EXPERIMENT #5

Image Classification with Neural Networks on Jetson Nano

Instructor: Dr. Jafar Saniie
ECE 498-07

Acknowledgment: I acknowledge all of the work (including figures and codes) belongs to me and/or persons who are referenced.

Signature: Maggie Barclay & Theo Guidroz

Introduction

A. Purpose

The purpose of this lab is to understand the concept of image classification. The following terms will be introduced:

• Content and construction of conventional image datasets
• Neural networks and their structures
• Neural network training
• Evaluation and inference of a trained neural network

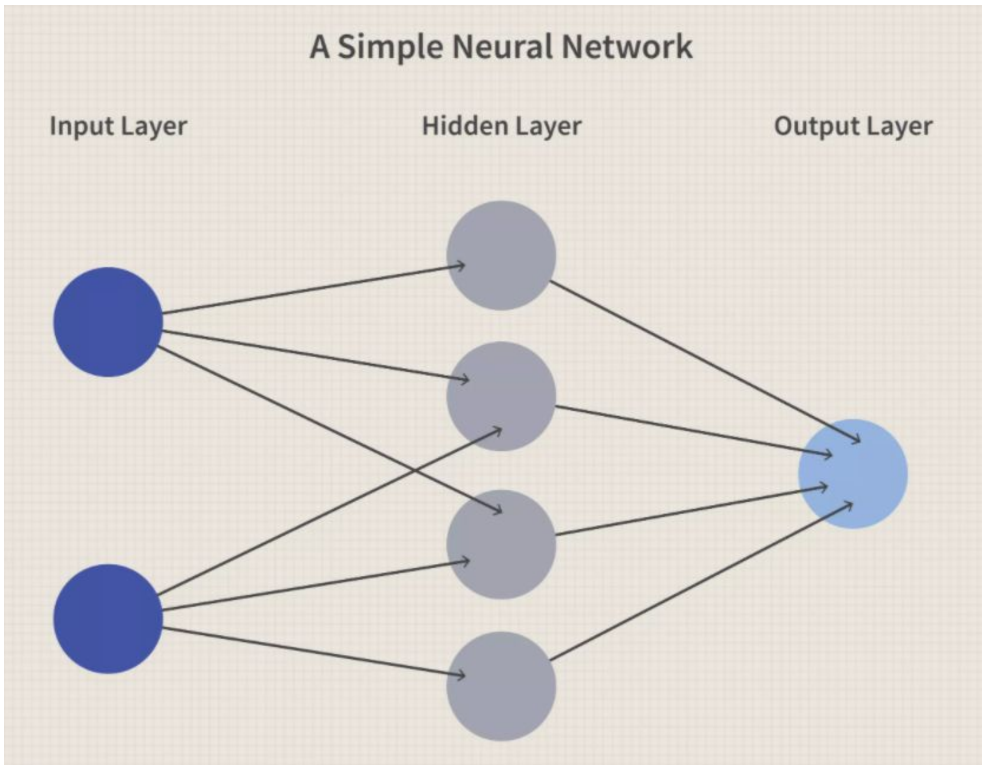
B. Background

A. Dataset

The creation of a dataset is the most time consuming and labor-intensive part of deep learning development workflow. Thousands of images need to be picked and labelled to train the weights. To simplify the process of creating a dataset, various methods can be used, including procedural generation of images from a simulated environment, using specific API (application programming interface) to fetch contents.

B. Basics of Neural Networks and their Training

A neural network is a series of algorithms that endeavors to recognize underlying relationships in a set of data through a process that mimics the way the human brain operates.



II. Lab Procedure and Equipment List

A. Equipment

Equipment

• 1 x Jetson Nano single-board computer
• Internet connectivity
• I/O devices (USB keyboard/mouse, etc.)
• External monitor with HDMI/DP port

B. Procedure

Part A

1. Installing TensorFlow on Jetson Nano
2. Running the Fashion MNIST datase

Preliminary Assignment

1. Look for information of the CIFAR-100 dataset, and describe its structure and labeling.

In the cifar-100 dataset, 5000 labelled examples are provided, 10000 examples are unlabeled. (source: https://www.kaggle.com/c/ml2016-7-cifar-100) There are 100 classes that small color photos in the set are apart of.

