Vhaijaiyanthishree Venkataramanan
Instructor: Dr. Sedat Ozer
Course Title: CAP 5415 Computer Vision
Submission Date: November 5, 2018.

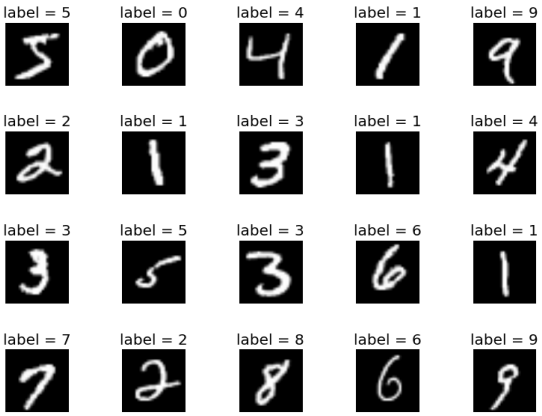

Fig 1: MNIST Dataset

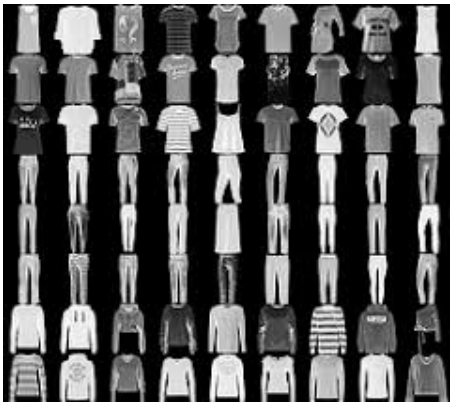

Fig 2: Fashion MNIST Dataset

Report - Programming Assignment 4

Note: All the computations were carried out on Google Colab

PART 1 & 2:

Your first task: run the template code and simply write down (report) the accuracy value you obtained on the test data after 12 epochs in your report file. (5 points). You do not need to submit a code for this step.

Solution: The template code on Keras for the MNIST dataset was run on Google Colab. The accuracy values obtained for each epoch can be seen from the below figures 3 and 4.

```
Using TensorFlow backend.
Downloading data from https://s3.amazonaws.com/img-
datasets/mnist.npz
11493376/11490434 [==============================] - 2s
0us/step
x_train shape: (60000, 28, 28, 1)
60000 train samples
10000 test samples
Train on 60000 samples, validate on 10000 samples
Epoch 1/12
60000/60000 [==============================] - 13s
218us/step - loss: 0.2631 - acc: 0.9187 - val_loss:
0.0563 - val_acc: 0.9817
Epoch 2/12
60000/60000 [==============================] - 9s
152us/step - loss: 0.0913 - acc: 0.9726 - val_loss:
0.0404 - val_acc: 0.9859
Epoch 3/12
60000/60000 [==============================] - 9s
152us/step - loss: 0.0684 - acc: 0.9795 - val_loss:
0.0409 - val_acc: 0.9858
Epoch 4/12
60000/60000 [==============================] - 9s
152us/step - loss: 0.0560 - acc: 0.9832 - val_loss:
0.0302 - val_acc: 0.9895
Epoch 5/12
60000/60000 [==============================] - 9s
151us/step - loss: 0.0473 - acc: 0.9858 - val_loss:
0.0314 - val_acc: 0.9893
Epoch 6/12
60000/60000 [==============================] - 9s
152us/step - loss: 0.0422 - acc: 0.9866 - val_loss:
0.0267 - val_acc: 0.9902
Epoch 7/12
60000/60000 [==============================] - 9s
152us/step - loss: 0.0382 - acc: 0.9879 - val_loss:
0.0287 - val_acc: 0.9902
```

Fig 3:Accuracy values for each epoch using MNIST Dataset

```
Epoch 8/12
60000/60000 [==============================] - 9s
151us/step - loss: 0.0342 - acc: 0.9891 - val_loss:
0.0253 - val_acc: 0.9911
Epoch 9/12
60000/60000 [==============================] - 9s
151us/step - loss: 0.0317 - acc: 0.9899 - val_loss:
0.0266 - val_acc: 0.9908
Epoch 10/12
60000/60000 [==============================] - 9s
152us/step - loss: 0.0295 - acc: 0.9914 - val_loss:
0.0277 - val_acc: 0.9919
Epoch 11/12
60000/60000 [==============================] - 9s
152us/step - loss: 0.0282 - acc: 0.9911 - val_loss:
0.0249 - val_acc: 0.9920
Epoch 12/12
60000/60000 [==============================] - 9s
151us/step - loss: 0.0256 - acc: 0.9919 - val_loss:
0.0232 - val_acc: 0.9926
Test loss: 0.023185137596364074
Test accuracy: 0.9926
```

Fig 4: Accuracy values for each epoch using MNIST Dataset

PART 3:

Your next task is understanding the used network model in the example. Write down the network model (the architecture) from the code in your report (5 points).

Here is an example for describing a simple model in the text form: Conv ===> Pooling ===> FC ===> softmax. Make sure you also list the number of filters, filter dims, number of units and activation functionsfor each layer (where applicable). As an alternative, you can draw a figure illustrating the network architecture as well, if you prefer drawing figures. You do not need to submit a code for this step.

```
model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3),
                 activation='relu',
                 input_shape=input_shape))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))
```

Fig 5: Code snippet describing the model

Solution: Figure 5 describes the code snippet corresponding to the architecture of the model. Below is a detailed description of the architecture step by step:

- **model = Sequential()**
  This part of the code describes the model type as Sequential.

- **model.add(Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=input_shape))**
  This part of the code describes that a 2D Convolutional layer is added to process the 2D MNIST input images. The Conv2D() layer function has been passed four arguments:
  - The number of output channels, 32.
  - The input kernel_size, a 3x3 moving window.
  - The activation function, Rectified Linear Unit (ReLU).
  - The size of the input layer, described at the beginning of the code as 28x28x1.

- **model.add(Conv2D(64, (3, 3), activation='relu'))**
  model.add(MaxPooling2D(pool_size=(2, 2)))
  - This layer is a convolutional + max pooling layer with 64 output channels.
  - The default strides argument in the Conv2D() function is (1, 1) in Keras.
  - The input tensor for this layer is (batch_size, 28, 28, 32) where 28x28 is the size of the image and 32 is the number of output channels from the previous layer.

- **model.add(Dropout(0.25))**
  This layer is a dropout layer which drops 25% of the units of the network before proceeding to the next layer. This is done in order to prevent overfitting.

- **model.add(Flatten())**
  After building the convolutional layers, the output from these layers are flattened using the above line of code in oder to enter the fully connected layers

- **model.add(Dense(128, activation='relu'))**
  This layer is a Fully Connected layer. The Dense() function is passed two arguments:
  - The number of nodes, 128
  - The activation function, Rectified Linear Unit(ReLU), which is used to activate each of these nodes.

- **model.add(Dropout(0.5))**

This layer is a dropout layer which drops 50% of the units of the network before proceeding to the next layer.

- **model.add(Dense(num_classes, activation='softmax'))**

This is the final layer or the output layer of the network which is yet again a Fully Connected layer. This layer uses a softmax classification and the number of nodes in this layer corresponds to the number of classes present in the MNIST dataset which is 10.

