In [ ]:
```
%matplotlib inline
```

SYDE 522 Assignment 4

Deep Networks and Convolution

Due: Nov 29, 2023

The purpose of this assignment is to train some deep networks to recognize images. We will start with MNIST (the standard hand-written numbers dataset) and then go on to the harder CIFAR-10 (recognizing pictures in 10 categories).

All of these datasets are publically available, and for this assignment you should use the Python package `tensorflow` to implement and train the neural network. On most python systems this can be installed with:

```
pip install tensorflow
```

For more information on installing, see https://www.tensorflow.org/install

For this assignment, you will not need to install the GPU version of `tensorflow`. If you want to work with larger deep learning models and make things run faster (for your project, for example), then instructions for doing so are at https://www.tensorflow.org/install/gpu

This assignment sheet is a Jupyter Notebook file https://jupyter.org/ which you can use if you want as a starting point.

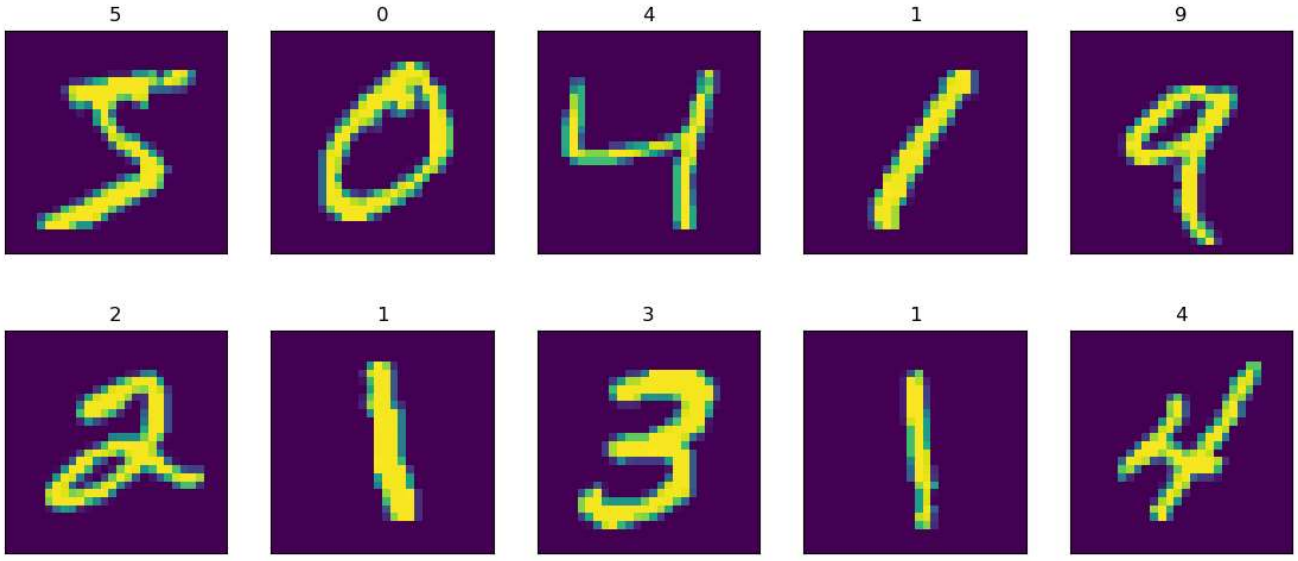
Part 1: MNIST

First, we download the data files. They are already split into a training set and a test set.

In [ ]:
```
import tensorflow.keras.datasets.mnist as mnist

(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train = x_train / 255.0   # rescale the images to be between 0 and 1
x_test = x_test / 255.0     # rescale the images to be between 0 and 1
```

Let's show the first 10 training images and their category labels.

In [ ]:
```
import matplotlib.pyplot as plt
plt.figure(figsize=(14,6))
for i in range(10):
    plt.subplot(2, 5, i+1)
    plt.imshow(x_train[i])
    plt.xticks([])
    plt.yticks([])
    plt.title('%d' % y_train[i])
plt.show()
```



The `y_train` and `y_target` values are currently integers (0 to 9). The output from our network is a vector where one value should be 1 and the others should be 0 (one-hot encoding), so we need to convert to that format. (Note: there is a special loss function `sparse_categorical_crossentropy` that automatically converts for us. We'll use that later, but for now do this manually, just to keep things clear.)

In [ ]:
```
import numpy as np

y_train_target = np.eye(10)[y_train]
y_test_target = np.eye(10)[y_test]
```

```
print('original target:', y_train[0])
print('  vector target:', y_train_target[0])
```

```
original target: 5
  vector target: [0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
```

Now we define the model using Keras https://keras.io/, which is meant for being able to quickly define all the different structures you might want to add to your network. We use `Sequential` to indicate that this is just a big feed-forward network, and we define each layer in turn. `Dense` is a layer where all the components are connected to all of the previous layer's outputs.

In [ ]:
```
import tensorflow as tf

model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),    # input is a 28x28 image
  tf.keras.layers.Dense(32, activation='relu'),     # 32 neurons in the middle "hidden" layer
  tf.keras.layers.Dense(10, activation='relu')      # 10 outputs (one for each category)
])

# define what we want to minimize (the thing that we take the derivative of to get the weight changes)
def my_loss(y_true, y_predict):
    return (y_true-y_predict)**2

model.compile(optimizer=tf.keras.optimizers.SGD(learning_rate=0.1), # use stochastic gradient descent
              loss=my_loss,
              metrics=['accuracy']  # in addition to the loss, also compute the categorization accuracy
             )
```

```
WARNING:absl:At this time, the v2.11+ optimizer `tf.keras.optimizers.SGD` runs slowly on M1/M2 Macs, please use the legacy Keras optimizer instead, loc
ated at `tf.keras.optimizers.legacy.SGD`.
```

Before doing any training, let's see how well it performs.

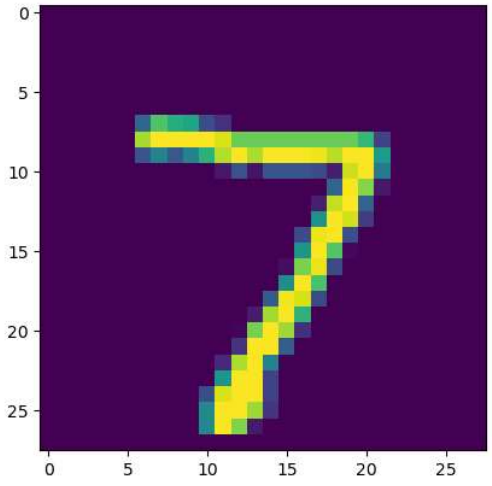
In [ ]:
```
loss, accuracy = model.evaluate(x_test,  y_test_target)
```

```
313/313 [==============================] - 0s 421us/step - loss: 0.1334 - accuracy: 0.0607
```

As expected, it's pretty bad, since we haven't done any training yet. The accuracy is near 10%, which is what we would expect by chance. But what actual numbers is it outputting?

In [ ]:
```
output = model.predict(x_test)
category = np.argmax(output, axis=1)

plt.imshow(x_test[0])
plt.show()
print('actual output from network:', output[0])
print('category (the largest output):', category[0])
```

```
313/313 [==============================] - 0s 444us/step
```




```
actual output from network: [0.23694628 0.         0.         0.26332176 0.         0.
 0.17939256 0.         0.         0.        ]
category (the largest output): 3
```

So it currently thinks the first image in the test dataset is is a 6, when it should actually be a 7 (note: when you run this, you may get a different number, as each network starts out randomly different).

