\
n")

#Pre
info(eight_train)
info(eight_test)

#TL
info(two_train)
info(two_test)

#2TL
info(full_train)
info(full_test)
```

```
Shape y unique (8000, 785) [0. 1. 2. 3. 8. 4. 7. 9.]

Shape y unique (8000, 785) [0. 1. 2. 3. 8. 4. 7. 9.]

Shape y unique (2000, 785) [5. 6.]

Shape y unique (2000, 785) [5. 6.]

Shape y unique (10000, 785) [0. 1. 2. 3. 8. 6. 5. 4. 7. 9.]

Shape y unique (10000, 785) [0. 1. 2. 3. 8. 6. 5. 4. 7. 9.]
```

Data Cleansing

```
def barajar (arg):
    return shuffle(arg)

eight_train = barajar(eight_train)
eight_test = barajar(eight_test)

two_train = barajar(two_train)
two_test = barajar(two_test)

full_train = barajar(full_train)
full_test = barajar(full_test)

#Split

def split_data(x,y):
    return x.drop('label', 1)/255, y['label']
```

```
X_train_8, y_train_8 = split_data(eight_train,eight_train)
X_test_8, y_test_8 = split_data(eight_test,eight_test)

X_train_2,y_train_2 =split_data(two_train,two_train)
X_test_2, y_test_2 = split_data(two_test,two_test)

X_train_full,y_train_full=split_data(full_train,full_train)
X_test_full, y_test_full = split_data(full_test,full_test)

X_train_8 = np.array([i.reshape(28,28,1) for i in X_train_8.values])
X_test_8 = np.array([i.reshape(28,28,1) for i in X_test_8.values])
X_train_2 = np.array([i.reshape(28,28,1) for i in X_train_2.values])
X_test_2 = np.array([i.reshape(28,28,1) for i in X_test_2.values])
X_train_full = np.array([i.reshape(28,28,1) for i in
X_train_full.values])
X_test_full = np.array([i.reshape(28,28,1) for i in
X_test_full.values])

def print_trshape (arg):
    return print("Shape de tensor TRAINING: ",arg.shape)
def print_teshape (arg):
    return print("Shape de tensor TESTING: ", arg.shape,"\n")

print_trshape(X_train_8)
print_teshape(X_test_8)
print_trshape(X_train_2)
print_teshape(X_test_2)
print_trshape(X_train_full)
print_teshape(X_test_full)
```

```
Shape de tensor TRAINING:  (8000, 28, 28, 1)
Shape de tensor TESTING:  (8000, 28, 28, 1)

Shape de tensor TRAINING:  (2000, 28, 28, 1)
Shape de tensor TESTING:  (2000, 28, 28, 1)

Shape de tensor TRAINING:  (10000, 28, 28, 1)
Shape de tensor TESTING:  (10000, 28, 28, 1)
```

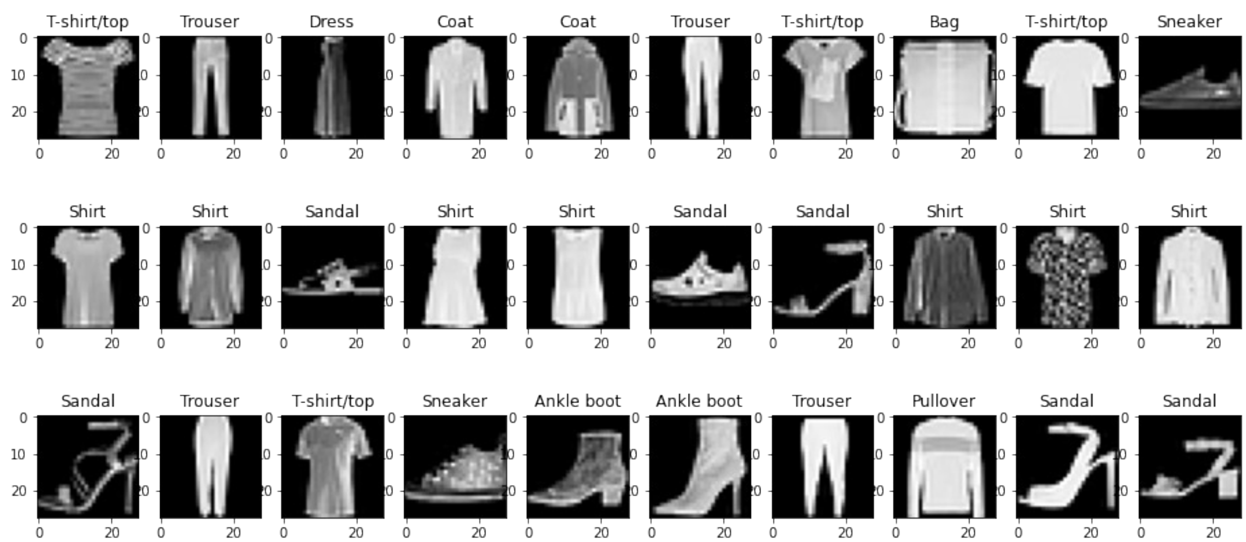
Exploratory Data Analysis