Input ===> Conv(32, 3 x 3 filters, activation=relu) ===> Conv(64, 3 x 3 filters, activation=relu, MaxPooling(2 x 2 filters) ===> Dropout (p=0.25) ===> Flatten (144 units) ===> FC (128 Dense layer, activation = relu) ===> Dropout (p=0.5) ===> FC (10, Dense layer) ===> softmax

PART 4:

Now, let's have a look at how changing epoch value affects the network's performance. Change the epoch (iteration) value to 30 in the code and plot both training and test accuracies vs. epoch. There are many ways to plot that in python. If you don't want to write your own python function for that task, consider using Tensorboard (hint: check keras.callbacks.TensorBoard()). Include your plot in your report. (2 points) save and submit your code as PA4_Part4.py (10 points)

Solution:

```
Test loss: 0.029049916175357113
Test accuracy: 0.9918
Number of Epochs: 30
```

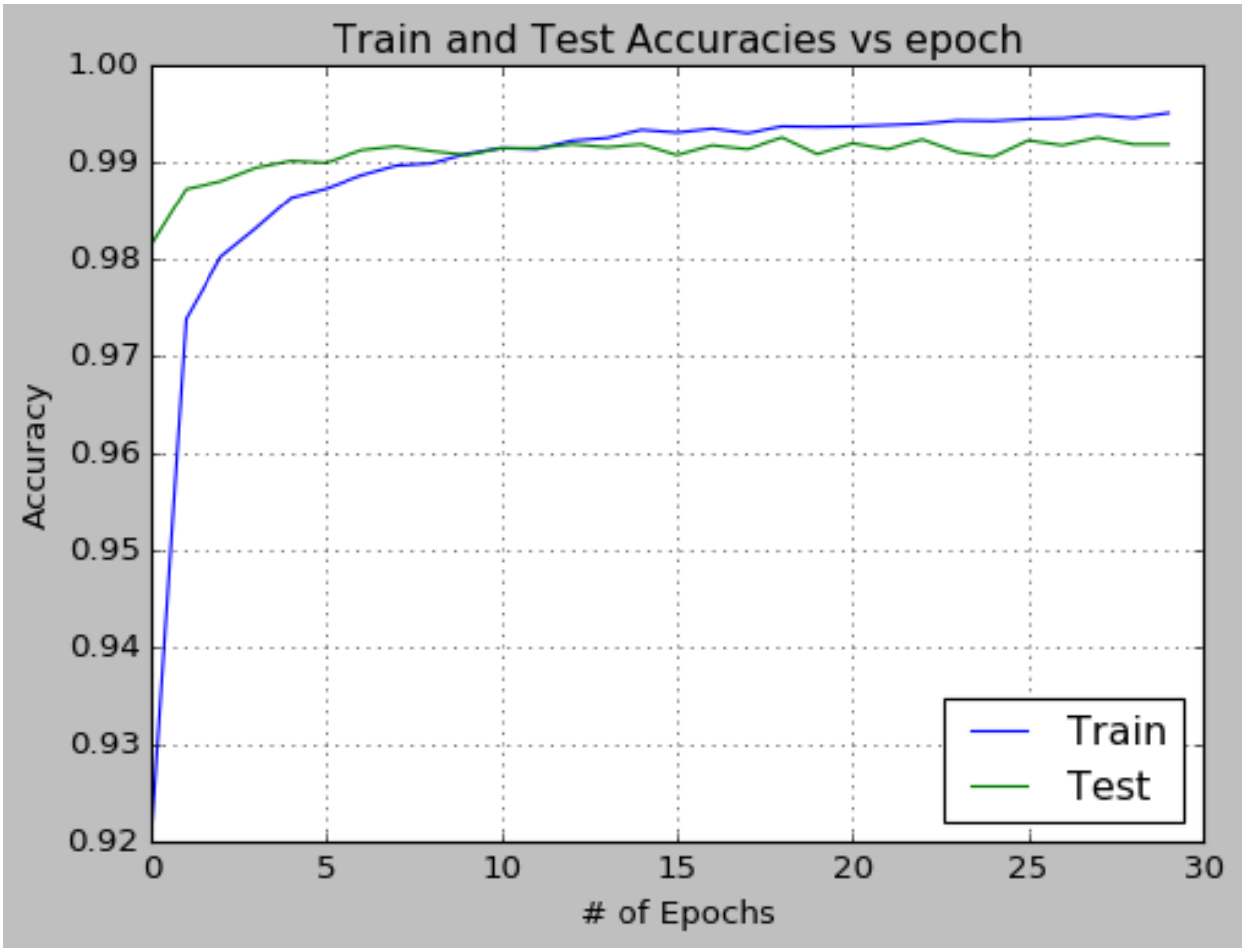

Fig 6: Train and Test Accuracies vs epoch on MNIST dataset using ConvNet Model

PART 5:

Plot both the training & test (validation) losses vs. epoch over 30 epochs(3 points). Note that this particular python code uses the test data as the validation data. a. save and submit your code as PA4_Part5.py (10 points)