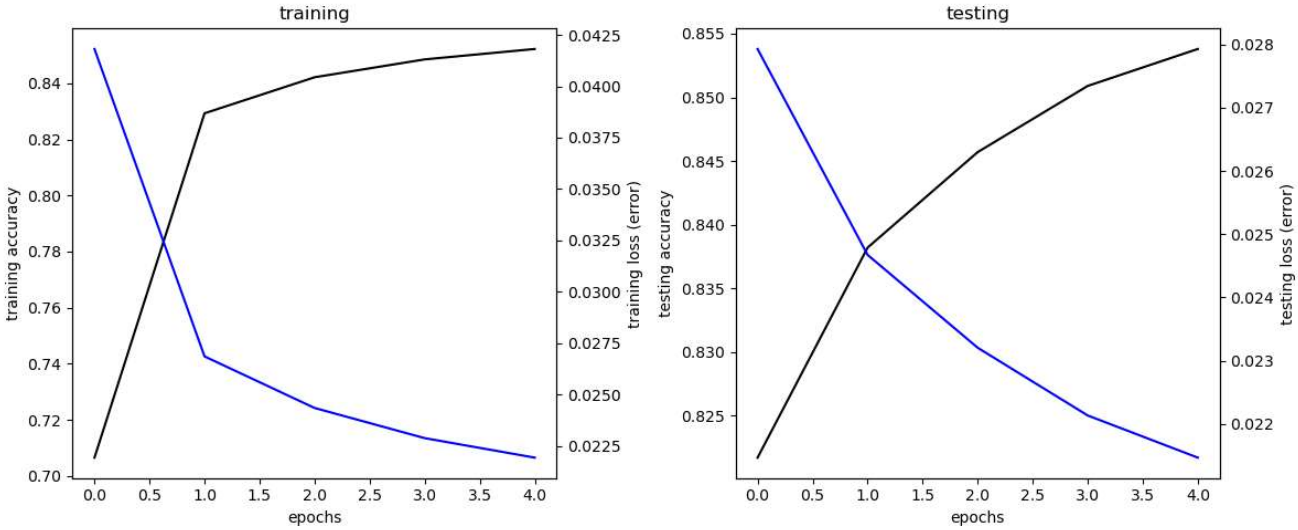
Now let's try training the model. We'll just do 5 epochs of training (5 times through all the training data). While it is training, we also tell it to see how well the model is performing on the testing data (technically this is considered to be validation data, since we're looking at it while we are doing training. Real testing data would only ever be used at the very end after all training has finished).

In [ ]:
```
model.fit(x_train, y_train_target, epochs=5, validation_data=(x_test, y_test_target));
```

```
Epoch 1/5
1875/1875 [==============================] - 1s 577us/step - loss: 0.0398 - accuracy: 0.7398 - val_loss: 0.0270 - val_accuracy: 0.8329
Epoch 2/5
1875/1875 [==============================] - 1s 535us/step - loss: 0.0264 - accuracy: 0.8385 - val_loss: 0.0243 - val_accuracy: 0.8486
Epoch 3/5
1875/1875 [==============================] - 1s 568us/step - loss: 0.0242 - accuracy: 0.8489 - val_loss: 0.0227 - val_accuracy: 0.8570
Epoch 4/5
1875/1875 [==============================] - 1s 533us/step - loss: 0.0228 - accuracy: 0.8555 - val_loss: 0.0217 - val_accuracy: 0.8621
Epoch 5/5
1875/1875 [==============================] - 1s 534us/step - loss: 0.0218 - accuracy: 0.8601 - val_loss: 0.0209 - val_accuracy: 0.8650
{'loss': [0.03977584093809128, 0.02640184946358204, 0.024185996502637863, 0.02276952937245369, 0.021783282980322838], 'accuracy': [0.7398333549499512,
0.8385000228881836, 0.8489333391189575, 0.8555333614349365, 0.8600833415985107], 'val_loss': [0.0269723329693079, 0.02429671213030815, 0.02273800596594
8105, 0.02165607176721096, 0.020931407809257507], 'val_accuracy': [0.8328999876976013, 0.8485999703407288, 0.8569999933242798, 0.8621000051498413, 0.86
50000095367432]}
```

We can plot the categorization accuracy over time and the error over time for both the training set and the test set.

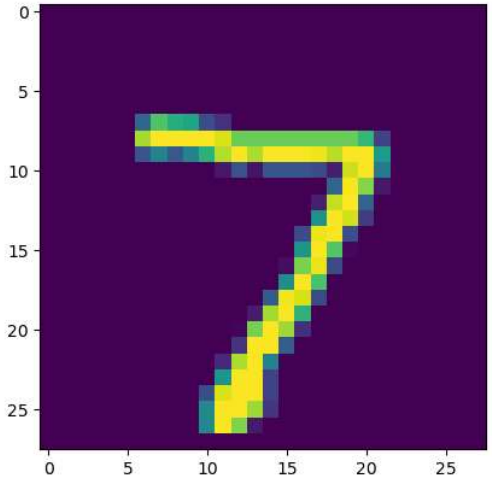
In [ ]:
```
plt.figure(figsize=(12,5))
plt.subplot(1, 2, 1)
plt.plot(model.history.history['accuracy'], c='k')
plt.ylabel('training accuracy')
plt.xlabel('epochs')
plt.twinx()
plt.plot(model.history.history['loss'], c='b')
plt.ylabel('training loss (error)')
plt.title('training')

plt.subplot(1, 2, 2)
plt.plot(model.history.history['val_accuracy'], c='k')
plt.ylabel('testing accuracy')
plt.xlabel('epochs')
plt.twinx()
plt.plot(model.history.history['val_loss'], c='b')
plt.ylabel('testing loss (error)')
plt.title('testing')
plt.tight_layout()
plt.show()
```



Now let's see how well it performs on the test set.

In [ ]:
```
output = model.predict(x_test)
category = np.argmax(output, axis=1)

plt.imshow(x_test[0])
plt.show()
print('actual output from network:', output[0])
print('category (the largest output):', category[0])
```

```
313/313 [==============================] - 0s 379us/step
```



```
actual output from network: [0.        0.        0.        0.        0.        0.        0.
 0.8715112 0.        0.        ]
category (the largest output): 7
```

Well it learned the first item well, but it's only getting around 85% on the test set (which is also about what it's getting on the training set). To get a better sense of what it's doing, let's generate the *confusion* matrix. This indicates what category it puts things in, as compared to what category they should be in.

In [ ]:
```
confusion=np.zeros((10,10), dtype=int)
np.add.at(confusion, (category, y_test), 1)
print(confusion)
```

```
[[ 966    1   16  403    2   12   11    2    7   13]
 [   0 1114    2    7    1    1    3   15    2    5]
 [   2    4  967   78    6    2    2   13    7    1]
 [   0    0    0    0    0    0    0    0    0    0]
 [   0    0    6    6  935    4    9    5    8   26]
 [   4    1    5  255    0  838   18    3   17    6]
 [   3    6    5    6    9   11  909    0    5    1]
 [   1    1   10   43    2    3    1  973   13   14]
 [   4    8   18  189    4   15    5    0  908   15]
 [   0    0    3   23   23    6    0   17    7  928]]
```

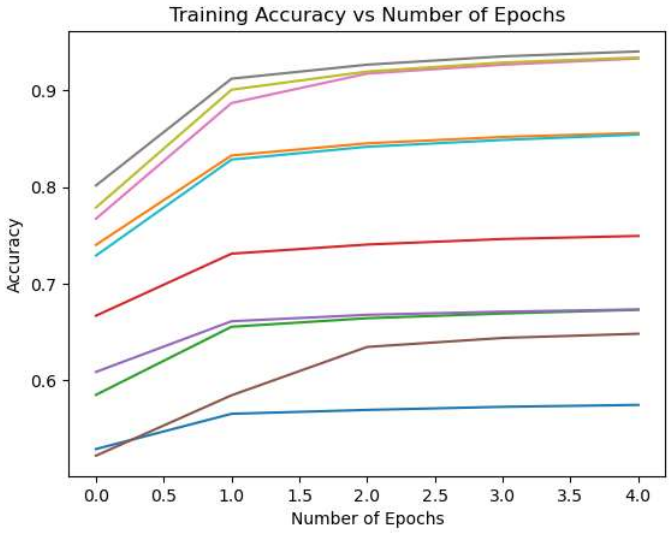
The ideal confusion matrix will have ~1000 along the diagonal and 0 everywhere else. According to this confusion matrix (yours will be different), 9's got mis-calssified as 4's a lot (and vice-versa).