```
class_names =
["T-shirt/top","Trouser","Pullover","Dress","Coat","Sandal",
              "Shirt","Sneaker","Bag","Ankle boot"]

numero_imagenes = 10
```

```
def plot_pics(arg1, arg2):
    fig, axs = plt.subplots(nrows=1, ncols=10, figsize=(16, 8))
    for i in range(numero_imagenes):
        axs[i].imshow(tf.reshape(arg1[i], [28, 28])*255,vmin=0,
vmax=255,cmap='gray')
        axs[i].title.set_text(str(class_names[int(arg2.values[i])]))

plot_pics(X_train_8, y_train_8)
plot_pics(X_train_2,y_train_2)
plot_pics(X_train_full,y_train_full)
```




```
#8 Cat

dic_8={0:0, 1.0:1, 2.0:2, 3.0:3, 4.0:4, 7.0:5, 8.0:6, 9.0:7}
def dic_ev8(a):
    return dic_8[a]

y_train_8 = y_train_8.apply(dic_ev8)
y_test_8 = y_test_8.apply(dic_ev8)

#2 Cat

dic_2={5.0:0, 6.0:1}
def dic_ev2(a):
    return dic_2[a]

y_train_2 = y_train_2.apply(dic_ev2)
y_test_2 = y_test_2.apply(dic_ev2)

def token_create(arg):
    return np.array([np.eye(1,size, int(i))[0] for i in arg.values])
```

```python
size=8
y_train_8 =token_create(y_train_8)
y_test_8 =token_create(y_test_8)
size=2
y_train_2 =token_create(y_train_2)
y_test_2 =token_create(y_test_2)
size=10
y_train_full =token_create(y_train_full)
y_test_full =token_create(y_test_full)

def forma(arg):
    return print(arg.shape)

forma(y_train_8)
forma(y_test_8)
forma(y_train_2)
forma(y_test_2)
forma(y_train_full)
forma(y_test_full)
```

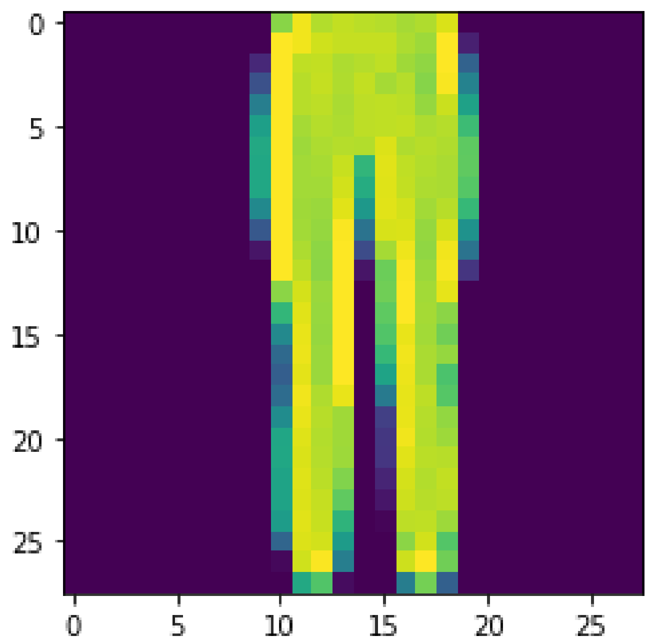
```
(8000, 8)
(8000, 8)
(2000, 2)
(2000, 2)
(10000, 10)
(10000, 10)
```