Solution:

```
Test loss: 0.030702306761612407
Test accuracy: 0.9919
```

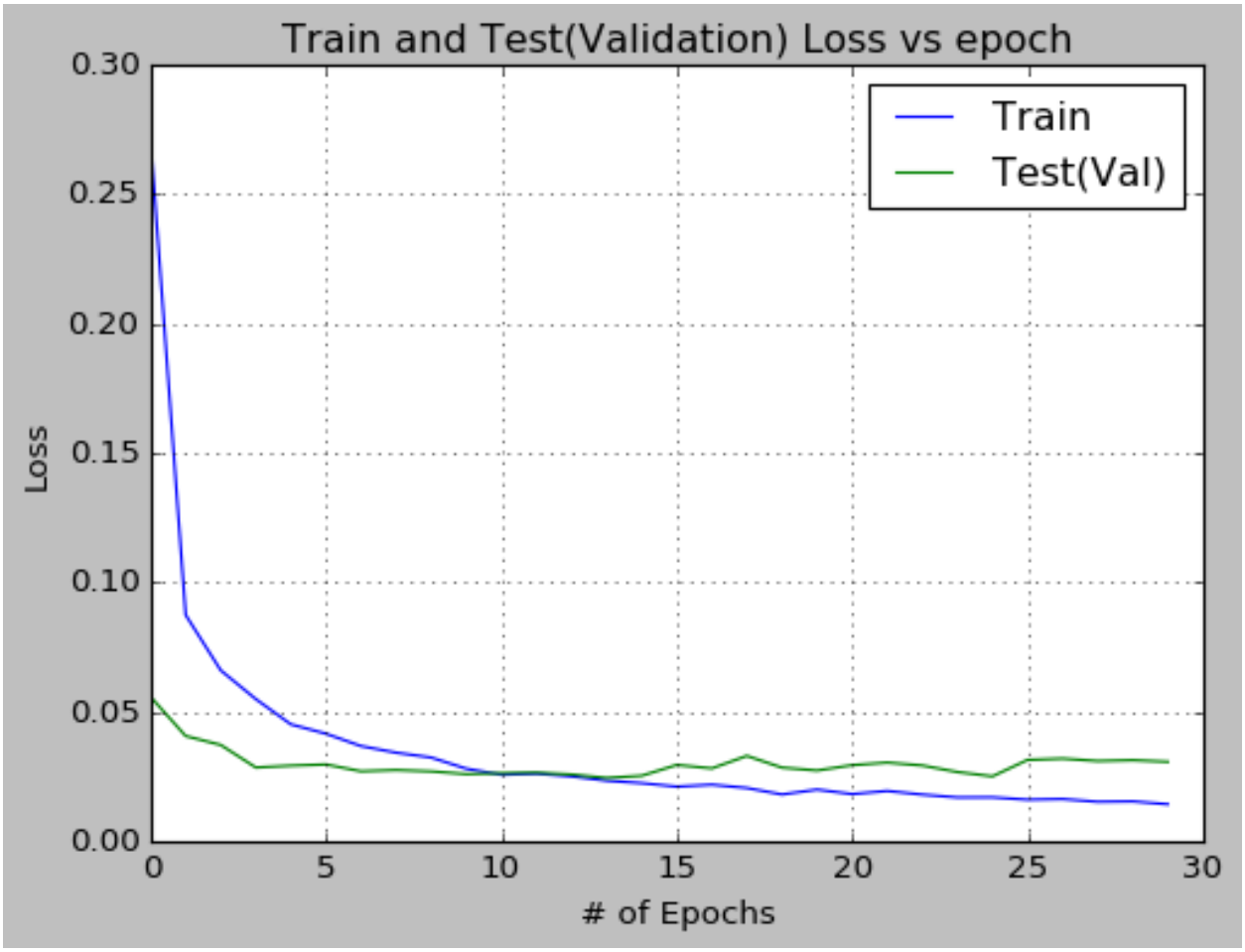

Fig 7: Train and Test(Validation) Loss vs epoch on MNIST dataset using ConvNet Model

PART 6:

Now, let's focus on changing (or editing) the existing network architecture in that existing code. Current template file uses a simple ConvNet model and we will replace that with the LeNet5 model that we studied in the class. Replace the simple ConvNet model with the LeNet5 model. Refer to the lecture slides for the full LeNet5 model and its details. Note that there are also many available Keras implementations available online for LeNet5. You can use any of them as a starting point as long as they match the model that we discussed and studied in the class. Train the new network (the LeNet5 model) on the MNIST dataset and plot the accuracy (both training and test) vs. epoch over 30 epochs. Show your plot in your report (3 points). Note: change only the model in the code. Do not change the other sections in the code: such as loss functions, optimization parameters, etc.

Solution:

```
Test loss: 0.021558979793304026
Test accuracy: 0.9926
```

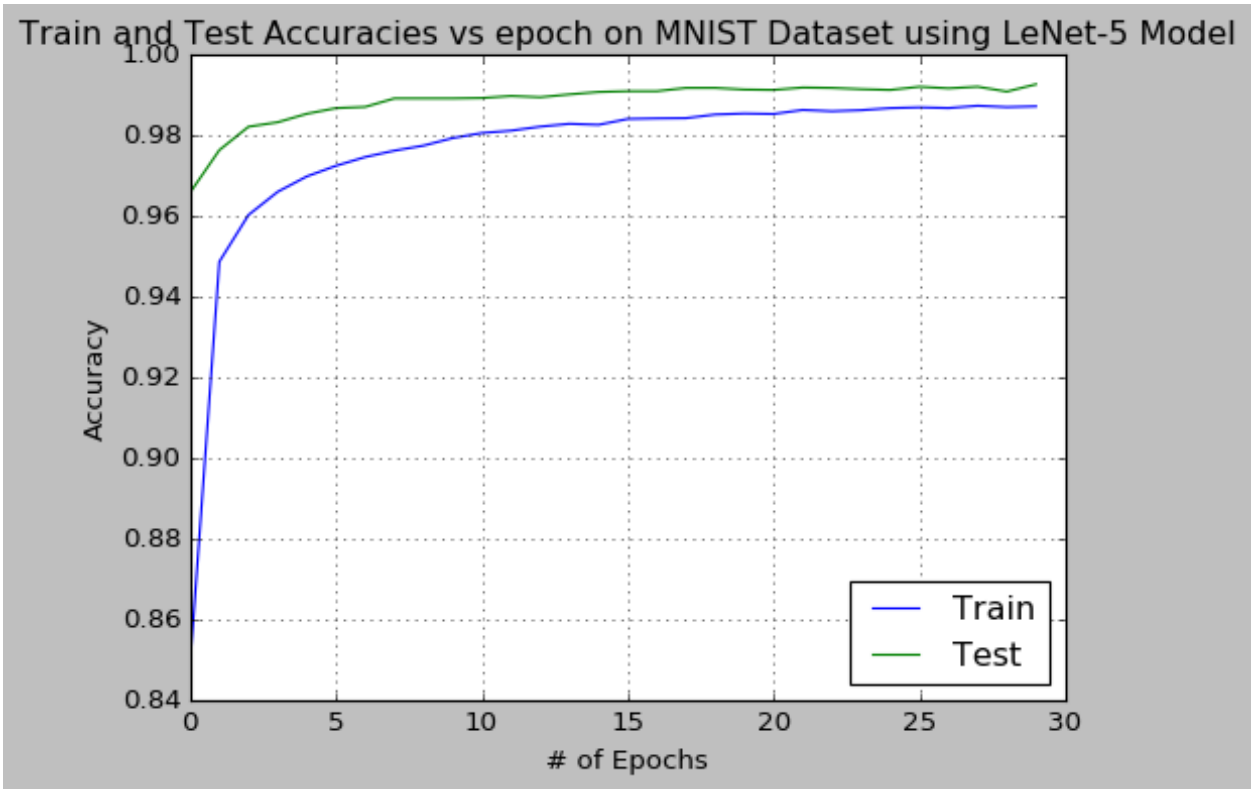

Fig 8: Train and Test Accuracies vs epoch on MNIST dataset using LeNet-5 Model

PART 7:

You now have a working LeNet5 model that you trained on the MNIST dataset and now it can recognize the digits. Now, let's train the LeNet5 model on another dataset. Train your LeNet5 architecture on the Fashion-MNIST dataset and plot the accuracy (both training and test) vs. epoch over 30 epochs. Now you have a LeNet5 model trained on fashion-MNIST dataset. Now that we have that already available, let's see if a simpler model would do any better. Train the simple ConvNet model (the original model in the template file) on the Fashion-MNIST as well and then compare the result of LeNet5 model to the basic ConvNet model you used above. Report your plots and the average training time per epoch for each model in your pdf file (3 points). LeNet5 is a deeper model compared to the simple ConvNet model. Did using a deeper net help in this case to obtain a higher accuracy? (2 points) Note: in this step, you focus on changing only the data related section from your previous code (PA4_Par6.py).

