National University of Computer and Emerging Sciences



Lab Manual 09

CL461-Artificial Intelligence Lab

|  |  |
| --- | --- |
| Course Instructor | Dr. Mubasher Baig |
| Lab Instructor (s) | Saad Ali  Mahmood Hussain |
| Section | A |
| Semester | Spring 2021 |

Department of Computer Science

FAST-NU, Lahore, Pakistan

Table of Contents

[**Objectives**](#_heading=h.30j0zll) **2**

[**Task Distribution**](#_heading=h.1fob9te) **2**

[**Import Fashion MNIST Dataset**](#_heading=h.rbmnkm7h0pyn) **3**

[**4. Explore the data**](#_heading=h.45izeg91tp45) **5**

[**5. Preprocess the data**](#_heading=h.oiscpythhlkn) **6**

[**6. Building the Model**](#_heading=h.mhqk8gmh6ex) **8**

[6.1 Set up the layers](#_heading=h.ishsg0frs9kx) 8

[6.2 Compile the model](#_heading=h.xjl96ufr4j26) 9

[**7. Train the model**](#_heading=h.7i6imws6cqvc) **9**

[7.1 Feed the model](#_heading=h.y2n3khdbwggf) 9

[7.2 Evaluate accuracy](#_heading=h.yslx2zw1xtph) 10

[7.3 Make predictions](#_heading=h.b2p47df26mwo) 10

[7.4 Verify predictions](#_heading=h.tmbrtkvqlvov) 12

[**8. Use Trained Model**](#_heading=h.4bco1jaqfpra) **12**

[**9. Exercise**](#_heading=h.oa81hik691v7) **14**

[**10 . Submission Instructions:**](#_heading=h.63v6a3kk45h0) **14**

# Objectives

This lab aims to build a multiclass image classifier using a feed-forward neural network. The basics of building a simple neural network will be discussed.

# Task Distribution

|  |  |
| --- | --- |
| **Total Time** | **170 Minutes** |
| Data Exploration | 20 Minutes |
| Make ANN | 20 Minutes |
| Make Predictions | 30 Minutes |
| Exercise | 90 Minutes |
| Online Submission | 10 Minutes |
|  |  |

# Import Fashion MNIST Dataset

This guide uses *tf.keras*, a high-level API to build and train models in TensorFlow. First of all import the libraries. Keras comes with Tensorflow, it is pre-built into Tensorflow and is not required to import separately as well.

# TensorFlow and tf.keras

import tensorflow as tf

# Helper libraries

import numpy as np

import matplotlib.pyplot as plt

print(tf.\_\_version\_\_)

We are going to use the *Fashion MNIST* dataset which contains 70,000 grayscale images in 10 categories. The images show individual articles of clothing at low resolution (28 by 28 pixels), as:



Fashion MNIST is intended as a drop-in replacement for the classic MNIST dataset—often used as the "Hello, World" of machine learning programs for computer vision. The MNIST dataset contains images of handwritten digits (0, 1, 2, etc.) in a format identical to that of the articles of clothing you'll use here.

This lab uses Fashion MNIST for variety, and because it's a slightly more challenging problem than regular MNIST. Both datasets are relatively small and are used to verify that an algorithm works as expected. They're good starting points to test and debug code.

Here, 60,000 images are used to train the network and 10,000 images to evaluate how accurately the network learned to classify images. One can access the Fashion MNIST directly from TensorFlow. Import and load the Fashion MNIST data directly from TensorFlow:

fashion\_mnist = tf.keras.datasets.fashion\_mnist

(train\_images, train\_labels), (test\_images, test\_labels) = fashion\_mnist.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, we are storing them in list to use later when plotting the images:

class\_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',

'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']

# 4. Explore the data

Let's explore the format of the dataset before training the model. The following shows there are 60,000 images in the training set, with each image represented as 28 x 28 pixels:

train\_images.shape

(60000, 28, 28)

Likewise, there are 60,000 labels in the training set:

len(train\_labels)

60000

Each label is an integer between 0 and 9:

train\_labels

array([9, 0, 0, ..., 3, 0, 5], dtype=uint8)

