Machine Learning Assignment

Clothing Image Classifier

C.N Samarasekara
IT17126816

Contents

Introduction 2

Setting up the Environment 2

The Dataset 3

Training Data 4

Creating and Compiling the Model 5

Training the Model 6

Feed the model 6

Evaluate accuracy 7

Putting the Created Model to Use 7

Introduction

This document was prepared to act as a brief guide for the assignment which was prepared on behalf of the Machine Learning for Cyber Security module. The machine learning algorithm which was chosen to be part of the assignment revolves around neural networks. A neural network which is used to classify images of various clothes.

Setting up the Environment

PyCharm was used as the IDE for programming the neural network. Apart from this Anaconda needs to be downloaded and installed. Once Anaconda is installed open the command prompt to install TensorFlow using the *pip install tensorflow* command. Before installing tensorflow conda needs to work for that it must be set to path and then the conda environment has to be installed using the *create -n [name] python-3.6*. Once the environment is set enter the environment using the *activate [environment name]* command and install tensorflow as stated above. Another package which is essential in this scenario is keras which can be installed using the *pip install keras* command. Main package requirements are done additional packages can be downloaded likewise or through the terminal in PyCharm.

Now open PyCharm set up a new project configure Project Interpreter and Configuration path for the python file accordingly and install required additional packages. For this algorithm we require both numpy and matplotlib both of which can be downloaded using the command *pip install* as a prefix in the conda environment.

The Dataset

In the python file start off by importing the required packages.

```
import tensorflow as tf
from tensorflow import keras
import numpy as np
import matplotlib.pyplot as plt
```

tensorflow - is a library for scientific computing and machine learning in Python.

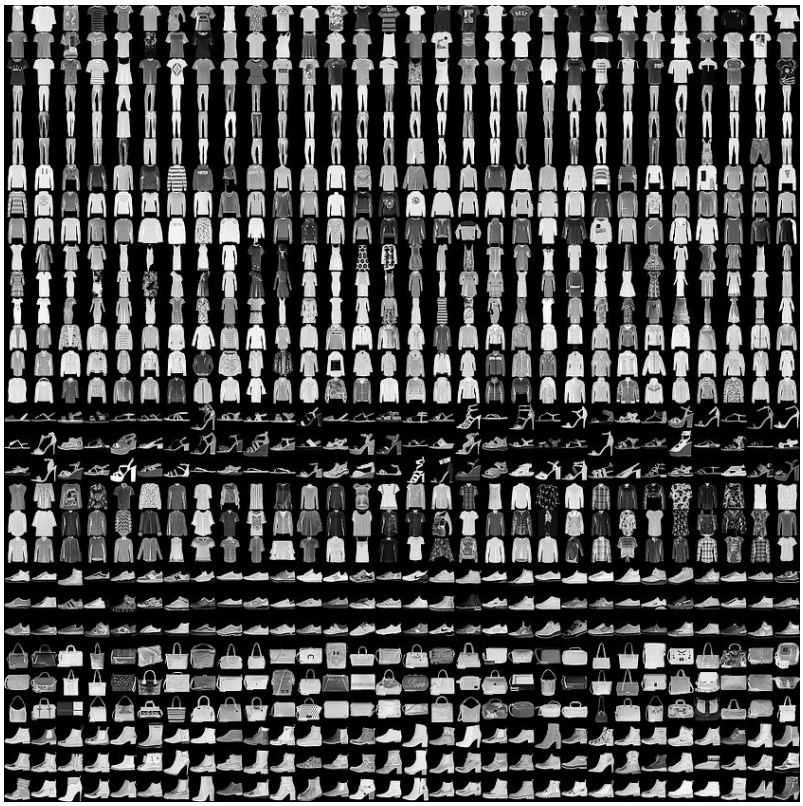
keras - is an API for tensor flow which essentially just allows us to write less code.

numpy - is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices

matplotlib - is a nice library for graphing and showing images and different information.