```python
print(y_train_8[5])

plt.imshow(X_train_8[5])
```

```
[0. 1. 0. 0. 0. 0. 0. 0.]
```

```
<matplotlib.image.AxesImage at 0x7978d5b8a550>
```



Model Building

```
#8 Categories
#AlexNet Architecture:

#Input Layer

model_input = Input(shape=(28,28,1))

#Block 1

x = Conv2D(filters=32, kernel_size=(3,3), strides=(2,2),
activation='relu', padding= 'same')(model_input)
x = Conv2D(filters=32, kernel_size=(3,3), activation='relu',
padding="same")(x)
x = BatchNormalization()(x)
x = MaxPool2D((3,3), strides=(2,2))(x)

#Block 3

x = Conv2D(filters=64, kernel_size=(3,3), strides=(1,1),
activation='relu', padding="same")(x)
x = Conv2D(filters=64, kernel_size=(3,3), strides=(1,1),
activation='relu', padding="same")(x)
x = BatchNormalization()(x)
x = MaxPool2D((2,2), strides=(2,2))(x)

#Block 4
```

```
flat = Flatten()(x)
y = Dense(512, activation='relu')(flat)
y = Dropout(0.3)(y)
y = Dense(128, activation='relu')(y)
y = Dense(128, activation='relu')(y)
y = Dropout(0.3)(y)
trainable = Dense(128, activation='relu')(y)

# Output Layer

output1 = Dense(8, activation='softmax')(trainable)

model = Model(inputs = model_input, outputs = output1)

print(model.summary())
```

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
input_1 (InputLayer)         [(None, 28, 28, 1)]       0         
_________________________________________________________________
conv2d (Conv2D)              (None, 14, 14, 32)        320       
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 14, 14, 32)        9248      
_________________________________________________________________
batch_normalization (BatchNo (None, 14, 14, 32)        128       
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 6, 6, 32)          0         
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 6, 6, 64)          18496     
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 6, 6, 64)          36928     
_________________________________________________________________
batch_normalization_1 (Batch (None, 6, 6, 64)          256       
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 3, 3, 64)          0         
_________________________________________________________________
flatten (Flatten)            (None, 576)               0         
_________________________________________________________________
dense (Dense)                (None, 512)               295424    
_________________________________________________________________
dropout (Dropout)            (None, 512)               0         
_________________________________________________________________
dense_1 (Dense)              (None, 128)               65664     
_________________________________________________________________
dense_2 (Dense)              (None, 128)               16512     
_________________________________________________________________
dropout_1 (Dropout)          (None, 128)               0         
_________________________________________________________________
```

```
dense_3 (Dense)              (None, 128)               16512
_________________________________________________________________
dense_4 (Dense)              (None, 8)                 1032
=================================================================
Total params: 460,520
Trainable params: 460,328
Non-trainable params: 192
_________________________________________________________________
None


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


print("weights : ",len(model.weights))
print("trainable weights: ", len(model.trainable_weights))
print("non trainable weights: ",len(model.non_trainable_weights))

weights :  26
trainable weights:  22
non trainable weights:  4


early_stopping = EarlyStopping(monitor='loss',patience=2)


model.fit(X_train_8, y_train_8, epochs=10,callbacks=[early_stopping],
validation_data=(X_test_8, y_test_8))

Epoch 1/10
250/250 [==============================] - 10s 35ms/step - loss:
1.1768 - accuracy: 0.5790 - val_loss: 1.8343 - val_accuracy: 0.7205
Epoch 2/10
250/250 [==============================] - 8s 32ms/step - loss: 0.4516
- accuracy: 0.8436 - val_loss: 0.5476 - val_accuracy: 0.8839
Epoch 3/10
250/250 [==============================] - 8s 33ms/step - loss: 0.3539
- accuracy: 0.8755 - val_loss: 0.2540 - val_accuracy: 0.9125
Epoch 4/10
250/250 [==============================] - 8s 33ms/step - loss: 0.2966
- accuracy: 0.8986 - val_loss: 0.2621 - val_accuracy: 0.9021
Epoch 5/10
250/250 [==============================] - 8s 31ms/step - loss: 0.2637
- accuracy: 0.9082 - val_loss: 0.2120 - val_accuracy: 0.9249
Epoch 6/10
250/250 [==============================] - 8s 30ms/step - loss: 0.2306
- accuracy: 0.9187 - val_loss: 0.1813 - val_accuracy: 0.9350
Epoch 7/10
250/250 [==============================] - 8s 31ms/step - loss: 0.2265
- accuracy: 0.9158 - val_loss: 0.2204 - val_accuracy: 0.9166
```

```
Epoch 8/10
250/250 [==============================] - 8s 33ms/step - loss: 0.2043
- accuracy: 0.9265 - val_loss: 0.1755 - val_accuracy: 0.9369
Epoch 9/10
250/250 [==============================] - 8s 31ms/step - loss: 0.1881
- accuracy: 0.9291 - val_loss: 0.1465 - val_accuracy: 0.9466
Epoch 10/10
250/250 [==============================] - 8s 31ms/step - loss: 0.1773
- accuracy: 0.9376 - val_loss: 0.1366 - val_accuracy: 0.9506

<tensorflow.python.keras.callbacks.History at 0x7978d5a30190>

accuracy= model.evaluate(X_test_8,y_test_8)[1]
print ('Model accuracy:' ,accuracy*100, '%')

250/250 [==============================] - 2s 7ms/step - loss: 0.1366
- accuracy: 0.9506
Model accuracy: 95.06250023841858 %
```