Solution:

```
Test loss: 0.3080132947266102
Test accuracy: 0.9062
```

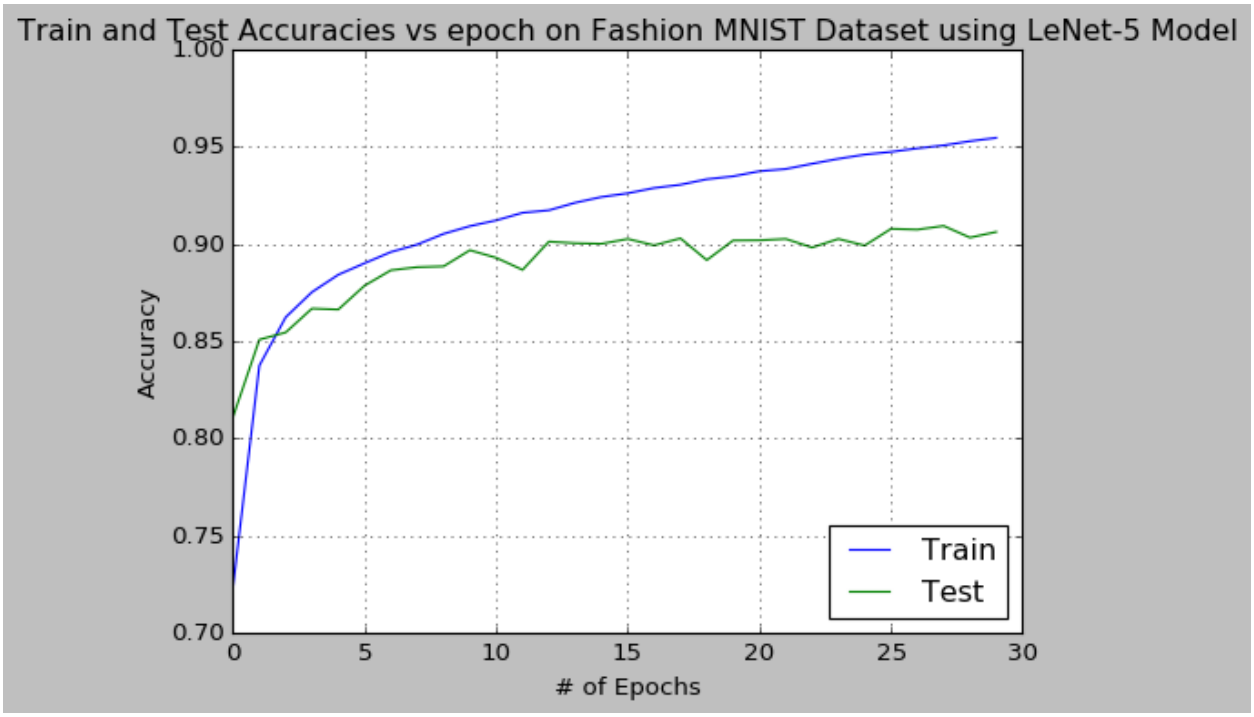

Fig 9: Train and Test Accuracies vs epoch on Fashion MNIST dataset using LeNet-5 Model

From the output logs, the test accuracy achieved for the LeNet-5 Model when trained on the Fashion MNIST dataset is 0.9062 or 90.62%. Also the average training time per epoch is approximately 6s.

Fig 10: Results of the LeNet-5 Model

```
Test loss: 0.22746761379912495
Test accuracy: 0.9306
```

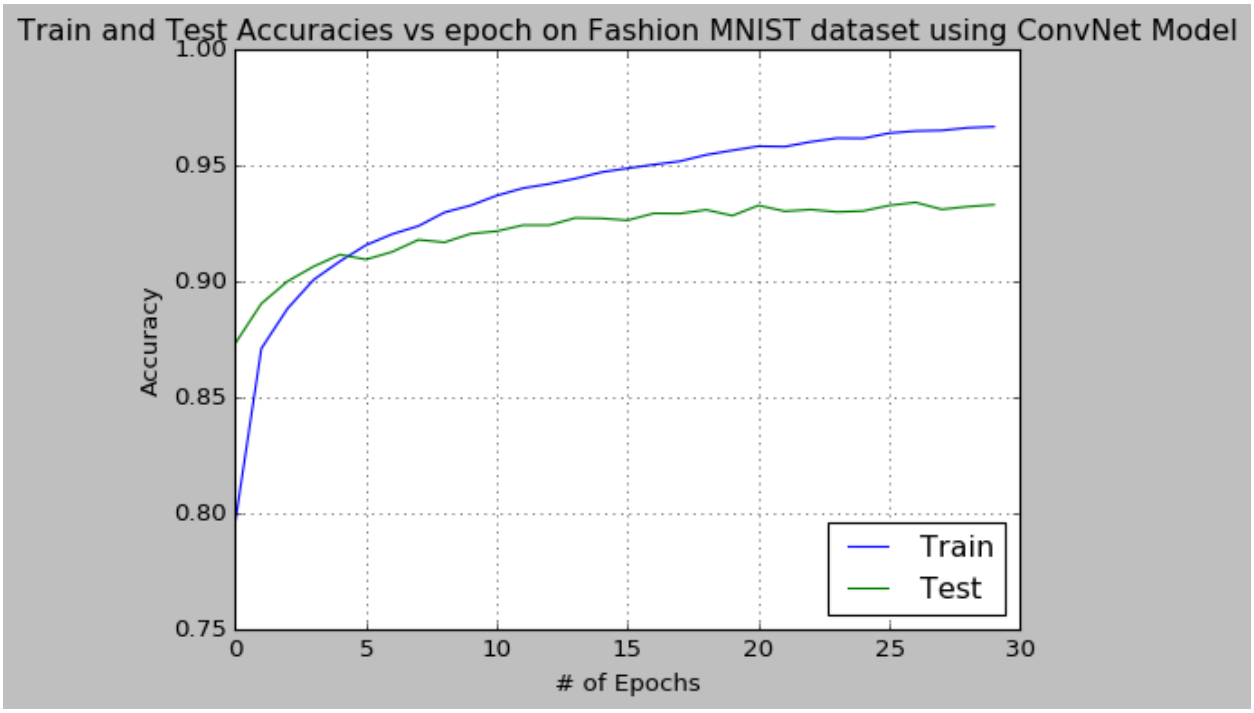

Fig 11: Train and Test Accuracies vs epoch on Fashion MNIST dataset using ConvNet Model

From the output logs, the test accuracy achieved for the ConvNet Model when trained on the Fashion MNIST dataset is 0.9306 or 93.06%. Also the average training time per epoch is approximately 10s.

Fig 12: Results of the ConvNet Model

RESULT ANALYSIS:

From Figures 10 and 12, we can understand that using a deeper net like LeNet-5 did not help increasing the accuracy of predictions in the case of Fashion MNIST Dataset. A reason to this could be the usage a minimal number of feature maps in the convolution layers.

PART 8:

The original particular template that you downloaded from GitHub uses "Adadelta" as the default optimization algorithm. Let's see how Stochastic Gradient Descent would do in this model instead of Adadelta. Change the optimizer to stochastic gradient descent (SGD) as shown below in your current LeNet5 implementation. Plot the accuracy vs. epoch over 30 epochs for both training and testing data. Here, you can use either MNIST or Fashion MNIST dataset. Also plot the loss (or cost J) vs. epoch over 30 epochs. You can use the template given below for setting the optimizer to SGD. (Optional: You are welcome to use additional parameters for SGD, if you prefer.) include your plot in your report file (2 points).