Import the dataset using keras

```
data = keras.datasets.fashion_mnist
```



Training Data

It is important to split our dataset into training and testing sets, respectively. This enables us to train the algorithm using a specific set of data while the rest can be used to test the prediction accuracy of the algorithm.

```
(train_images, train_labels), (test_images, test_labels) = data.load_data()
```

Loading the dataset returns four NumPy arrays:

The train_images and train_labels arrays are the *training set*—the data the model uses to learn.

The model is tested against the test set, the test_images, and test_labels arrays.

The images are 28x28 NumPy arrays, with pixel values ranging from 0 to 255. The labels are an array of integers, ranging from 0 to 9. These correspond to the *class* of clothing the image represents:

LABEL	CLASS
0	T-Shirt/Top
1	Trouser
2	Pullover
3	Dress
4	Coat
5	Sandal
6	Shirt
7	Sneaker

8	Bag
9	Ankle Boot

Each image is mapped to a single label. Since the *class names* are not included with the dataset, store them here to use later when plotting the images:

```
class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',
               'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']
```

The data must be preprocessed before training the network. If you inspect the first image in the training set, you will see that the pixel values fall in the range of 0 to 255. Scale these values to a range of 0 to 1 before feeding them to the neural network model. To do so, divide the values by 255. It is important that the training set and the testing set be preprocessed in the same way.

Creating and Compiling the Model

Building the neural network requires configuring the layers of the model, then compiling the model.

The basic building block of a neural network is the layer. Layers extract representations from the data fed into them. Hopefully, these representations are meaningful for the problem at hand.

Most of deep learning consists of chaining together simple layers. Most layers, such as keras.layers.Dense, have parameters that are learned during training.

```
model = keras.Sequential([
    keras.layers.Flatten(input_shape=(28, 28)),
    keras.layers.Dense(128, activation="relu"),
    keras.layers.Dense(10, activation="softmax")
])
```

The first layer in this network, keras.layers.Flatten, transforms the format of the images from a two-dimensional array (of 28 by 28 pixels) to a one-dimensional array (of 28 * 28 = 784 pixels). Think of this layer as unstacking rows of pixels in the image and lining them up. This layer has no parameters to learn; it only reformats the data.

After the pixels are flattened, the network consists of a sequence of two keras.layers.Dense layers. These are densely connected, or fully connected, neural layers. The first Dense layer has 128 nodes (or neurons). The second (and last) layer returns a logits array with length of 10. Each node contains a score that indicates the current image belongs to one of the 10 classes.

Before the model is ready for training, it needs a few more settings. These are added during the model's compile step:

Loss function - Measures how accurate the model is during training. You want to minimize this function to "steer" the model in the right direction.

Optimizer - This is how the model is updated based on the data it sees and its loss function.

Metrics - Used to monitor the training and testing steps. The following example uses *accuracy*, the fraction of the images that are correctly classified.

```
model.compile(optimizer="adam", loss="sparse_categorical_crossentropy", metrics=["accuracy"])
```

Training the Model

Training the neural network model requires the following steps:

- Feed the training data to the model. In this example, the training data is in the train_images and train_labels arrays.
- The model learns to associate images and labels.
- You ask the model to make predictions about a test set—in this example, the test_images array.
- Verify that the predictions match the labels from the test_labels array.

Feed the model

To start training, call the model.fit method—so called because it "fits" the model to the training data:

```
model.fit(train_images, train_labels, epochs=5)
```

Evaluate accuracy

Next, compare how the model performs on the test dataset:

```
test_loss, test_acc = model.evaluate(test_images, test_labels)
```

Putting the Created Model to Use

There is no point in creating an algorithm and not putting it to use.

```
prediction = model.predict(test_images)

for i in range (5):
   plt.grid(False),
   plt.imshow(test_images[i], cmap=plt.cm.binary)
   plt.xlabel("Actual: " + class_names[test_labels[i]])
   plt.title("Prediction: " + class_names[np.argmax(prediction[i])])
   plt.show()
```