- Transfer Learning through Concat Set Up

```
model.trainable=False

#####

input_1 = keras.layers.Input(shape = (28,28,1))

model_con = model(input_1)

layer_2 = keras.layers.Dense(2, activation='relu')(model_con)

concatenate = keras.layers.concatenate([model_con, layer_2])

output = keras.layers.Dense(10, activation='softmax')(concatenate)

top_model = keras.models.Model(inputs = [input_1],outputs = [output])

top_model.summary()

Model: "model_1"
______________________________________________________________________
____________________________
Layer (type)                    Output Shape         Param #
Connected to
======================================================================
============================
input_2 (InputLayer)            [(None, 28, 28, 1)]  0

______________________________________________________________________
```

```
____________________________
model (Functional)              (None, 8)            460520
input_2[0][0]
______________________________________________________________________
____________________________
dense_5 (Dense)                 (None, 2)            18
model[0][0]
______________________________________________________________________
____________________________
concatenate (Concatenate)       (None, 10)           0
model[0][0]

dense_5[0][0]
______________________________________________________________________
____________________________
dense_6 (Dense)                 (None, 10)           110
concatenate[0][0]
======================================================================
============================
Total params: 460,648
Trainable params: 128
Non-trainable params: 460,520
______________________________________________________________________
____________________________
```

```
opt = tf.keras.optimizers.SGD(learning_rate=0.001)
top_model.compile(optimizer=opt, loss='categorical_crossentropy',
metrics=['accuracy'])
```

```
print("weights : ",len(top_model.weights))
print("trainable weights: ", len(top_model.trainable_weights))
print("non trainable weights: ",len(top_model.non_trainable_weights))
```

```
weights :  30
trainable weights:  4
non trainable weights:  26
```

```
history1 = top_model.fit(X_train_full, y_train_full, epochs=15,
validation_data=(X_test_full, y_test_full))
```

```
Epoch 1/15
313/313 [==============================] - 6s 16ms/step - loss: 2.5363
- accuracy: 0.0314 - val_loss: 2.5205 - val_accuracy: 0.0346
Epoch 2/15
313/313 [==============================] - 5s 16ms/step - loss: 2.5125
- accuracy: 0.0422 - val_loss: 2.4959 - val_accuracy: 0.0443
Epoch 3/15
313/313 [==============================] - 5s 15ms/step - loss: 2.4943
```

```
- accuracy: 0.0465 - val_loss: 2.4717 - val_accuracy: 0.0671
Epoch 4/15
313/313 [==============================] - 5s 15ms/step - loss: 2.4689
- accuracy: 0.0934 - val_loss: 2.4476 - val_accuracy: 0.1200
Epoch 5/15
313/313 [==============================] - 5s 15ms/step - loss: 2.4469
- accuracy: 0.1132 - val_loss: 2.4238 - val_accuracy: 0.1220
Epoch 6/15
313/313 [==============================] - 5s 14ms/step - loss: 2.4202
- accuracy: 0.1183 - val_loss: 2.4001 - val_accuracy: 0.1493
Epoch 7/15
313/313 [==============================] - 5s 15ms/step - loss: 2.3934
- accuracy: 0.1455 - val_loss: 2.3767 - val_accuracy: 0.1524
Epoch 8/15
313/313 [==============================] - 5s 14ms/step - loss: 2.3673
- accuracy: 0.1535 - val_loss: 2.3538 - val_accuracy: 0.1545
Epoch 9/15
313/313 [==============================] - 5s 15ms/step - loss: 2.3537
- accuracy: 0.1481 - val_loss: 2.3312 - val_accuracy: 0.1563
Epoch 10/15
313/313 [==============================] - 5s 15ms/step - loss: 2.3269
- accuracy: 0.1510 - val_loss: 2.3086 - val_accuracy: 0.1581
Epoch 11/15
313/313 [==============================] - 5s 15ms/step - loss: 2.3057
- accuracy: 0.1560 - val_loss: 2.2861 - val_accuracy: 0.1593
Epoch 12/15
313/313 [==============================] - 5s 15ms/step - loss: 2.2841
- accuracy: 0.1553 - val_loss: 2.2637 - val_accuracy: 0.1604
Epoch 13/15
313/313 [==============================] - 5s 15ms/step - loss: 2.2632
- accuracy: 0.1567 - val_loss: 2.2414 - val_accuracy: 0.1612
Epoch 14/15
313/313 [==============================] - 5s 15ms/step - loss: 2.2417
- accuracy: 0.1588 - val_loss: 2.2191 - val_accuracy: 0.1619
Epoch 15/15
313/313 [==============================] - 5s 15ms/step - loss: 2.2184
- accuracy: 0.1573 - val_loss: 2.1969 - val_accuracy: 0.1627
```

```python
accuracy= top_model.evaluate(X_test_full,y_test_full)[1]
print ('Model accuracy:' ,accuracy*100, '%')
```