Solution:

```
Test loss: 0.6282634254455567
Test accuracy: 0.768
```

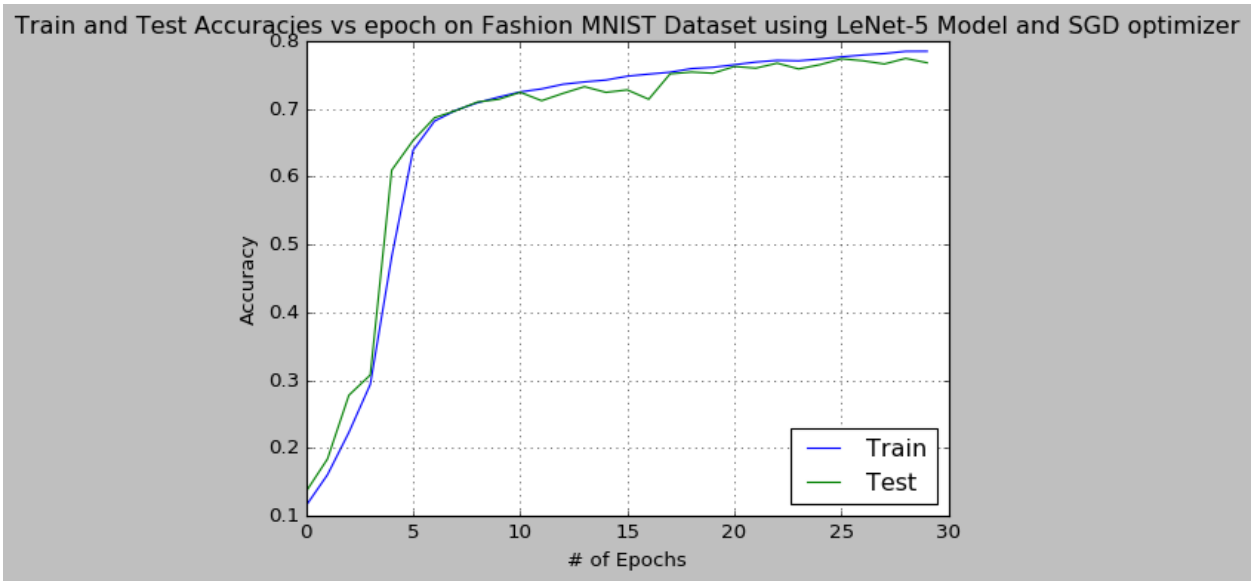

Fig 13: Train and Test Accuracies vs epoch on Fashion MNIST dataset using LeNet-5 Model and SGD Optimizer

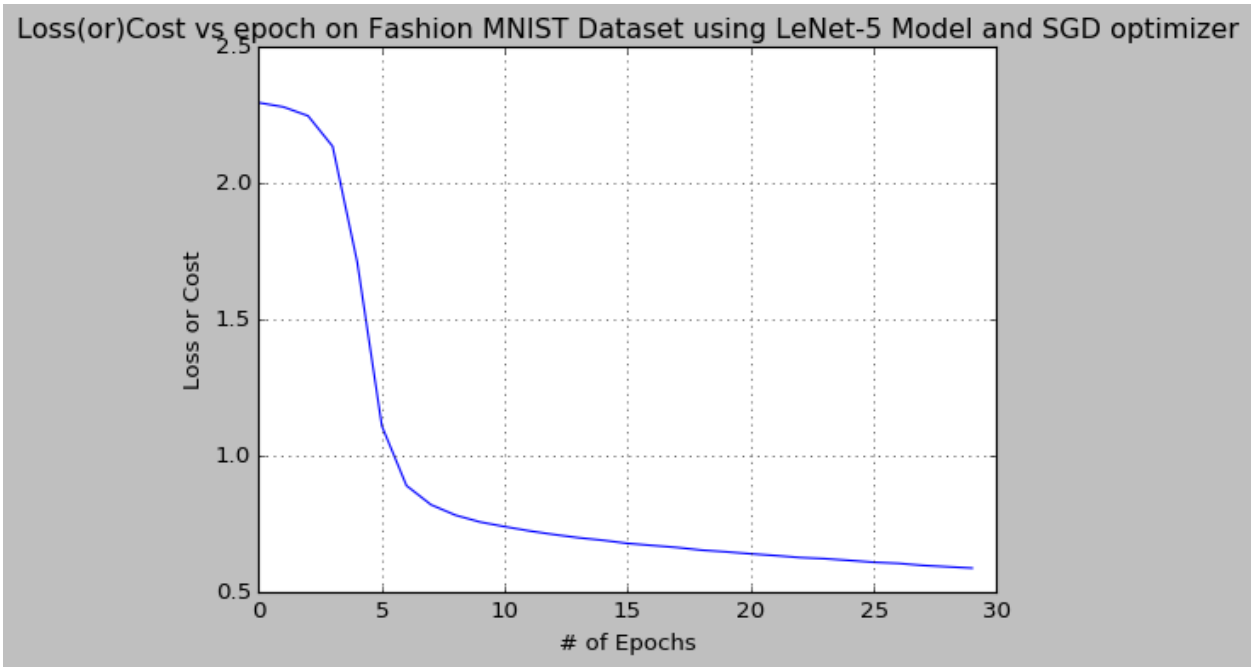

Fig 14: Loss(or)Cost vs epoch on Fashion MNIST dataset using LeNet-5 Model and SGD Optimizer

PART 9:

We know that learning rate is also an important hyper-parameter. Let's study that here. Use 5 random (but meaningful) learning rates and train your LeNet5 model at each of those 5 random learning rates separately on the MNIST dataset. Plot your accuracy vs. epoch results over 30 epochs for each of those learning rate values in a for-loop (over the 5 different learning rates). Which learning rate yields the best result? Include your chosen 5 learning rate values and their corresponding plots in your report (2 points) save and submit your code as PA4_Part9.py (10 points)

Solution:

The LeNet-5 model was trained using five different learning rates and AdaDelta optimizer on the MNIST dataset. The accuracies corresponding to each learning rate over 30 epochs are resulted in the following table 1 and the corresponding graphs are shown in Figures 15, 16, 17, 18 and 19.

Learning Rate	Accuracy
0.01	0.947
0.001	0.7411
0.0001	0.1395
0.002	0.8751
1.0	0.9903

Table 1: Accuracies for different learning rates on MNIST dataset using LeNet-5 model and AdaDelta optimizer

```
Learning rate 0.01
Test loss: 0.188755272975564
Test accuracy: 0.947
```