2. Why creating an image dataset is generally labor intensive and time consuming?

Every element has to be framed and label and this can take a lot of time since thousands of images have to be labeled to train data.

3. Review part B of the Background section, and explain how a neural network is trained to distinguish cats and dogs.

A cost function is created that outputs a value that is determined by the difference between the prediction and the ground truth. A back propagation is used which goes through the neural network in reverse to check which neuron causes the largest increase in cost function. A cat will be distinguished from a dog because of the cost functions' ability to discern prediction errors and train itself by tweaking weights values. These unique values allow the network to distinguish between cats and dogs.

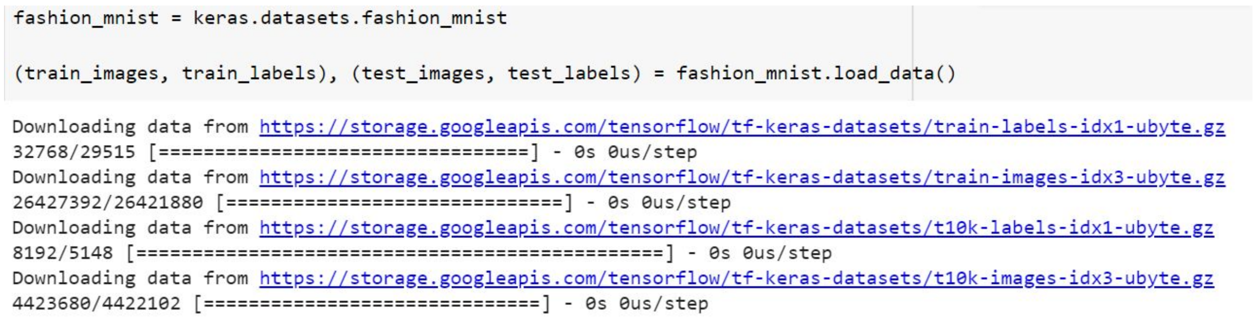
4. It seems that an overfitted neural network can still be quite accurate if we are only using training data for inference. Is this advantageous? Explain.

Overfitting a neural network happens when a model tries to predict a trend in data that's too noisy. If a function is too closely fit to a set of data points, this may be an overfitting issue. However, it is advantageous since this network may still be able to be used for inference. If the model is still able to make predictions that are accurate despite being an overfitted network, this is advantageous.

A. Installing TensorFlow on Jetson Nano

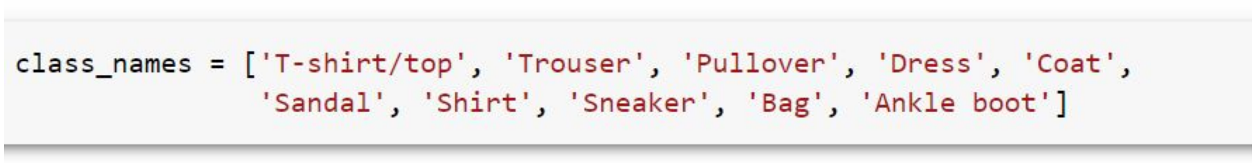
B. Clothing Classification using Neural Networks

1. Import the Fashion MNIST dataset



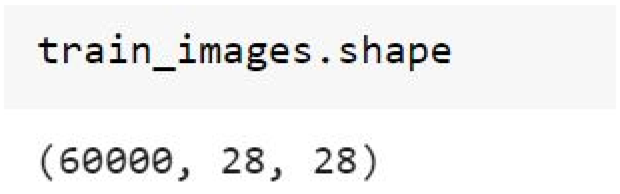
```
fashion_mnist = keras.datasets.fashion_mnist

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

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-labels-idx1-ubyte.gz
32768/29515 [=================================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-images-idx3-ubyte.gz
26427392/26421880 [==============================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-labels-idx1-ubyte.gz
8192/5148 [===============================================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-images-idx3-ubyte.gz
4423680/4422102 [==============================] - 0s 0us/step
```