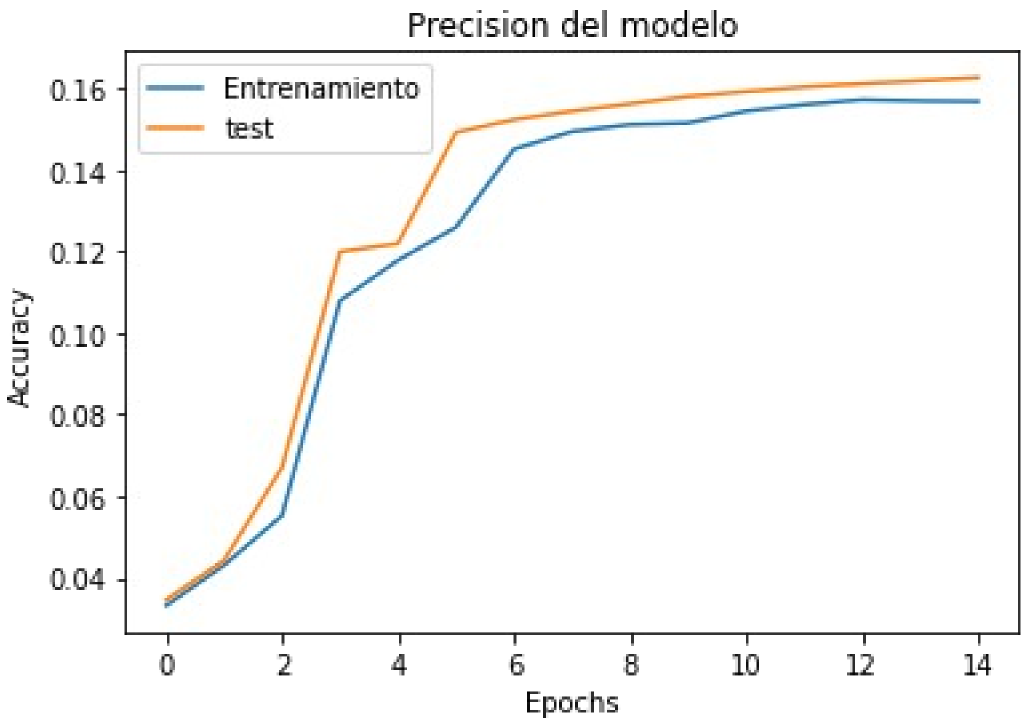
```
313/313 [==============================] - 2s 8ms/step - loss: 2.1969
- accuracy: 0.1627
Model accuracy: 16.269999742507935 %
```