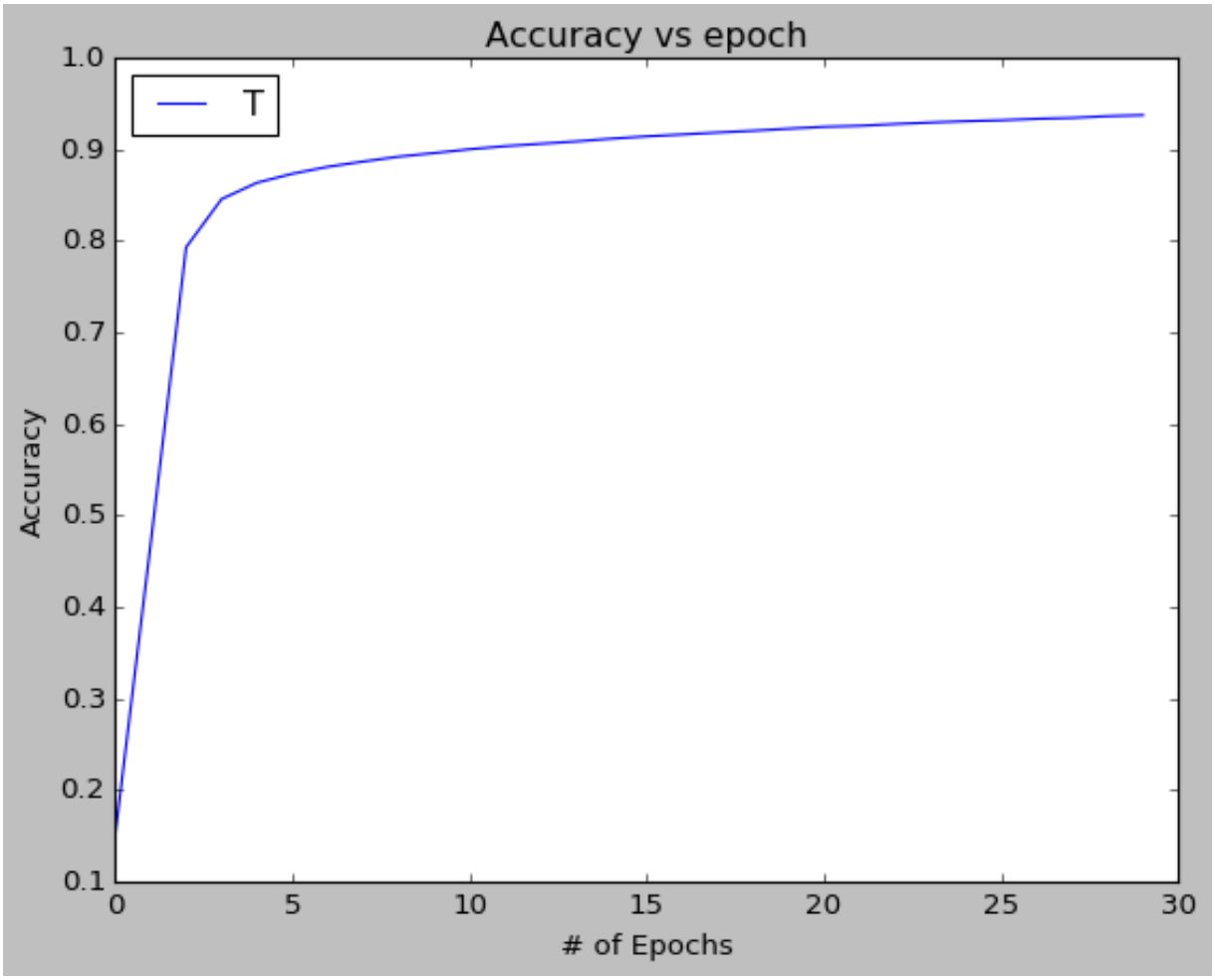

Fig 15: Accuracy vs epoch on MNIST dataset using LeNet-5 Model, AdaDelta Optimizer and Learning Rate of 0.01

```
Learning rate 0.001
Test loss: 0.9533560702323913
Test accuracy: 0.7411
```

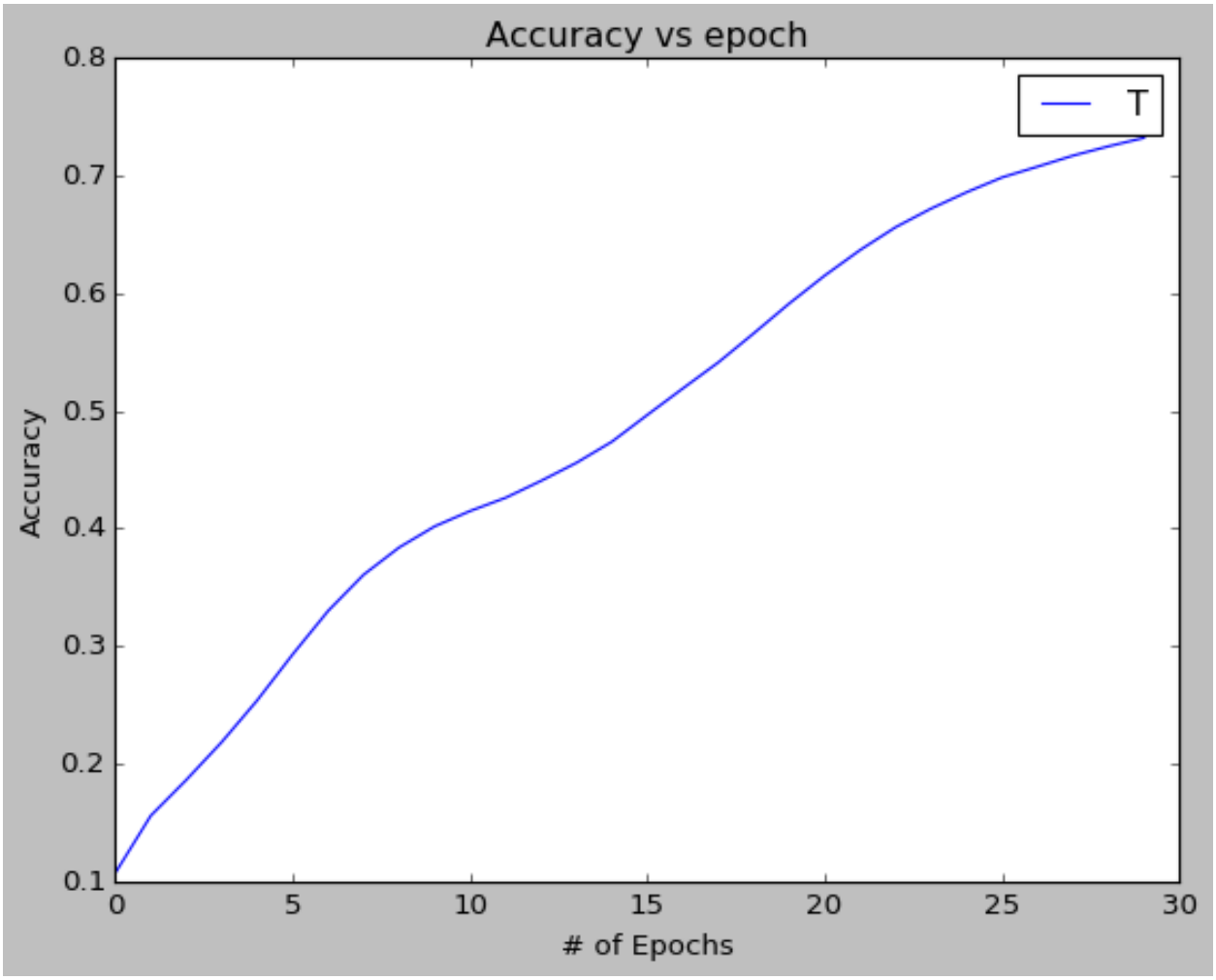

Fig 16: Accuracy vs epoch on MNIST dataset using LeNet-5 Model, AdaDelta Optimizer and Learning Rate of 0.001

```
Learning rate 0.0001
Test loss: 2.283204183959961
Test accuracy: 0.1395
```