1. [1 mark]: Run the MNIST model as defined here 10 times. Note that you have to re-create the `model` each time: if you just run `model.fit` over again, then it will just train the same model, continuing from where it left off. Each of the 10 models should be trained for 5 epochs. Make a single plot with training accuracy vs epoch for all 10 runs. You should see that each time you train the model, it behaves differently. Why does this happen?

In [ ]:
```
plt.figure()
for i in range(10):
    # Create new model
    model = tf.keras.models.Sequential([
      tf.keras.layers.Flatten(input_shape=(28, 28)),   # input is a 28x28 image
      tf.keras.layers.Dense(32, activation='relu'),    # 32 neurons in the middle "hidden" layer
      tf.keras.layers.Dense(10, activation='relu')     # 10 outputs (one for each category)
    ])

    # Compile model
    model.compile(optimizer=tf.keras.optimizers.SGD(learning_rate=0.1), # use stochastic gradient descent
                  loss=my_loss,
                  metrics=['accuracy']  # in addition to the loss, also compute the categorization accuracy
                 )

    # Train each model for 5 epochs
    history = model.fit(x_train, y_train_target, epochs=5, validation_data=(x_test, y_test_target));

    # Plot the training accuracy on the same graph
    plt.plot(history.history['accuracy'])

plt.title("Training Accuracy vs Number of Epochs")
plt.ylabel("Training Accuracy")
plt.xlabel("Number of Epochs")
plt.show()
```

```
Epoch 1/5
1875/1875 [==============================] - 1s 695us/step - loss: 0.0571 - accuracy: 0.5290 - val_loss: 0.0509 - val_accuracy: 0.5615
Epoch 2/5
1875/1875 [==============================] - 1s 520us/step - loss: 0.0504 - accuracy: 0.5655 - val_loss: 0.0492 - val_accuracy: 0.5675
Epoch 3/5
1875/1875 [==============================] - 1s 526us/step - loss: 0.0490 - accuracy: 0.5695 - val_loss: 0.0483 - val_accuracy: 0.5709
Epoch 4/5
1875/1875 [==============================] - 1s 652us/step - loss: 0.0482 - accuracy: 0.5727 - val_loss: 0.0477 - val_accuracy: 0.5724
Epoch 5/5
1875/1875 [==============================] - 1s 542us/step - loss: 0.0476 - accuracy: 0.5747 - val_loss: 0.0472 - val_accuracy: 0.5745
Epoch 1/5
1875/1875 [==============================] - 1s 562us/step - loss: 0.0393 - accuracy: 0.7403 - val_loss: 0.0274 - val_accuracy: 0.8301
Epoch 2/5
1875/1875 [==============================] - 1s 540us/step - loss: 0.0264 - accuracy: 0.8328 - val_loss: 0.0243 - val_accuracy: 0.8451
Epoch 3/5
1875/1875 [==============================] - 1s 521us/step - loss: 0.0239 - accuracy: 0.8454 - val_loss: 0.0228 - val_accuracy: 0.8522
Epoch 4/5
1875/1875 [==============================] - 1s 520us/step - loss: 0.0225 - accuracy: 0.8519 - val_loss: 0.0218 - val_accuracy: 0.8539
Epoch 5/5
1875/1875 [==============================] - 1s 524us/step - loss: 0.0216 - accuracy: 0.8561 - val_loss: 0.0214 - val_accuracy: 0.8561
Epoch 1/5
1875/1875 [==============================] - 1s 557us/step - loss: 0.0523 - accuracy: 0.5851 - val_loss: 0.0431 - val_accuracy: 0.6507
Epoch 2/5
1875/1875 [==============================] - 1s 516us/step - loss: 0.0422 - accuracy: 0.6555 - val_loss: 0.0407 - val_accuracy: 0.6618
Epoch 3/5
1875/1875 [==============================] - 1s 518us/step - loss: 0.0403 - accuracy: 0.6644 - val_loss: 0.0396 - val_accuracy: 0.6667
Epoch 4/5
1875/1875 [==============================] - 1s 554us/step - loss: 0.0392 - accuracy: 0.6693 - val_loss: 0.0389 - val_accuracy: 0.6696
Epoch 5/5
1875/1875 [==============================] - 1s 545us/step - loss: 0.0385 - accuracy: 0.6731 - val_loss: 0.0384 - val_accuracy: 0.6722
Epoch 1/5
1875/1875 [==============================] - 1s 564us/step - loss: 0.0460 - accuracy: 0.6669 - val_loss: 0.0375 - val_accuracy: 0.7224
Epoch 2/5
1875/1875 [==============================] - 1s 525us/step - loss: 0.0363 - accuracy: 0.7312 - val_loss: 0.0353 - val_accuracy: 0.7315
Epoch 3/5
1875/1875 [==============================] - 1s 536us/step - loss: 0.0342 - accuracy: 0.7406 - val_loss: 0.0338 - val_accuracy: 0.7398
Epoch 4/5
1875/1875 [==============================] - 1s 525us/step - loss: 0.0330 - accuracy: 0.7463 - val_loss: 0.0328 - val_accuracy: 0.7439
Epoch 5/5
1875/1875 [==============================] - 1s 517us/step - loss: 0.0321 - accuracy: 0.7495 - val_loss: 0.0321 - val_accuracy: 0.7465
Epoch 1/5
1875/1875 [==============================] - 1s 560us/step - loss: 0.0498 - accuracy: 0.6088 - val_loss: 0.0423 - val_accuracy: 0.6602
Epoch 2/5
1875/1875 [==============================] - 1s 518us/step - loss: 0.0415 - accuracy: 0.6612 - val_loss: 0.0402 - val_accuracy: 0.6691
Epoch 3/5
1875/1875 [==============================] - 1s 524us/step - loss: 0.0399 - accuracy: 0.6679 - val_loss: 0.0393 - val_accuracy: 0.6710
Epoch 4/5
1875/1875 [==============================] - 1s 519us/step - loss: 0.0389 - accuracy: 0.6712 - val_loss: 0.0384 - val_accuracy: 0.6753
Epoch 5/5
1875/1875 [==============================] - 1s 517us/step - loss: 0.0382 - accuracy: 0.6734 - val_loss: 0.0379 - val_accuracy: 0.6763
Epoch 1/5
1875/1875 [==============================] - 1s 564us/step - loss: 0.0583 - accuracy: 0.5221 - val_loss: 0.0520 - val_accuracy: 0.5588
Epoch 2/5
1875/1875 [==============================] - 1s 520us/step - loss: 0.0497 - accuracy: 0.5846 - val_loss: 0.0452 - val_accuracy: 0.6272
Epoch 3/5
1875/1875 [==============================] - 1s 547us/step - loss: 0.0442 - accuracy: 0.6347 - val_loss: 0.0430 - val_accuracy: 0.6417
Epoch 4/5
1875/1875 [==============================] - 1s 551us/step - loss: 0.0424 - accuracy: 0.6440 - val_loss: 0.0418 - val_accuracy: 0.6477
Epoch 5/5
1875/1875 [==============================] - 1s 518us/step - loss: 0.0415 - accuracy: 0.6483 - val_loss: 0.0410 - val_accuracy: 0.6507
Epoch 1/5
1875/1875 [==============================] - 1s 557us/step - loss: 0.0369 - accuracy: 0.7673 - val_loss: 0.0266 - val_accuracy: 0.8414
Epoch 2/5
1875/1875 [==============================] - 1s 539us/step - loss: 0.0222 - accuracy: 0.8870 - val_loss: 0.0183 - val_accuracy: 0.9146
Epoch 3/5
1875/1875 [==============================] - 1s 542us/step - loss: 0.0177 - accuracy: 0.9175 - val_loss: 0.0163 - val_accuracy: 0.9275
Epoch 4/5
1875/1875 [==============================] - 1s 519us/step - loss: 0.0161 - accuracy: 0.9267 - val_loss: 0.0151 - val_accuracy: 0.9342
Epoch 5/5
1875/1875 [==============================] - 1s 519us/step - loss: 0.0149 - accuracy: 0.9331 - val_loss: 0.0144 - val_accuracy: 0.9364
Epoch 1/5
1875/1875 [==============================] - 1s 561us/step - loss: 0.0339 - accuracy: 0.8017 - val_loss: 0.0205 - val_accuracy: 0.9033
Epoch 2/5
1875/1875 [==============================] - 1s 518us/step - loss: 0.0188 - accuracy: 0.9122 - val_loss: 0.0166 - val_accuracy: 0.9244
Epoch 3/5
1875/1875 [==============================] - 1s 518us/step - loss: 0.0160 - accuracy: 0.9267 - val_loss: 0.0146 - val_accuracy: 0.9347
Epoch 4/5
1875/1875 [==============================] - 1s 522us/step - loss: 0.0144 - accuracy: 0.9353 - val_loss: 0.0134 - val_accuracy: 0.9403
Epoch 5/5
1875/1875 [==============================] - 1s 516us/step - loss: 0.0133 - accuracy: 0.9404 - val_loss: 0.0127 - val_accuracy: 0.9428
Epoch 1/5
1875/1875 [==============================] - 1s 556us/step - loss: 0.0364 - accuracy: 0.7789 - val_loss: 0.0236 - val_accuracy: 0.8857
Epoch 2/5
1875/1875 [==============================] - 1s 517us/step - loss: 0.0210 - accuracy: 0.9008 - val_loss: 0.0182 - val_accuracy: 0.9194
Epoch 3/5
1875/1875 [==============================] - 1s 520us/step - loss: 0.0175 - accuracy: 0.9196 - val_loss: 0.0159 - val_accuracy: 0.9295
Epoch 4/5
1875/1875 [==============================] - 1s 520us/step - loss: 0.0158 - accuracy: 0.9290 - val_loss: 0.0147 - val_accuracy: 0.9349
Epoch 5/5
1875/1875 [==============================] - 1s 516us/step - loss: 0.0147 - accuracy: 0.9341 - val_loss: 0.0140 - val_accuracy: 0.9386
Epoch 1/5
1875/1875 [==============================] - 1s 564us/step - loss: 0.0408 - accuracy: 0.7292 - val_loss: 0.0283 - val_accuracy: 0.8241
Epoch 2/5
1875/1875 [==============================] - 1s 530us/step - loss: 0.0271 - accuracy: 0.8285 - val_loss: 0.0249 - val_accuracy: 0.8415
```

```
Epoch 3/5
1875/1875 [==============================] - 1s 541us/step - loss: 0.0245 - accuracy: 0.8418 - val_loss: 0.0235 - val_accuracy: 0.8473
Epoch 4/5
1875/1875 [==============================] - 1s 519us/step - loss: 0.0230 - accuracy: 0.8488 - val_loss: 0.0223 - val_accuracy: 0.8531
Epoch 5/5
1875/1875 [==============================] - 1s 518us/step - loss: 0.0220 - accuracy: 0.8544 - val_loss: 0.0216 - val_accuracy: 0.8555
```