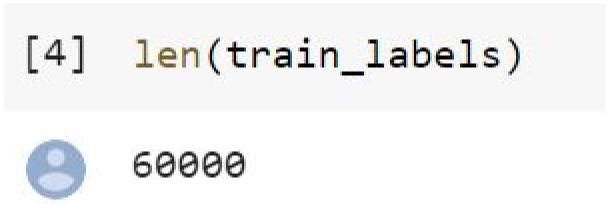
```
class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',
               'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']
```

2. Check the structure of the dataset



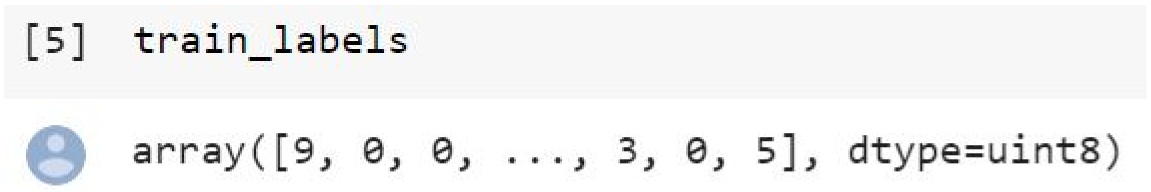
```
train_images.shape
```

```
(60000, 28, 28)
```



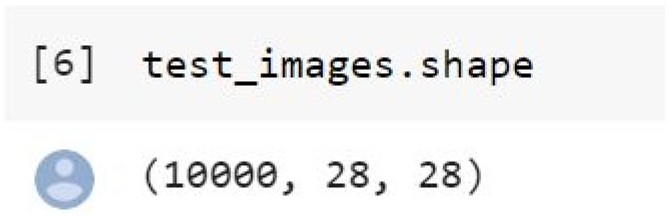
```
[4] len(train_labels)
```

```
60000
```



```
[5] train_labels
```

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



```
[6] test_images.shape
```

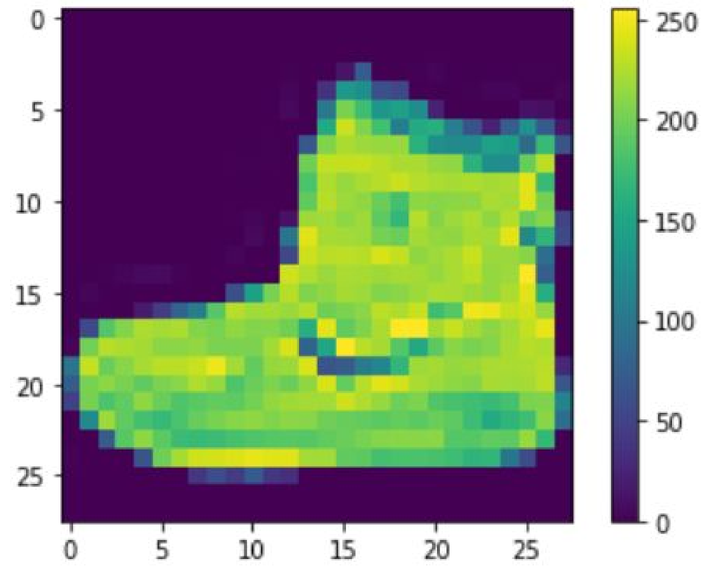
```
(10000, 28, 28)
```

```
[9]  len(test_labels)
```