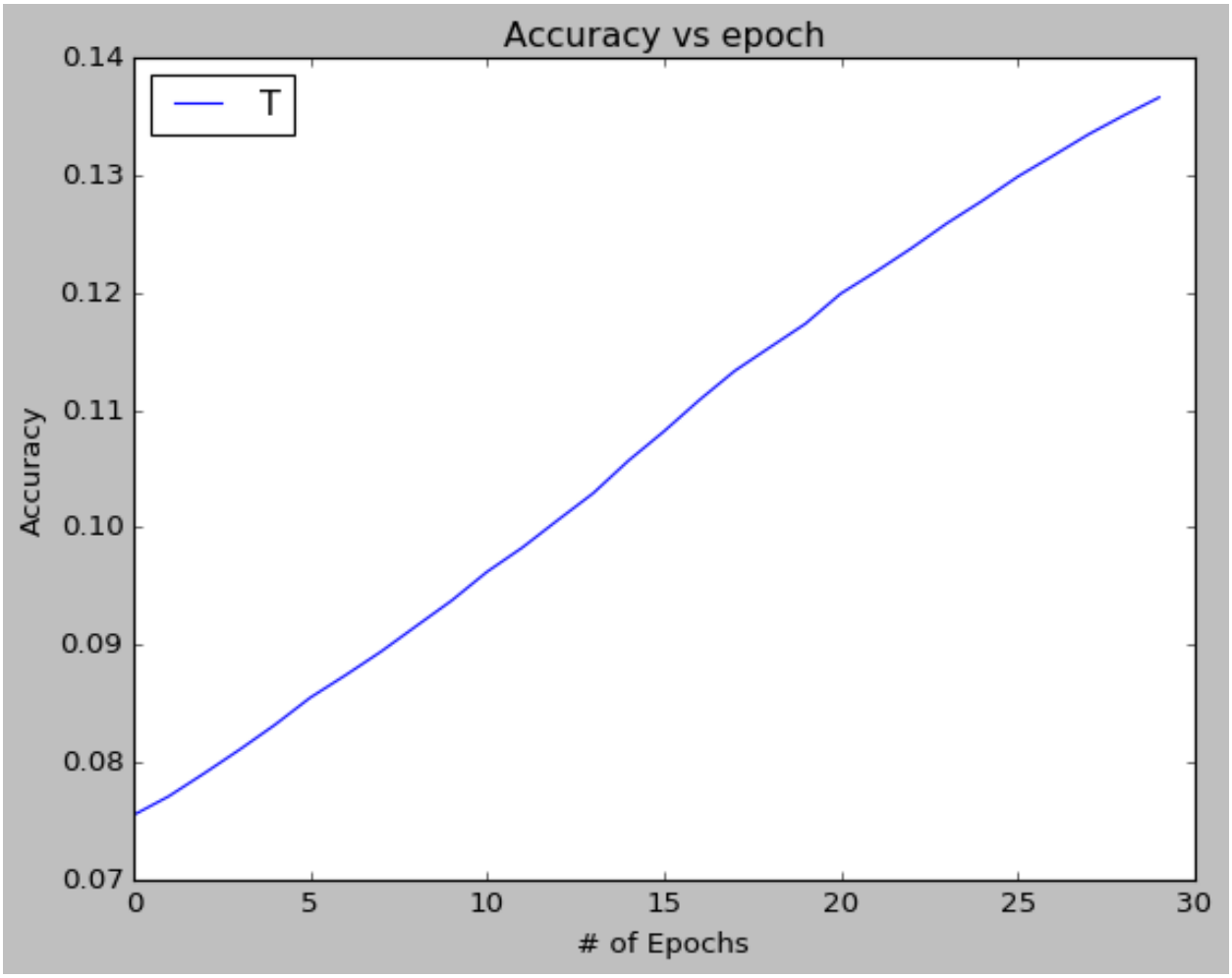

Fig 17: Accuracy vs epoch on MNIST dataset using LeNet-5 Model, AdaDelta Optimizer and Learning Rate of 0.0001

```
Learning rate 0.002
Test loss: 0.45455820784568784
Test accuracy: 0.8751
```

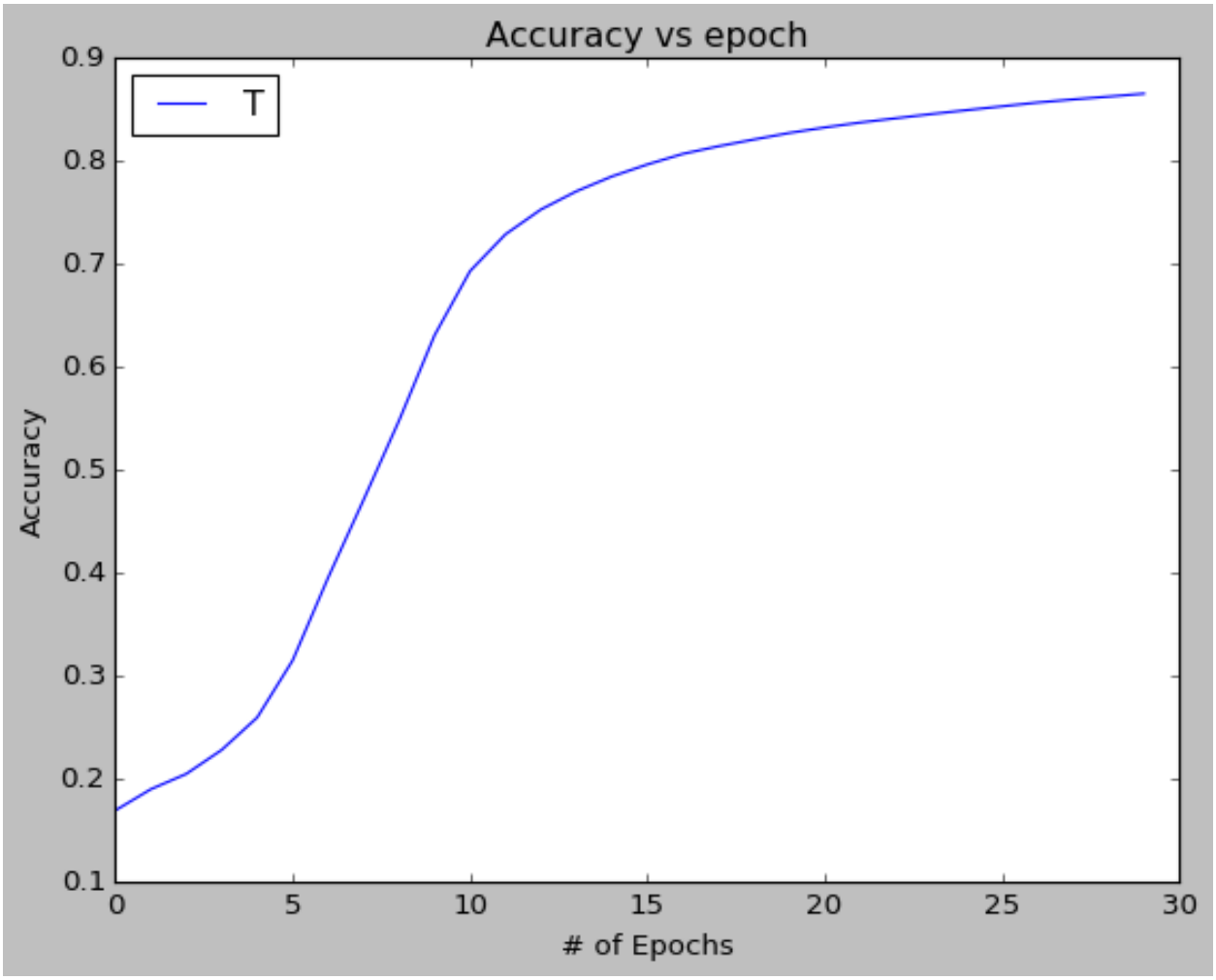

Fig 18: Accuracy vs epoch on MNIST dataset using LeNet-5 Model, AdaDelta Optimizer and Learning Rate of 0.002

```
Learning rate 1.0
Test loss: 0.052970787806792506
Test accuracy: 0.9903
```