There are 10,000 images in the test set. Again, each image is represented as 28 x 28 pixels:

test\_images.shape

(10000, 28, 28)

And the test set contains 10,000 images labels:

len(test\_labels)

10000

# 5. Preprocess the data

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:

plt.figure()

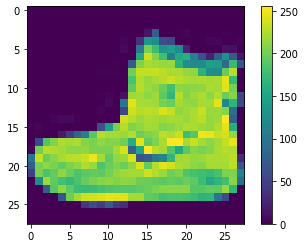
plt.imshow(train\_images[0])

plt.colorbar()

plt.grid(False)

plt.show()

The resultant image is:



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's important that the *training set* and the *testing set* be preprocessed in the same way:

train\_images = train\_images / 255.0

test\_images = test\_images / 255.0

To verify that the data is in the correct format and that you're ready to build and train the network, let's display the first 25 images from the *training set* and display the class name below each image.

plt.figure(figsize=(10,10))

for i in range(25):

plt.subplot(5,5,i+1)

plt.xticks([])

plt.yticks([])

plt.grid(False)

plt.imshow(train\_images[i], cmap=plt.cm.binary)

plt.xlabel(class\_names[train\_labels[i]])

plt.show()



# 6. Building the Model

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

## 6.1 Set up the layers

The basic building block of a neural network is the *layer*. Layers extract representations from the data fed into them.

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

model = tf.keras.Sequential([

tf.keras.layers.Flatten(input\_shape=(28, 28)),

tf.keras.layers.Dense(128, activation='relu'),

tf.keras.layers.Dense(10)

])

The first layer in this network, *tf.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 *tf.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.

## 6.2 Compile the model

Before the model is ready for training, it needs a few more settings. These are added during the model's [*compile*](https://www.tensorflow.org/api_docs/python/tf/keras/Model#compile) step:

* [*Loss function*](https://www.tensorflow.org/api_docs/python/tf/keras/losses) —This measures how accurate the model is during training. You want to minimize this function to "steer" the model in the right direction.
* [*Optimizer*](https://www.tensorflow.org/api_docs/python/tf/keras/optimizers) —This is how the model is updated based on the data it sees and its loss function.
* [*Metrics*](https://www.tensorflow.org/api_docs/python/tf/keras/metrics) —Used to monitor the training and testing steps. This example uses *accuracy*, the fraction of the images that are correctly classified.

model.compile(optimizer='adam',

loss=tf.keras.losses.SparseCategoricalCrossentropy(from\_logits=True),

metrics=['accuracy'])

# 7. Train the model

Training the neural network model requires the following steps:

1. Feed the training data to the model. In this, the training data is in the train\_images and train\_labels arrays.
2. The model learns to associate images and labels.
3. Then make predictions about a test set—in this example, the test\_images array.
4. Verify that the predictions match the labels from the test\_labels array.

### 

## 7.1 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=10)

## 7.2 Evaluate accuracy

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

test\_loss, test\_acc = model.evaluate(test\_images, test\_labels, verbose=2)

print('\nTest accuracy:', test\_acc)

Test accuracy: 0.8852999806404114

## 7.3 Make predictions

With the model trained, you can use it to make predictions about some images. The model's linear outputs, logits. Attach a softmax layer to convert the logits to probabilities, which are easier to interpret.

probability\_model = tf.keras.Sequential([model,

tf.keras.layers.Softmax()])

predictions = probability\_model.predict(test\_images)

Here, the model has predicted the label for each image in the testing set. Let's take a look at the first prediction:

predictions[0]

array([4.2506403e-07, 2.3205451e-09, 1.2338440e-08, 1.6134087e-09,

5.6793318e-08, 1.1142134e-02, 2.3751019e-07, 1.2638311e-02,

1.6467922e-07, 9.7621870e-01], dtype=float32)

A prediction is an array of 10 numbers. They represent the model's "confidence" that the image corresponds to each of the 10 different articles of clothing. You can see which label has the highest confidence value:

np.argmax(predictions[0])

9

So, the model is most confident that this image is an ankle boot, or class\_names[9]. Examining the test label shows that this classification is correct:

test\_labels[0]

9

Let's create functions to plot the image as well as an array of values.