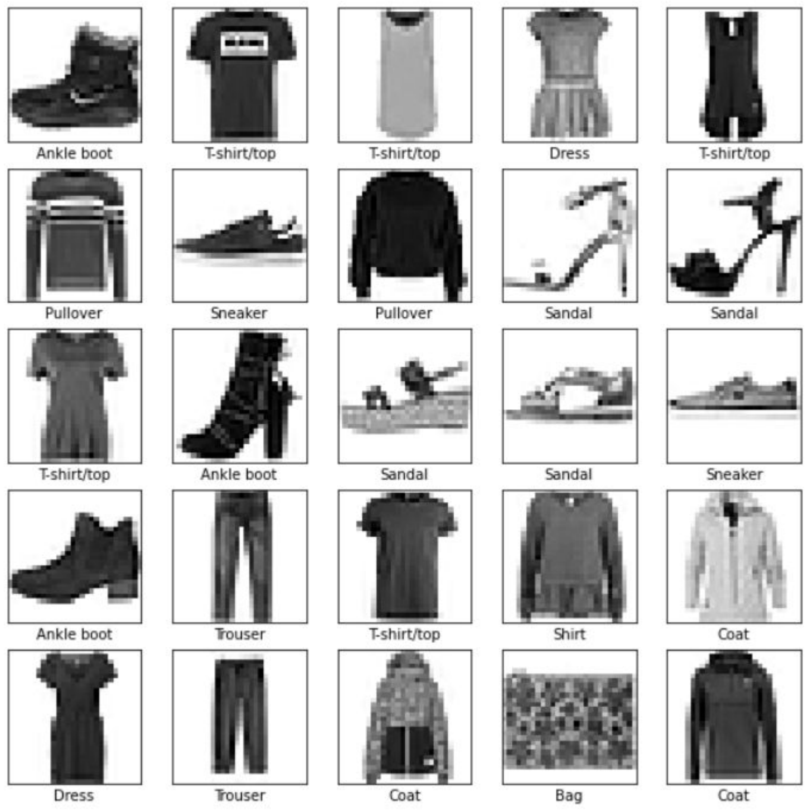
```
10000
```

3. Preprocess the data

```
plt.figure()
plt.imshow(train_images[0])
plt.colorbar()
plt.grid(False)
plt.show()
```

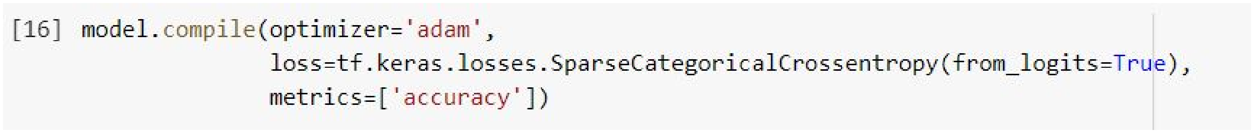


```
plt.figure(figsize=(10,10))
for i in range(25):
    plt.subplot(5,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(train_images[i], cmap=plt.cm.binary)
    plt.xlabel(class_names[train_labels[i]])
plt.show()
```



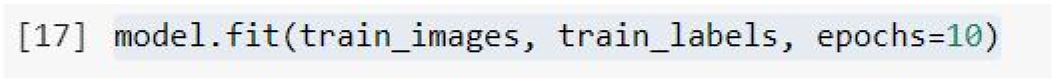
4. Build the model

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



```
[16] model.compile(optimizer='adam',
                   loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
                   metrics=['accuracy'])
```