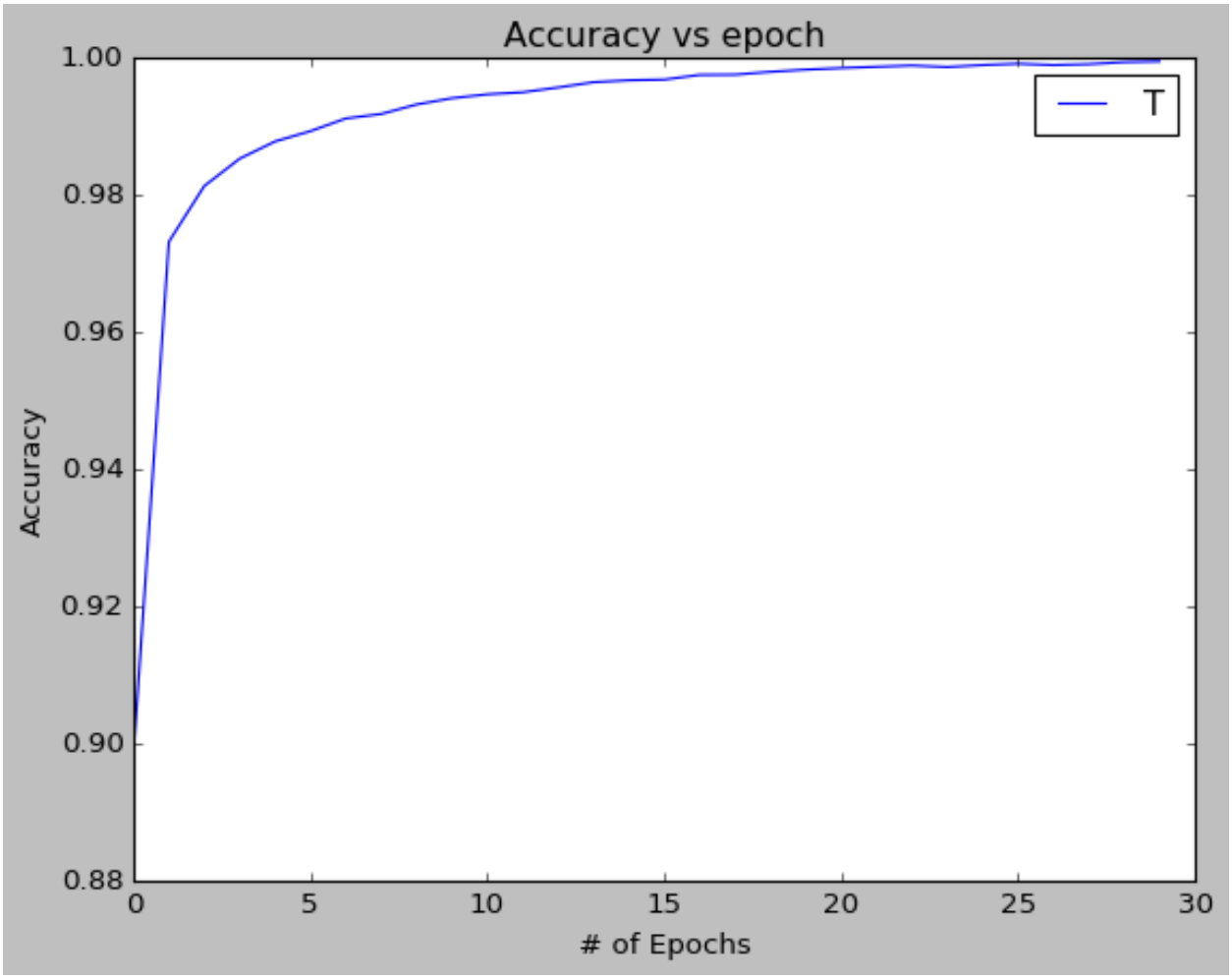

Fig 19: Accuracy vs epoch on MNIST dataset using LeNet-5 Model, AdaDelta Optimizer and Learning Rate of 1.0

PART 10:

By looking at all your results that you obtained in this assignment, comment on which optimization algorithm (AdaDelta or SGD) worked better in your case in your report (3 points). What did you observe by changing the learning rate to any of those 5 values in the previous step: Was all of those 5 learning rates useful, is using a smaller or a higher value better for the learning rate in SGD? (5 points) Did you see any tradeoff between the learning rate and convergence? (5 points) (In some plots, you should see that if you included more iterations/epochs, you would get even better accuracies while in others you may notice that you have already reached to a saturated point where including more iterations does not help). You do not need to submit your code for this step. Answer the questions in report.

Solution:

1) Comment on which optimization algorithm (AdaDelta or SGD) worked better in your case in your report (3 points)

Dataset	Optimizer	Model	Test Accuracy
MNIST	AdaDelta	ConvNet	0.9918
MNIST	AdaDelta	LeNet-5	0.9931
MNIST	SGD	LeNet-5	0.9771
MNIST	SGD	ConvNet	0.9841
Fashion MNIST	AdaDelta	LeNet-5	0.9062
Fashion MNIST	AdaDelta	ConvNet	0.9306
Fashion MNIST	SGD	LeNet-5	0.768
Fashion MNIST	SGD	ConvNet	0.885

Table 2: Comparison of the performance of different datasets and models on AdaDelta and SGD optimizers

From the above table 2, we can see that AdaDelta optimization algorithm works the best for MNIST Dataset using LeNet-5 model by achieving a high accuracy of 0.9931.
AdaDelta performs better because an exhaustive search was not carried out for the best learning rate for SGD optimizer although SGD performs better and converges faster, in theory.

2) What did you observe by changing the learning rate to any of those 5 values in the previous step: Was all of those 5 learning rates useful, is using a smaller or a higher value better for the learning rate in SGD? (5 points)

Learning Rate	Accuracy
0.0001	0.2292
0.001	0.896
0.002	0.9376
0.01	0.9771
1.0	0.9832

Table 3: Accuracies for different learning rates on MNIST dataset using LeNet-5 model and SGD optimizer

From the above table 3, we can see that:

- Using a higher value for the learning rate yields a higher test accuracy for the SGD optimization algorithm.
- Also, it can be observed (top to bottom on table 3) that as the learning rate increases, the accuracy also increases.
- The least learning rate of 0.0001 does not seem to be very useful as it gives a very low accuracy.
- It cannot be concluded that an increase in learning rate would increase the performance because higher or smaller values can end up providing bad results. It would be always better to fine tune the learning rate by observing it's global trend in relation to the accuracy.

3) Did you see any tradeoff between the learning rate and convergence? (5 points)

From figures 15, 16, 17, 18 and 19, it can be observed that:

- When a higher learning rate is reached at a particular epoch in the training process, the final accuracy still converges to a value very close to that of a different learning rate.
- So there was a tradeoff between convergence and a lower value of the learning rate.
- The closer we are to the optimal value of the learning rate, the more reliable and faster is the convergence to the minima of the cost function when the number of epochs is large.
- If the learning rate is too low, there is no convergence at all as it can particularly be seen in figure 17 with a very low learning rate of 0.0001.