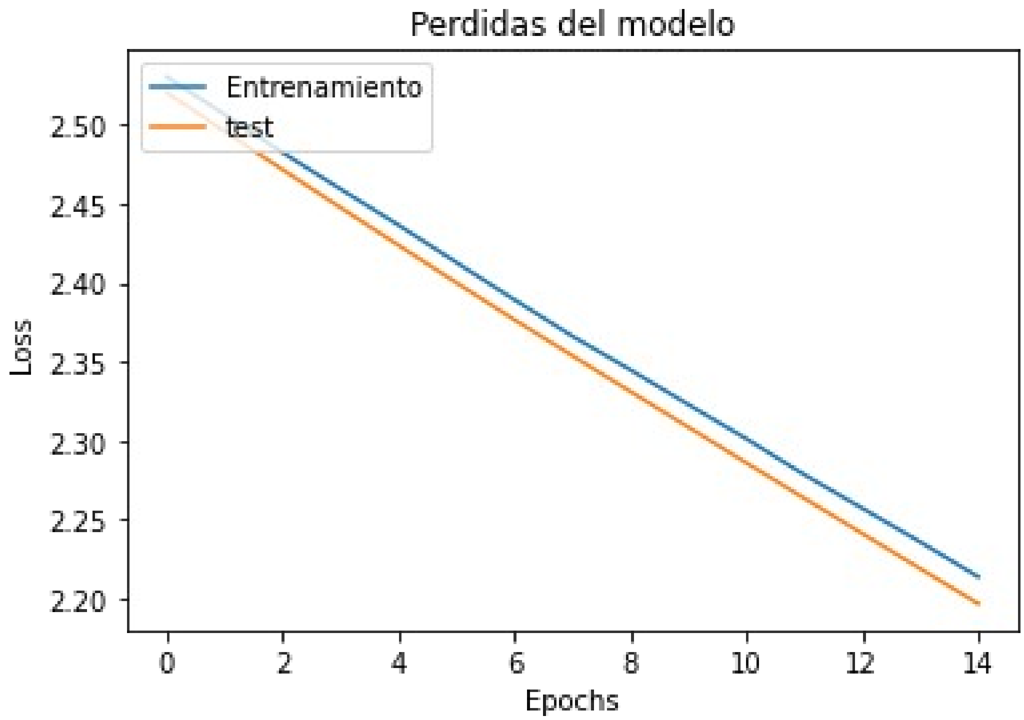
```python
plt.plot(history1.history['accuracy'])
plt.plot(history1.history['val_accuracy'])
plt.title('Precision del modelo')
```

```
plt.ylabel('Accuracy')
plt.xlabel('Epochs')
plt.legend(['Entrenamiento', 'test'], loc='upper left')
plt.show()



plt.plot(history1.history['loss'])
plt.plot(history1.history['val_loss'])
plt.title('Perdidas del modelo')
plt.ylabel('Loss')
plt.xlabel('Epochs')
plt.legend(['Entrenamiento', 'test'], loc='upper left')
plt.show()
```