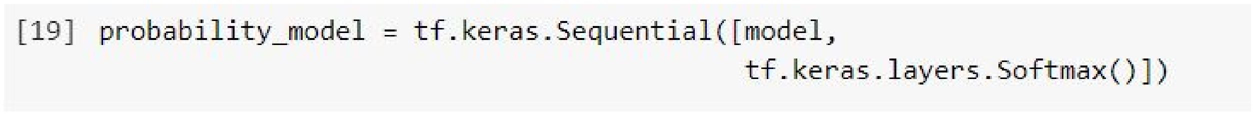
5. Train the model



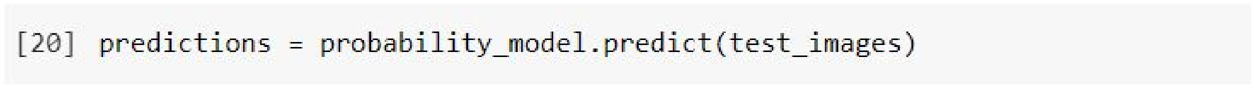
```
[17] model.fit(train_images, train_labels, epochs=10)
```

*

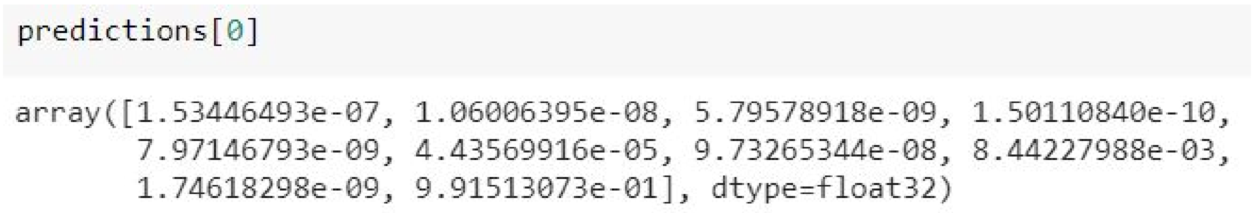
6. Make and verify predictions:



```
[19] probability_model = tf.keras.Sequential([model,
                                              tf.keras.layers.Softmax()])
```

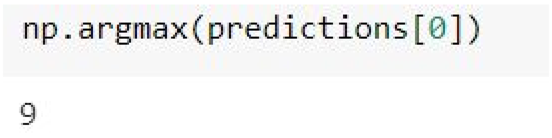


```
[20] predictions = probability_model.predict(test_images)
```



```
predictions[0]
```

```
array([1.53446493e-07, 1.06006395e-08, 5.79578918e-09, 1.50110840e-10,
       7.97146793e-09, 4.43569916e-05, 9.73265344e-08, 8.44227988e-03,
       1.74618298e-09, 9.91513073e-01], dtype=float32)
```