The reason each model behaves differently is because each model is initialized with random weights. The differences in the starting weights lead to different outcomes. Furthermore, the SGD optimizer is stochastic, which means that they introduce randomness while updating the parameters. These factors lead to the model behaving differently each time it is trained.

2. [1 mark]: Do the same thing as question 1, but train for 20 epochs. Plot the training error vs epochs for 10 different runs on one plot. On a separate plot show the testing error vs epochs for the 10 different runs. How does the testing error compare to the training error? Do the models all eventually learn to solve the problem well? What happens to them?

In [ ]:
```python
import numpy as np

train_errs = []
test_errs = []

def get_error(accuracy):
    return 1 - accuracy

err_func = np.vectorize(get_error)

num_models = 10

for i in range(num_models):
    # Create new model
    model = tf.keras.models.Sequential([
      tf.keras.layers.Flatten(input_shape=(28, 28)),   # input is a 28x28 image
      tf.keras.layers.Dense(32, activation='relu'),    # 32 neurons in the middle "hidden" layer
      tf.keras.layers.Dense(10, activation='relu')     # 10 outputs (one for each category)
    ])

    # Compile model
    model.compile(optimizer=tf.keras.optimizers.SGD(learning_rate=0.1), # use stochastic gradient descent
                  loss=my_loss,
                  metrics=['accuracy']  # in addition to the loss, also compute the categorization accuracy
                 )

    # Train each model for 20 epochs
    history = model.fit(x_train, y_train_target, epochs=20, validation_data=(x_test, y_test_target));

    # Store training and test errors
    train_err = err_func(history.history['accuracy'])
    train_errs.append(train_err)

    test_err = err_func(history.history['val_accuracy'])
    test_errs.append(test_err)

# Plot training errors
plt.figure()
for i in range(num_models):
    plt.plot(train_errs[i])
plt.title("Training Error vs Number of Epochs")
plt.ylabel("Training Error")
plt.xlabel("Number of Epochs")
plt.show()

# Plot test errors
plt.figure()
for i in range(num_models):
    plt.plot(test_errs[i])
plt.title("Test Error vs Number of Epochs")
```

```
plt.ylabel("Test Error")
plt.xlabel("Number of Epochs")
plt.show()
```