*Plotting* the image comes first.

def plot\_image(i, predictions\_array, true\_label, img):

true\_label, img = true\_label[i], img[i]

plt.grid(False)

plt.xticks([])

plt.yticks([])

plt.imshow(img, cmap=plt.cm.binary)

predicted\_label = np.argmax(predictions\_array)

if predicted\_label == true\_label:

color = 'blue'

else:

color = 'red'

plt.xlabel("{} {:2.0f}% ({})".format(class\_names[predicted\_label],

100\*np.max(predictions\_array),

class\_names[true\_label]),

color=color)

Now comes to plot the array values.

def plot\_value\_array(i, predictions\_array, true\_label):

true\_label = true\_label[i]

plt.grid(False)

plt.xticks(range(10))

plt.yticks([])

thisplot = plt.bar(range(10), predictions\_array, color="#777777")

plt.ylim([0, 1])

predicted\_label = np.argmax(predictions\_array)

thisplot[predicted\_label].set\_color('red')

thisplot[true\_label].set\_color('blue')

## 7.4 Verify predictions

With the model trained, you can use it to make predictions about some images.

Let's look at the 0th image, predictions, and prediction array. Correct prediction labels are blue and incorrect prediction labels are red. The number gives the percentage (out of 100) for the predicted label.

i = 0

plt.figure(figsize=(6,3))

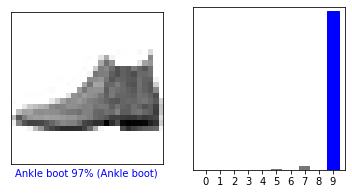
plt.subplot(1,2,1)

plot\_image(i, predictions[i], test\_labels, test\_images)

plt.subplot(1,2,2)

plot\_value\_array(i, predictions[i], test\_labels)

plt.show()



# 8. Use Trained Model

Finally, use the trained model to make a prediction about a single image.

# Grab an image from the test dataset.

img = test\_images[1]

print(img.shape)

28x28

tf.keras models are optimized to make predictions on a *batch*, or collection, of examples at once. Accordingly, even though you're using a single image, you need to add it to a list:

# Add the image to a batch where it's the only member.

img = (np.expand\_dims(img,0))

print(img.shape)

(1x28x28)

Now predict the correct label for this image:

predictions\_single = probability\_model.predict(img)

print(predictions\_single)

[[4.5591700e-05 5.8131788e-10 9.9679011e-01 7.1024013e-11 3.0492423e-03

4.2878790e-15 1.1495490e-04 8.2176217e-25 1.0162458e-09 5.4687570e-12]]

Now plot the value

plot\_value\_array(1, predictions\_single[0], test\_labels)

\_ = plt.xticks(range(10), class\_names, rotation=45)

# 

Now the value predicted by model is:

np.argmax(predictions\_single[0])

2

# 9. Exercise

1- The most commonly used activation functions in classifications are basically three

a- Rectified Linear Unit (**Relu**)

b- Logistic (**Sigmoid**)

c- Hyperbolic Tangent (**Tanh**)

In the lab example we used the first one and accuracy comes above 88%, Use other ones like logistic and hyperbolic to see which one of these three returns maximum accuracy.

Note: These activation functions are to be added in a dense layer. (10)

2- ANN networks called Deep learning when it contains more than two hidden layers. Add another hidden layer to our network contains the best activation function, one chosen from Q#1 with the number of neurons first 128 then 100 and at last 75. Summarize your answer with accuracy from each number. (10)

3- Import the MNIST hand-written digits dataset and apply the ANN on it. Conclude your accuracy with the best number of layers, neurons and activation functions. (10)

# 10 . Submission Instructions:

1. For Examples and Exercises given in manual, no dataset is required because the dataset used is available in Tensorflow library.
2. Use Google Colab for coding because you don’t need to install Tensorflow there
3. To make the submission, Create a Jupyter Notebook File (lab10\_rollno.ipynb), create a .zip file.