Conclusion

```
predictions = model.predict(X_test_8)
for i in range (10):
    print(" 0 Tshirt 1 Trouser 2 Pullover 3 Dress 4 Coat 5 Shirt 6
Sneaker 7 Bag 8 Ankle Boot 9 Sandals")
    print(predictions[i])
    print(y_test_8[i])
    print (" \n ")

 0 Tshirt 1 Trouser 2 Pullover 3 Dress 4 Coat 5 Shirt 6 Sneaker 7 Bag
8 Ankle Boot 9 Sandals
[4.1879010e-03 3.6924467e-07 9.9557269e-01 8.7664412e-06 2.1941577e-04
 7.4486650e-08 3.5693952e-06 7.1753720e-06]
[0. 0. 1. 0. 0. 0. 0. 0.]


 0 Tshirt 1 Trouser 2 Pullover 3 Dress 4 Coat 5 Shirt 6 Sneaker 7 Bag
8 Ankle Boot 9 Sandals
[3.8317103e-06 5.0826850e-07 2.0940690e-06 1.8621617e-05 9.0859558e-08
 9.9958068e-01 4.5800302e-06 3.8962424e-04]
[0. 0. 0. 0. 0. 1. 0. 0.]


 0 Tshirt 1 Trouser 2 Pullover 3 Dress 4 Coat 5 Shirt 6 Sneaker 7 Bag
8 Ankle Boot 9 Sandals
```

```
[3.5849368e-04 1.3277645e-04 1.1121876e-02 3.8090914e-03 9.8380369e-01
 1.7560505e-04 4.2745526e-04 1.7093636e-04]
[0. 0. 0. 0. 1. 0. 0. 0.]


 0 Tshirt 1 Trouser 2 Pullover 3 Dress 4 Coat 5 Shirt 6 Sneaker 7 Bag
8 Ankle Boot 9 Sandals
[2.0298962e-03 1.7920752e-04 9.9443518e-02 9.6661324e-04 8.7917739e-01
 2.9140810e-04 1.7401733e-02 5.1026914e-04]
[0. 0. 0. 0. 1. 0. 0. 0.]


 0 Tshirt 1 Trouser 2 Pullover 3 Dress 4 Coat 5 Shirt 6 Sneaker 7 Bag
8 Ankle Boot 9 Sandals
[1.2607711e-07 9.9999857e-01 8.6619056e-10 1.2353161e-06 5.4990366e-08
 9.1890406e-09 7.8619664e-08 1.1652273e-08]
[0. 1. 0. 0. 0. 0. 0. 0.]


 0 Tshirt 1 Trouser 2 Pullover 3 Dress 4 Coat 5 Shirt 6 Sneaker 7 Bag
8 Ankle Boot 9 Sandals
[6.7940498e-08 9.9999940e-01 4.2110809e-10 4.9116670e-07 3.3530629e-08
 2.4040985e-09 2.5782681e-08 6.0060259e-09]
[0. 1. 0. 0. 0. 0. 0. 0.]


 0 Tshirt 1 Trouser 2 Pullover 3 Dress 4 Coat 5 Shirt 6 Sneaker 7 Bag
8 Ankle Boot 9 Sandals
[4.3800928e-02 2.0092788e-04 9.4061357e-01 3.4366122e-03 1.0368619e-02
 6.8375375e-05 1.3266193e-03 1.8443525e-04]
[0. 0. 1. 0. 0. 0. 0. 0.]


 0 Tshirt 1 Trouser 2 Pullover 3 Dress 4 Coat 5 Shirt 6 Sneaker 7 Bag
8 Ankle Boot 9 Sandals
[9.9725788e-05 1.2327445e-05 5.0804980e-02 1.0772609e-04 9.4821662e-01
 2.3845470e-05 6.1404979e-04 1.2067027e-04]
[0. 0. 0. 0. 1. 0. 0. 0.]


 0 Tshirt 1 Trouser 2 Pullover 3 Dress 4 Coat 5 Shirt 6 Sneaker 7 Bag
8 Ankle Boot 9 Sandals
[1.7981726e-08 2.1510833e-07 2.8274809e-07 5.8681014e-08 5.8893832e-08
 2.5087883e-04 1.2848589e-07 9.9974841e-01]
[0. 0. 0. 0. 0. 0. 0. 1.]


 0 Tshirt 1 Trouser 2 Pullover 3 Dress 4 Coat 5 Shirt 6 Sneaker 7 Bag
8 Ankle Boot 9 Sandals
```

```
[2.1285855e-03 9.9134630e-01 1.6072961e-04 5.5585783e-03 3.7278386e-04
 1.7664653e-04 1.6110239e-04 9.5323921e-05]
[0. 1. 0. 0. 0. 0. 0. 0.]
```

```python
predictions = top_model.predict(X_test_full)
for i in range (10):