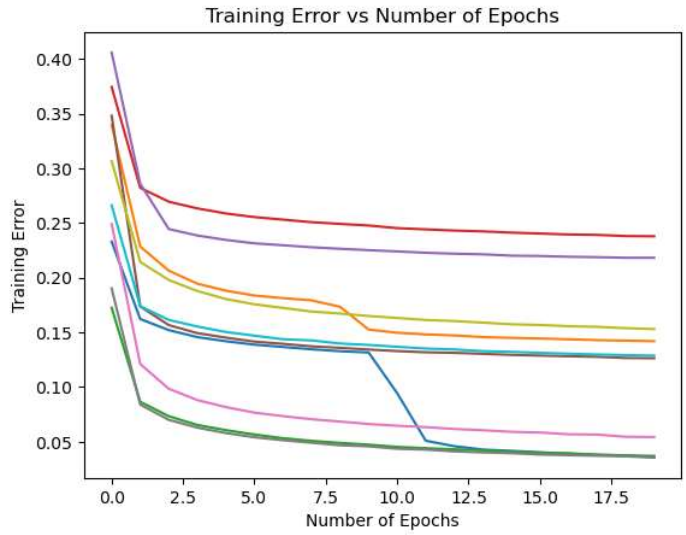
```
WARNING:absl:At this time, the v2.11+ optimizer `tf.keras.optimizers.SGD` runs slowly on M1/M2 Macs, please use the legacy Keras optimizer instead, located at `tf.keras.optimizers.legacy.SGD`.
Epoch 1/20
1875/1875 [==============================] - 1s 718us/step - loss: 0.0356 - accuracy: 0.7671 - val_loss: 0.0262 - val_accuracy: 0.8329
Epoch 2/20
1875/1875 [==============================] - 1s 637us/step - loss: 0.0251 - accuracy: 0.8378 - val_loss: 0.0237 - val_accuracy: 0.8433
Epoch 3/20
1875/1875 [==============================] - 1s 628us/step - loss: 0.0230 - accuracy: 0.8480 - val_loss: 0.0222 - val_accuracy: 0.8516
Epoch 4/20
1875/1875 [==============================] - 1s 534us/step - loss: 0.0217 - accuracy: 0.8543 - val_loss: 0.0213 - val_accuracy: 0.8569
Epoch 5/20
1875/1875 [==============================] - 1s 534us/step - loss: 0.0209 - accuracy: 0.8581 - val_loss: 0.0207 - val_accuracy: 0.8580
Epoch 6/20
1875/1875 [==============================] - 1s 537us/step - loss: 0.0203 - accuracy: 0.8612 - val_loss: 0.0203 - val_accuracy: 0.8600
Epoch 7/20
1875/1875 [==============================] - 1s 533us/step - loss: 0.0198 - accuracy: 0.8634 - val_loss: 0.0197 - val_accuracy: 0.8622
Epoch 8/20
1875/1875 [==============================] - 1s 533us/step - loss: 0.0193 - accuracy: 0.8655 - val_loss: 0.0195 - val_accuracy: 0.8638
Epoch 9/20
1875/1875 [==============================] - 1s 533us/step - loss: 0.0190 - accuracy: 0.8672 - val_loss: 0.0191 - val_accuracy: 0.8652
Epoch 10/20
1875/1875 [==============================] - 1s 534us/step - loss: 0.0186 - accuracy: 0.8684 - val_loss: 0.0188 - val_accuracy: 0.8673
Epoch 11/20
1875/1875 [==============================] - 1s 540us/step - loss: 0.0161 - accuracy: 0.9056 - val_loss: 0.0129 - val_accuracy: 0.9423
Epoch 12/20
1875/1875 [==============================] - 1s 535us/step - loss: 0.0120 - accuracy: 0.9491 - val_loss: 0.0118 - val_accuracy: 0.9504
Epoch 13/20
1875/1875 [==============================] - 1s 533us/step - loss: 0.0111 - accuracy: 0.9541 - val_loss: 0.0112 - val_accuracy: 0.9520
Epoch 14/20
1875/1875 [==============================] - 1s 532us/step - loss: 0.0106 - accuracy: 0.9572 - val_loss: 0.0109 - val_accuracy: 0.9537
Epoch 15/20
1875/1875 [==============================] - 1s 533us/step - loss: 0.0103 - accuracy: 0.9584 - val_loss: 0.0108 - val_accuracy: 0.9549
Epoch 16/20
1875/1875 [==============================] - 1s 534us/step - loss: 0.0099 - accuracy: 0.9597 - val_loss: 0.0104 - val_accuracy: 0.9554
Epoch 17/20
1875/1875 [==============================] - 1s 541us/step - loss: 0.0097 - accuracy: 0.9609 - val_loss: 0.0101 - val_accuracy: 0.9578
Epoch 18/20
1875/1875 [==============================] - 1s 618us/step - loss: 0.0095 - accuracy: 0.9621 - val_loss: 0.0101 - val_accuracy: 0.9582
Epoch 19/20
1875/1875 [==============================] - 1s 536us/step - loss: 0.0093 - accuracy: 0.9625 - val_loss: 0.0098 - val_accuracy: 0.9588
Epoch 20/20
1875/1875 [==============================] - 1s 534us/step - loss: 0.0091 - accuracy: 0.9641 - val_loss: 0.0096 - val_accuracy: 0.9604
WARNING:absl:At this time, the v2.11+ optimizer `tf.keras.optimizers.SGD` runs slowly on M1/M2 Macs, please use the legacy Keras optimizer instead, located at `tf.keras.optimizers.legacy.SGD`.
Epoch 1/20
1875/1875 [==============================] - 1s 578us/step - loss: 0.0502 - accuracy: 0.6604 - val_loss: 0.0398 - val_accuracy: 0.7488
Epoch 2/20
1875/1875 [==============================] - 1s 542us/step - loss: 0.0378 - accuracy: 0.7715 - val_loss: 0.0361 - val_accuracy: 0.7939
Epoch 3/20
1875/1875 [==============================] - 1s 533us/step - loss: 0.0352 - accuracy: 0.7936 - val_loss: 0.0342 - val_accuracy: 0.8041
Epoch 4/20
1875/1875 [==============================] - 1s 532us/step - loss: 0.0337 - accuracy: 0.8055 - val_loss: 0.0332 - val_accuracy: 0.8135
Epoch 5/20
1875/1875 [==============================] - 1s 531us/step - loss: 0.0326 - accuracy: 0.8119 - val_loss: 0.0325 - val_accuracy: 0.8153
Epoch 6/20
1875/1875 [==============================] - 1s 531us/step - loss: 0.0319 - accuracy: 0.8164 - val_loss: 0.0318 - val_accuracy: 0.8152
Epoch 7/20
1875/1875 [==============================] - 1s 532us/step - loss: 0.0313 - accuracy: 0.8187 - val_loss: 0.0314 - val_accuracy: 0.8182
Epoch 8/20
1875/1875 [==============================] - 1s 531us/step - loss: 0.0308 - accuracy: 0.8207 - val_loss: 0.0311 - val_accuracy: 0.8186
Epoch 9/20
1875/1875 [==============================] - 1s 551us/step - loss: 0.0289 - accuracy: 0.8266 - val_loss: 0.0232 - val_accuracy: 0.8418
Epoch 10/20
1875/1875 [==============================] - 1s 535us/step - loss: 0.0219 - accuracy: 0.8475 - val_loss: 0.0220 - val_accuracy: 0.8452
Epoch 11/20
1875/1875 [==============================] - 1s 532us/step - loss: 0.0212 - accuracy: 0.8503 - val_loss: 0.0216 - val_accuracy: 0.8462
Epoch 12/20
1875/1875 [==============================] - 1s 574us/step - loss: 0.0208 - accuracy: 0.8519 - val_loss: 0.0213 - val_accuracy: 0.8467
Epoch 13/20
1875/1875 [==============================] - 1s 564us/step - loss: 0.0205 - accuracy: 0.8528 - val_loss: 0.0211 - val_accuracy: 0.8474
Epoch 14/20
1875/1875 [==============================] - 1s 536us/step - loss: 0.0202 - accuracy: 0.8543 - val_loss: 0.0209 - val_accuracy: 0.8490
Epoch 15/20
1875/1875 [==============================] - 1s 531us/step - loss: 0.0200 - accuracy: 0.8551 - val_loss: 0.0207 - val_accuracy: 0.8495
Epoch 16/20
1875/1875 [==============================] - 1s 537us/step - loss: 0.0198 - accuracy: 0.8556 - val_loss: 0.0206 - val_accuracy: 0.8503
Epoch 17/20
1875/1875 [==============================] - 1s 531us/step - loss: 0.0196 - accuracy: 0.8563 - val_loss: 0.0203 - val_accuracy: 0.8511
Epoch 18/20
1875/1875 [==============================] - 1s 531us/step - loss: 0.0194 - accuracy: 0.8571 - val_loss: 0.0203 - val_accuracy: 0.8515
Epoch 19/20
1875/1875 [==============================] - 1s 530us/step - loss: 0.0193 - accuracy: 0.8576 - val_loss: 0.0201 - val_accuracy: 0.8517
Epoch 20/20
1875/1875 [==============================] - 1s 531us/step - loss: 0.0192 - accuracy: 0.8581 - val_loss: 0.0200 - val_accuracy: 0.8521
WARNING:absl:At this time, the v2.11+ optimizer `tf.keras.optimizers.SGD` runs slowly on M1/M2 Macs, please use the legacy Keras optimizer instead, located at `tf.keras.optimizers.legacy.SGD`.
```