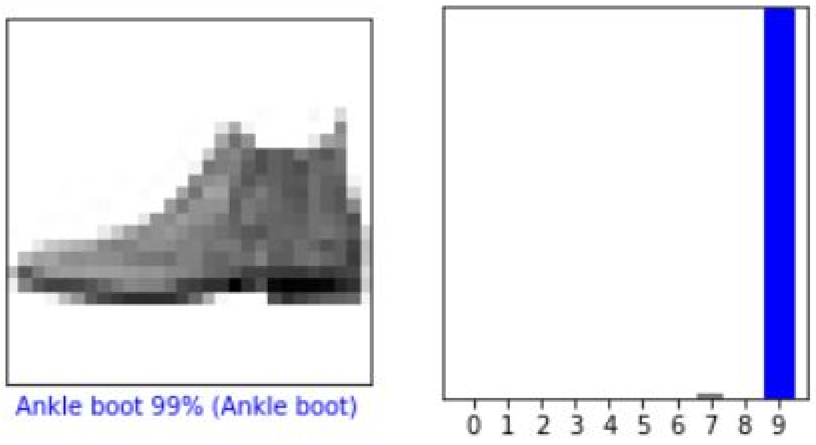


```
np.argmax(predictions[0])
```

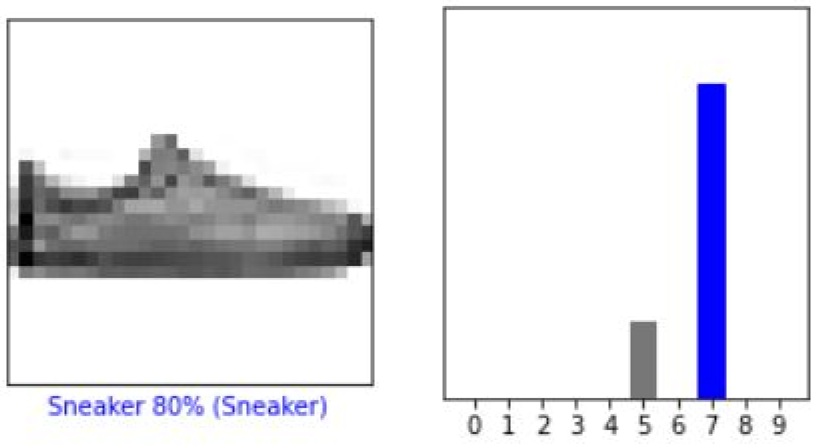
```
9
```

```python
def plot_image(i, predictions_array, true_label, img):
  predictions_array, true_label, img = predictions_array, 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)

def plot_value_array(i, predictions_array, true_label):
  predictions_array, true_label = predictions_array, 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')
```

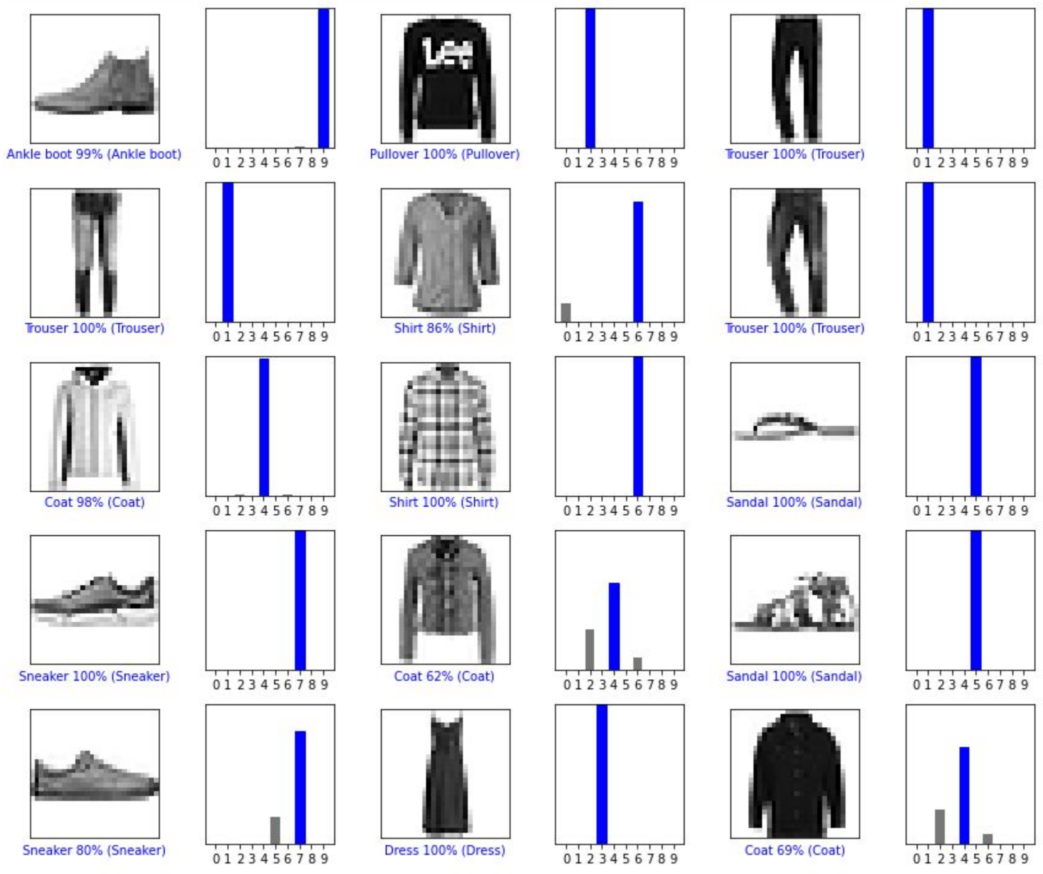
```
i = 0
plt.figure(figsize=(6,3))
plt.subplot(1,2,1)
plot_image(i, predictions[i], test_labels, test_images)
plt.subplot(1,2,2)
plot_value_array(i, predictions[i],  test_labels)
plt.show()
```



```
i = 12
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()
```



```
num_rows = 5
num_cols = 3
num_images = num_rows*num_cols
plt.figure(figsize=(2*2*num_cols, 2*num_rows))
for i in range(num_images):
  plt.subplot(num_rows, 2*num_cols, 2*i+1)
  plot_image(i, predictions[i], test_labels, test_images)
  plt.subplot(num_rows, 2*num_cols, 2*i+2)
  plot_value_array(i, predictions[i], test_labels)
plt.tight_layout()
plt.show()
```