    print(" 0 Tshirt 1 Trouser 2 Pullover 3 Dress 4 Coat 5 Shirt 6 
Sneaker 7 Bag 8 Ankle Boot 9 Sandals")
    print(predictions[i])
    print(y_test_full[i])
    print (" \n ")
```

```
 0 Tshirt 1 Trouser 2 Pullover 3 Dress 4 Coat 5 Shirt 6 Sneaker 7 Bag
8 Ankle Boot 9 Sandals
[0.09444856 0.13754228 0.09565465 0.07370472 0.09776801 0.09297951
 0.10284187 0.1247879  0.080479   0.09979356]
[0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]


 0 Tshirt 1 Trouser 2 Pullover 3 Dress 4 Coat 5 Shirt 6 Sneaker 7 Bag
8 Ankle Boot 9 Sandals
[0.09464802 0.1407929  0.09446929 0.07448235 0.10105004 0.08976121
 0.10083769 0.12978604 0.07723617 0.09693629]
[0. 0. 0. 0. 0. 0. 1. 0. 0. 0.]


 0 Tshirt 1 Trouser 2 Pullover 3 Dress 4 Coat 5 Shirt 6 Sneaker 7 Bag
8 Ankle Boot 9 Sandals
[0.0898374  0.1014533  0.10785203 0.14454204 0.12257438 0.07086245
 0.0991917  0.12202717 0.07819972 0.06345982]
[0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]


 0 Tshirt 1 Trouser 2 Pullover 3 Dress 4 Coat 5 Shirt 6 Sneaker 7 Bag
8 Ankle Boot 9 Sandals
[0.09481494 0.14163665 0.09435236 0.07378712 0.10166994 0.08945444
 0.10003187 0.13064131 0.0764694  0.09714194]
[0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]


 0 Tshirt 1 Trouser 2 Pullover 3 Dress 4 Coat 5 Shirt 6 Sneaker 7 Bag
8 Ankle Boot 9 Sandals
[0.09264627 0.10272601 0.10013771 0.06495832 0.06400622 0.10204131
 0.10365047 0.08223435 0.15607063 0.13152872]
[0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]


 0 Tshirt 1 Trouser 2 Pullover 3 Dress 4 Coat 5 Shirt 6 Sneaker 7 Bag
```

```
8 Ankle Boot 9 Sandals
[0.0685596  0.09278631 0.13562167 0.1874693  0.08323913 0.08584075
 0.09634583 0.10970314 0.08268142 0.05775287]
[0. 0. 0. 1. 0. 0. 0. 0. 0. 0.]


 0 Tshirt 1 Trouser 2 Pullover 3 Dress 4 Coat 5 Shirt 6 Sneaker 7 Bag
8 Ankle Boot 9 Sandals
[0.07737144 0.09301464 0.13265589 0.10807527 0.06461428 0.16534959
 0.05837079 0.07918883 0.12065062 0.10070863]
[0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]


 0 Tshirt 1 Trouser 2 Pullover 3 Dress 4 Coat 5 Shirt 6 Sneaker 7 Bag
8 Ankle Boot 9 Sandals
[0.08580378 0.11558326 0.07673448 0.11246651 0.0711045  0.06714806
 0.13618314 0.11843555 0.14186426 0.07467651]
[0. 0. 0. 0. 0. 0. 1. 0. 0. 0.]


 0 Tshirt 1 Trouser 2 Pullover 3 Dress 4 Coat 5 Shirt 6 Sneaker 7 Bag
8 Ankle Boot 9 Sandals
[0.08217502 0.11128508 0.07109394 0.13100885 0.0685946  0.06063852
 0.14774048 0.13119254 0.13387111 0.0623998 ]
[1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]


 0 Tshirt 1 Trouser 2 Pullover 3 Dress 4 Coat 5 Shirt 6 Sneaker 7 Bag
8 Ankle Boot 9 Sandals
[0.15877844 0.10651278 0.07198338 0.05941241 0.11578288 0.0692144
 0.09127741 0.08883485 0.11762045 0.12058301]
[0. 0. 0. 0. 0. 0. 0. 0. 1. 0.]
```