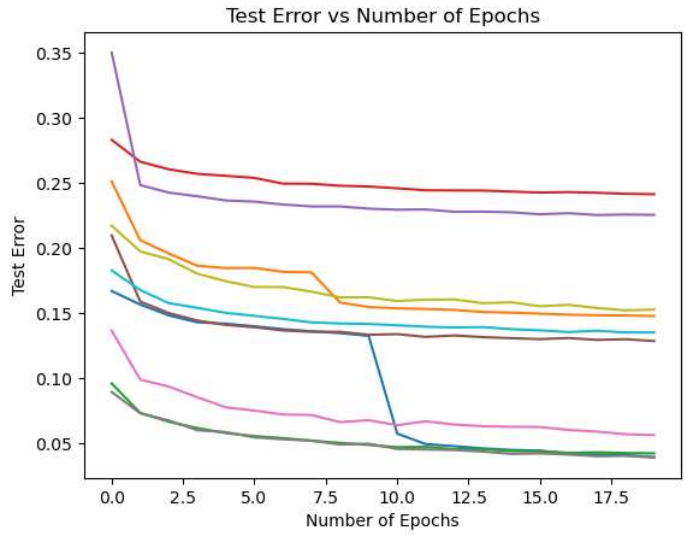
```
Epoch 1/20
1875/1875 [==============================] - 1s 580us/step - loss: 0.0318 - accuracy: 0.8276 - val_loss: 0.0204 - val_accuracy: 0.9038
Epoch 2/20
1875/1875 [==============================] - 1s 536us/step - loss: 0.0191 - accuracy: 0.9135 - val_loss: 0.0168 - val_accuracy: 0.9265
Epoch 3/20
1875/1875 [==============================] - 1s 536us/step - loss: 0.0165 - accuracy: 0.9267 - val_loss: 0.0152 - val_accuracy: 0.9332
Epoch 4/20
1875/1875 [==============================] - 1s 537us/step - loss: 0.0150 - accuracy: 0.9347 - val_loss: 0.0143 - val_accuracy: 0.9381
Epoch 5/20
1875/1875 [==============================] - 1s 536us/step - loss: 0.0139 - accuracy: 0.9394 - val_loss: 0.0133 - val_accuracy: 0.9419
Epoch 6/20
1875/1875 [==============================] - 1s 533us/step - loss: 0.0131 - accuracy: 0.9434 - val_loss: 0.0128 - val_accuracy: 0.9443
Epoch 7/20
1875/1875 [==============================] - 1s 534us/step - loss: 0.0124 - accuracy: 0.9468 - val_loss: 0.0123 - val_accuracy: 0.9458
Epoch 8/20
1875/1875 [==============================] - 1s 558us/step - loss: 0.0119 - accuracy: 0.9492 - val_loss: 0.0119 - val_accuracy: 0.9478
Epoch 9/20
1875/1875 [==============================] - 1s 550us/step - loss: 0.0115 - accuracy: 0.9511 - val_loss: 0.0115 - val_accuracy: 0.9495
Epoch 10/20
1875/1875 [==============================] - 1s 535us/step - loss: 0.0111 - accuracy: 0.9527 - val_loss: 0.0113 - val_accuracy: 0.9510
Epoch 11/20
1875/1875 [==============================] - 1s 540us/step - loss: 0.0108 - accuracy: 0.9548 - val_loss: 0.0109 - val_accuracy: 0.9528
Epoch 12/20
1875/1875 [==============================] - 1s 534us/step - loss: 0.0105 - accuracy: 0.9559 - val_loss: 0.0107 - val_accuracy: 0.9525
Epoch 13/20
1875/1875 [==============================] - 1s 536us/step - loss: 0.0102 - accuracy: 0.9570 - val_loss: 0.0104 - val_accuracy: 0.9543
Epoch 14/20
1875/1875 [==============================] - 1s 535us/step - loss: 0.0100 - accuracy: 0.9578 - val_loss: 0.0104 - val_accuracy: 0.9542
Epoch 15/20
1875/1875 [==============================] - 1s 536us/step - loss: 0.0098 - accuracy: 0.9591 - val_loss: 0.0102 - val_accuracy: 0.9557
Epoch 16/20
1875/1875 [==============================] - 1s 534us/step - loss: 0.0096 - accuracy: 0.9602 - val_loss: 0.0100 - val_accuracy: 0.9559
Epoch 17/20
1875/1875 [==============================] - 1s 536us/step - loss: 0.0094 - accuracy: 0.9607 - val_loss: 0.0099 - val_accuracy: 0.9571
Epoch 18/20
1875/1875 [==============================] - 1s 536us/step - loss: 0.0092 - accuracy: 0.9618 - val_loss: 0.0098 - val_accuracy: 0.9567
Epoch 19/20
1875/1875 [==============================] - 1s 545us/step - loss: 0.0091 - accuracy: 0.9626 - val_loss: 0.0096 - val_accuracy: 0.9572
Epoch 20/20
1875/1875 [==============================] - 1s 534us/step - loss: 0.0090 - accuracy: 0.9630 - val_loss: 0.0095 - val_accuracy: 0.9575
WARNING:absl:At this time, the v2.11+ optimizer `tf.keras.optimizers.SGD` runs slowly on M1/M2 Macs, please use the legacy Keras optimizer instead, loc
ated at `tf.keras.optimizers.legacy.SGD`.
Epoch 1/20
1875/1875 [==============================] - 1s 574us/step - loss: 0.0503 - accuracy: 0.6256 - val_loss: 0.0383 - val_accuracy: 0.7169
Epoch 2/20
1875/1875 [==============================] - 1s 532us/step - loss: 0.0378 - accuracy: 0.7179 - val_loss: 0.0356 - val_accuracy: 0.7337
Epoch 3/20
1875/1875 [==============================] - 1s 544us/step - loss: 0.0355 - accuracy: 0.7305 - val_loss: 0.0342 - val_accuracy: 0.7395
Epoch 4/20
1875/1875 [==============================] - 1s 555us/step - loss: 0.0341 - accuracy: 0.7366 - val_loss: 0.0331 - val_accuracy: 0.7430
Epoch 5/20
1875/1875 [==============================] - 1s 533us/step - loss: 0.0331 - accuracy: 0.7413 - val_loss: 0.0325 - val_accuracy: 0.7445
Epoch 6/20
1875/1875 [==============================] - 1s 531us/step - loss: 0.0324 - accuracy: 0.7446 - val_loss: 0.0320 - val_accuracy: 0.7461
Epoch 7/20
1875/1875 [==============================] - 1s 531us/step - loss: 0.0319 - accuracy: 0.7469 - val_loss: 0.0315 - val_accuracy: 0.7505
Epoch 8/20
1875/1875 [==============================] - 1s 531us/step - loss: 0.0314 - accuracy: 0.7493 - val_loss: 0.0313 - val_accuracy: 0.7506
Epoch 9/20
1875/1875 [==============================] - 1s 532us/step - loss: 0.0310 - accuracy: 0.7508 - val_loss: 0.0309 - val_accuracy: 0.7521
Epoch 10/20
1875/1875 [==============================] - 1s 532us/step - loss: 0.0306 - accuracy: 0.7523 - val_loss: 0.0307 - val_accuracy: 0.7527
Epoch 11/20
1875/1875 [==============================] - 1s 532us/step - loss: 0.0303 - accuracy: 0.7547 - val_loss: 0.0303 - val_accuracy: 0.7541
Epoch 12/20
1875/1875 [==============================] - 1s 530us/step - loss: 0.0300 - accuracy: 0.7558 - val_loss: 0.0301 - val_accuracy: 0.7556
Epoch 13/20
1875/1875 [==============================] - 1s 530us/step - loss: 0.0298 - accuracy: 0.7569 - val_loss: 0.0300 - val_accuracy: 0.7557
Epoch 14/20
1875/1875 [==============================] - 1s 531us/step - loss: 0.0296 - accuracy: 0.7577 - val_loss: 0.0298 - val_accuracy: 0.7558
Epoch 15/20
1875/1875 [==============================] - 1s 531us/step - loss: 0.0294 - accuracy: 0.7588 - val_loss: 0.0296 - val_accuracy: 0.7566
Epoch 16/20
1875/1875 [==============================] - 1s 530us/step - loss: 0.0292 - accuracy: 0.7596 - val_loss: 0.0295 - val_accuracy: 0.7573
Epoch 17/20
1875/1875 [==============================] - 1s 530us/step - loss: 0.0291 - accuracy: 0.7605 - val_loss: 0.0294 - val_accuracy: 0.7570
Epoch 18/20
1875/1875 [==============================] - 1s 534us/step - loss: 0.0289 - accuracy: 0.7609 - val_loss: 0.0293 - val_accuracy: 0.7575