7. Inference of the trained model

```
img = test_images[1]

print(img.shape)
```

```
(28, 28)
```

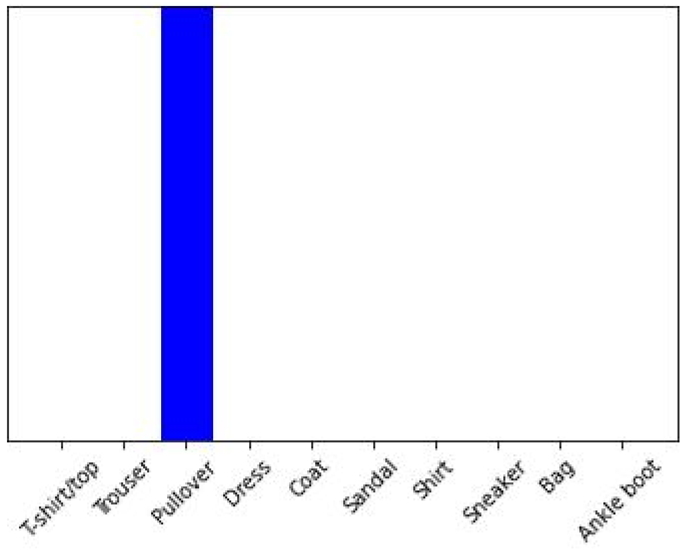
```
img = (np.expand_dims(img,0))

print(img.shape)
```

```
(1, 28, 28)
```

```
predictions_single = probability_model.predict(img)

print(predictions_single)
```

```
[[1.0163322e-05 8.3077240e-11 9.9989307e-01 7.5015244e-11 3.3217832e-05
  1.3539201e-17 6.3528583e-05 2.3964349e-21 4.5346841e-14 1.0482872e-14]]
```

```
plot_value_array(1, predictions_single[0], test_labels)
_ = plt.xticks(range(10), class_names, rotation=45)
```



```
np.argmax(predictions_single[0])
```

```
2
```

<u>Discussion</u>

1. Why do you need to make all images in the dataset to be of the same resolution before feeding them to the neural network? (10 pts)

Images are composed of matrices of pixel values and therefore need to be of the same matrix orders to optimize training. The only way to ensure that is by having them at the same resolution.

2. What is the cause of performance downgrade between testing and training results? (10pts)

The downgrade could be linked to overfitting, modelling noise or just memorization.

3. Check the MNIST database of handwritten digits http://yann.lecun.com/exdb/mnist/, and use what you learned in this lab to perform number recognition using neural networks with the dataset. (30 pts)

See code attached.

IV. Conclusions

In this lab, students built a deeper understanding of neural networks and image classification. Students were able to train their own neural network using MNIST databases.