Epoch 19/20
1875/1875 [==============================] - 1s 546us/step - loss: 0.0288 - accuracy: 0.7619 - val_loss: 0.0292 - val_accuracy: 0.7583
Epoch 20/20
1875/1875 [==============================] - 1s 548us/step - loss: 0.0286 - accuracy: 0.7622 - val_loss: 0.0291 - val_accuracy: 0.7586
WARNING:absl:At this time, the v2.11+ optimizer `tf.keras.optimizers.SGD` runs slowly on M1/M2 Macs, please use the legacy Keras optimizer instead, loc
ated at `tf.keras.optimizers.legacy.SGD`.
```

```
Epoch 1/20
1875/1875 [==============================] - 1s 574us/step - loss: 0.0517 - accuracy: 0.5942 - val_loss: 0.0434 - val_accuracy: 0.6499
Epoch 2/20
1875/1875 [==============================] - 1s 536us/step - loss: 0.0373 - accuracy: 0.7141 - val_loss: 0.0328 - val_accuracy: 0.7516
Epoch 3/20
1875/1875 [==============================] - 1s 534us/step - loss: 0.0320 - accuracy: 0.7556 - val_loss: 0.0312 - val_accuracy: 0.7574
Epoch 4/20
1875/1875 [==============================] - 1s 535us/step - loss: 0.0306 - accuracy: 0.7614 - val_loss: 0.0303 - val_accuracy: 0.7602
Epoch 5/20
1875/1875 [==============================] - 1s 535us/step - loss: 0.0297 - accuracy: 0.7655 - val_loss: 0.0295 - val_accuracy: 0.7634
Epoch 6/20
1875/1875 [==============================] - 1s 535us/step - loss: 0.0290 - accuracy: 0.7686 - val_loss: 0.0291 - val_accuracy: 0.7643
Epoch 7/20
1875/1875 [==============================] - 1s 634us/step - loss: 0.0285 - accuracy: 0.7704 - val_loss: 0.0288 - val_accuracy: 0.7665
Epoch 8/20
1875/1875 [==============================] - 1s 536us/step - loss: 0.0281 - accuracy: 0.7722 - val_loss: 0.0284 - val_accuracy: 0.7680
Epoch 9/20
1875/1875 [==============================] - 1s 535us/step - loss: 0.0278 - accuracy: 0.7736 - val_loss: 0.0283 - val_accuracy: 0.7680
Epoch 10/20
1875/1875 [==============================] - 1s 564us/step - loss: 0.0275 - accuracy: 0.7748 - val_loss: 0.0280 - val_accuracy: 0.7697
Epoch 11/20
1875/1875 [==============================] - 1s 535us/step - loss: 0.0273 - accuracy: 0.7760 - val_loss: 0.0278 - val_accuracy: 0.7705
Epoch 12/20
1875/1875 [==============================] - 1s 535us/step - loss: 0.0271 - accuracy: 0.7772 - val_loss: 0.0276 - val_accuracy: 0.7703
Epoch 13/20
1875/1875 [==============================] - 1s 543us/step - loss: 0.0269 - accuracy: 0.7781 - val_loss: 0.0275 - val_accuracy: 0.7721
Epoch 14/20
1875/1875 [==============================] - 1s 587us/step - loss: 0.0267 - accuracy: 0.7786 - val_loss: 0.0273 - val_accuracy: 0.7720
Epoch 15/20
1875/1875 [==============================] - 1s 557us/step - loss: 0.0265 - accuracy: 0.7799 - val_loss: 0.0272 - val_accuracy: 0.7725
Epoch 16/20
1875/1875 [==============================] - 1s 535us/step - loss: 0.0264 - accuracy: 0.7802 - val_loss: 0.0271 - val_accuracy: 0.7740
Epoch 17/20
1875/1875 [==============================] - 1s 536us/step - loss: 0.0263 - accuracy: 0.7809 - val_loss: 0.0271 - val_accuracy: 0.7732
Epoch 18/20
1875/1875 [==============================] - 1s 535us/step - loss: 0.0262 - accuracy: 0.7812 - val_loss: 0.0269 - val_accuracy: 0.7746
Epoch 19/20
1875/1875 [==============================] - 1s 535us/step - loss: 0.0260 - accuracy: 0.7818 - val_loss: 0.0268 - val_accuracy: 0.7742
Epoch 20/20
1875/1875 [==============================] - 1s 536us/step - loss: 0.0259 - accuracy: 0.7818 - val_loss: 0.0268 - val_accuracy: 0.7745
WARNING:absl:At this time, the v2.11+ optimizer `tf.keras.optimizers.SGD` runs slowly on M1/M2 Macs, please use the legacy Keras optimizer instead, loc
ated at `tf.keras.optimizers.legacy.SGD`.
Epoch 1/20
1875/1875 [==============================] - 1s 577us/step - loss: 0.0480 - accuracy: 0.6520 - val_loss: 0.0332 - val_accuracy: 0.7903
Epoch 2/20
1875/1875 [==============================] - 1s 531us/step - loss: 0.0275 - accuracy: 0.8263 - val_loss: 0.0248 - val_accuracy: 0.8410
Epoch 3/20
1875/1875 [==============================] - 1s 531us/step - loss: 0.0242 - accuracy: 0.8434 - val_loss: 0.0230 - val_accuracy: 0.8498
Epoch 4/20
1875/1875 [==============================] - 1s 531us/step - loss: 0.0227 - accuracy: 0.8507 - val_loss: 0.0220 - val_accuracy: 0.8556
Epoch 5/20
1875/1875 [==============================] - 1s 530us/step - loss: 0.0217 - accuracy: 0.8548 - val_loss: 0.0213 - val_accuracy: 0.8589
Epoch 6/20
1875/1875 [==============================] - 1s 530us/step - loss: 0.0210 - accuracy: 0.8584 - val_loss: 0.0207 - val_accuracy: 0.8607
Epoch 7/20
1875/1875 [==============================] - 1s 536us/step - loss: 0.0205 - accuracy: 0.8605 - val_loss: 0.0203 - val_accuracy: 0.8632
Epoch 8/20
1875/1875 [==============================] - 1s 530us/step - loss: 0.0200 - accuracy: 0.8628 - val_loss: 0.0200 - val_accuracy: 0.8642
Epoch 9/20
1875/1875 [==============================] - 1s 542us/step - loss: 0.0197 - accuracy: 0.8642 - val_loss: 0.0198 - val_accuracy: 0.8642
Epoch 10/20
1875/1875 [==============================] - 1s 555us/step - loss: 0.0194 - accuracy: 0.8657 - val_loss: 0.0195 - val_accuracy: 0.8665
Epoch 11/20
1875/1875 [==============================] - 1s 532us/step - loss: 0.0191 - accuracy: 0.8671 - val_loss: 0.0194 - val_accuracy: 0.8660
Epoch 12/20
1875/1875 [==============================] - 1s 531us/step - loss: 0.0189 - accuracy: 0.8682 - val_loss: 0.0191 - val_accuracy: 0.8681
Epoch 13/20
1875/1875 [==============================] - 1s 530us/step - loss: 0.0186 - accuracy: 0.8687 - val_loss: 0.0190 - val_accuracy: 0.8670
Epoch 14/20
1875/1875 [==============================] - 1s 531us/step - loss: 0.0184 - accuracy: 0.8696 - val_loss: 0.0189 - val_accuracy: 0.8683
Epoch 15/20
1875/1875 [==============================] - 1s 531us/step - loss: 0.0183 - accuracy: 0.8706 - val_loss: 0.0188 - val_accuracy: 0.8691