References

[1] Lab 5 manual

Code:

```
Import pytorch
from torchvision import transforms
import torchvision.datasets as datasets

import matplotlib.pyplot as plt

from model import Model

import numpy as np




mnist_trainset = datasets.MNIST(root='./data', train=True, download=True,
transform=transforms.Compose([transforms.ToTensor()]))
mnist_testset = datasets.MNIST(root='./data', train=False, download=True,
transform=transforms.Compose([transforms.ToTensor()]))


mnist_valset, mnist_testset = torch.utils.data.random_split(mnist_testset,
[int(0.9 * len(mnist_testset)), int(0.1 * len(mnist_testset))])


train_dataloader = torch.utils.data.DataLoader(mnist_trainset,
batch_size=64, shuffle=True)
val_dataloader = torch.utils.data.DataLoader(mnist_valset, batch_size=32,
shuffle=False)
test_dataloader = torch.utils.data.DataLoader(mnist_testset, batch_size=32,
shuffle=False)




# visualize data

fig=plt.figure(figsize=(20, 10))
```

```python
for i in range(1, 6):

   img = transforms.ToPILImage(mode='L')(mnist_trainset[i][0])

   fig.add_subplot(1, 6, i)

   plt.title(mnist_trainset[i][1])

   plt.imshow(img)

plt.show()




model = Model()

criterion = torch.nn.CrossEntropyLoss()

optimizer = torch.optim.Adam(model.parameters(), lr=0.001)






no_epochs = 100

train_loss = list()

val_loss = list()

best_val_loss = 1

for epoch in range(no_epochs):

   total_train_loss = 0

   total_val_loss = 0


   model.train()

   # training

   for itr, (image, label) in enumerate(train_dataloader):
       optimizer.zero_grad()

       pred = model(image)

       loss = criterion(pred, label)
       total_train_loss += loss.item()

       loss.backward()
       optimizer.step()

   total_train_loss = total_train_loss / (itr + 1)
   train_loss.append(total_train_loss)

   # validation
   model.eval()
   total = 0
   for itr, (image, label) in enumerate(val_dataloader):




       pred = model(image)

       loss = criterion(pred, label)
       total_val_loss += loss.item()
```

```
        pred = torch.nn.functional.softmax(pred, dim=1)

        for i, p in enumerate(pred):

            if label[i] == torch.max(p.data, 0)[1]:

                total = total + 1



    accuracy = total / len(mnist_valset)



    total_val_loss = total_val_loss / (itr + 1)

    val_loss.append(total_val_loss)



    print('\nEpoch: {}/{}, Train Loss: {:.8f}, Val Loss: {:.8f}, Val
Accuracy: {:.8f}'.format(epoch + 1, no_epochs, total_train_loss,
total_val_loss, accuracy))

    if total_val_loss < best_val_loss:

        best_val_loss = total_val_loss





fig=plt.figure(figsize=(20, 10))

plt.plot(np.arange(1, no_epochs+1), train_loss, label="Train loss")

plt.plot(np.arange(1, no_epochs+1), val_loss, label="Validation loss")

plt.xlabel('Loss')

plt.ylabel('Epochs')

plt.title("Loss Plots")

plt.legend(loc='upper right')

plt.show()
```

```
# test model
model.load_state_dict(torch.load("model.dth"))
model.eval()

results = list()
total = 0
for itr, (image, label) in enumerate(test_dataloader):

   pred = model(image)
   pred = torch.nn.functional.softmax(pred, dim=1)

   for i, p in enumerate(pred):
       if label[i] == torch.max(p.data, 0)[1]:
           total = total + 1
           results.append((image, torch.max(p.data, 0)[1]))

test_accuracy = total / (itr + 1)
print('Test accuracy {:.8f}'.format(test_accuracy))

# visualize results
fig=plt.figure(figsize=(20, 10))
for i in range(1, 11):
   img =
transforms.ToPILImage(mode='L')(results[i][0].squeeze(0).detach().cpu())
   fig.add_subplot(2, 5, i)
```

```
    plt.title(results[i][1].item())
    plt.imshow(img)
plt.show()
```