Epoch 16/20
1875/1875 [==============================] - 1s 540us/step - loss: 0.0181 - accuracy: 0.8712 - val_loss: 0.0186 - val_accuracy: 0.8698
Epoch 17/20
1875/1875 [==============================] - 1s 531us/step - loss: 0.0180 - accuracy: 0.8718 - val_loss: 0.0185 - val_accuracy: 0.8690
Epoch 18/20
1875/1875 [==============================] - 1s 536us/step - loss: 0.0178 - accuracy: 0.8723 - val_loss: 0.0184 - val_accuracy: 0.8704
Epoch 19/20
1875/1875 [==============================] - 1s 536us/step - loss: 0.0177 - accuracy: 0.8734 - val_loss: 0.0183 - val_accuracy: 0.8699
Epoch 20/20
1875/1875 [==============================] - 1s 531us/step - loss: 0.0176 - accuracy: 0.8737 - val_loss: 0.0182 - val_accuracy: 0.8713
WARNING:absl:At this time, the v2.11+ optimizer `tf.keras.optimizers.SGD` runs slowly on M1/M2 Macs, please use the legacy Keras optimizer instead, loc
ated at `tf.keras.optimizers.legacy.SGD`.
```

```
Epoch 1/20
1875/1875 [==============================] - 1s 584us/step - loss: 0.0407 - accuracy: 0.7509 - val_loss: 0.0285 - val_accuracy: 0.8633
Epoch 2/20
1875/1875 [==============================] - 1s 538us/step - loss: 0.0270 - accuracy: 0.8789 - val_loss: 0.0249 - val_accuracy: 0.9009
Epoch 3/20
1875/1875 [==============================] - 1s 536us/step - loss: 0.0244 - accuracy: 0.9015 - val_loss: 0.0232 - val_accuracy: 0.9062
Epoch 4/20
1875/1875 [==============================] - 1s 536us/step - loss: 0.0230 - accuracy: 0.9119 - val_loss: 0.0223 - val_accuracy: 0.9145
Epoch 5/20
1875/1875 [==============================] - 1s 558us/step - loss: 0.0221 - accuracy: 0.9183 - val_loss: 0.0215 - val_accuracy: 0.9222
Epoch 6/20
1875/1875 [==============================] - 1s 541us/step - loss: 0.0214 - accuracy: 0.9234 - val_loss: 0.0210 - val_accuracy: 0.9247
Epoch 7/20
1875/1875 [==============================] - 1s 536us/step - loss: 0.0209 - accuracy: 0.9266 - val_loss: 0.0207 - val_accuracy: 0.9277
Epoch 8/20
1875/1875 [==============================] - 1s 535us/step - loss: 0.0204 - accuracy: 0.9294 - val_loss: 0.0203 - val_accuracy: 0.9281
Epoch 9/20
1875/1875 [==============================] - 1s 534us/step - loss: 0.0201 - accuracy: 0.9316 - val_loss: 0.0202 - val_accuracy: 0.9335
Epoch 10/20
1875/1875 [==============================] - 1s 535us/step - loss: 0.0198 - accuracy: 0.9338 - val_loss: 0.0198 - val_accuracy: 0.9321
Epoch 11/20
1875/1875 [==============================] - 1s 539us/step - loss: 0.0195 - accuracy: 0.9354 - val_loss: 0.0195 - val_accuracy: 0.9359
Epoch 12/20
1875/1875 [==============================] - 1s 535us/step - loss: 0.0193 - accuracy: 0.9366 - val_loss: 0.0195 - val_accuracy: 0.9329
Epoch 13/20
1875/1875 [==============================] - 1s 535us/step - loss: 0.0191 - accuracy: 0.9384 - val_loss: 0.0192 - val_accuracy: 0.9354
Epoch 14/20
1875/1875 [==============================] - 1s 534us/step - loss: 0.0188 - accuracy: 0.9395 - val_loss: 0.0192 - val_accuracy: 0.9367
Epoch 15/20
1875/1875 [==============================] - 1s 534us/step - loss: 0.0187 - accuracy: 0.9409 - val_loss: 0.0190 - val_accuracy: 0.9371
Epoch 16/20
1875/1875 [==============================] - 1s 535us/step - loss: 0.0185 - accuracy: 0.9416 - val_loss: 0.0189 - val_accuracy: 0.9373
Epoch 17/20
1875/1875 [==============================] - 1s 536us/step - loss: 0.0184 - accuracy: 0.9432 - val_loss: 0.0188 - val_accuracy: 0.9395
Epoch 18/20
1875/1875 [==============================] - 1s 538us/step - loss: 0.0182 - accuracy: 0.9435 - val_loss: 0.0187 - val_accuracy: 0.9408
Epoch 19/20
1875/1875 [==============================] - 1s 536us/step - loss: 0.0181 - accuracy: 0.9455 - val_loss: 0.0186 - val_accuracy: 0.9428
Epoch 20/20
1875/1875 [==============================] - 1s 542us/step - loss: 0.0180 - accuracy: 0.9457 - val_loss: 0.0186 - val_accuracy: 0.9435
WARNING:absl:At this time, the v2.11+ optimizer `tf.keras.optimizers.SGD` runs slowly on M1/M2 Macs, please use the legacy Keras optimizer instead, loc
ated at `tf.keras.optimizers.legacy.SGD`.
Epoch 1/20
1875/1875 [==============================] - 1s 588us/step - loss: 0.0329 - accuracy: 0.8098 - val_loss: 0.0197 - val_accuracy: 0.9104
Epoch 2/20
1875/1875 [==============================] - 1s 532us/step - loss: 0.0187 - accuracy: 0.9161 - val_loss: 0.0166 - val_accuracy: 0.9267
Epoch 3/20
1875/1875 [==============================] - 1s 532us/step - loss: 0.0160 - accuracy: 0.9300 - val_loss: 0.0149 - val_accuracy: 0.9323
Epoch 4/20
1875/1875 [==============================] - 1s 531us/step - loss: 0.0145 - accuracy: 0.9371 - val_loss: 0.0138 - val_accuracy: 0.9397
Epoch 5/20
1875/1875 [==============================] - 1s 592us/step - loss: 0.0135 - accuracy: 0.9421 - val_loss: 0.0131 - val_accuracy: 0.9412
Epoch 6/20
1875/1875 [==============================] - 1s 531us/step - loss: 0.0127 - accuracy: 0.9459 - val_loss: 0.0125 - val_accuracy: 0.9452
Epoch 7/20
1875/1875 [==============================] - 1s 549us/step - loss: 0.0121 - accuracy: 0.9486 - val_loss: 0.0120 - val_accuracy: 0.9467
Epoch 8/20
1875/1875 [==============================] - 1s 535us/step - loss: 0.0116 - accuracy: 0.9509 - val_loss: 0.0117 - val_accuracy: 0.9477
Epoch 9/20
1875/1875 [==============================] - 1s 532us/step - loss: 0.0111 - accuracy: 0.9532 - val_loss: 0.0114 - val_accuracy: 0.9507
Epoch 10/20
1875/1875 [==============================] - 1s 531us/step - loss: 0.0108 - accuracy: 0.9542 - val_loss: 0.0112 - val_accuracy: 0.9501
Epoch 11/20
1875/1875 [==============================] - 1s 532us/step - loss: 0.0105 - accuracy: 0.9564 - val_loss: 0.0109 - val_accuracy: 0.9541
Epoch 12/20
1875/1875 [==============================] - 1s 532us/step - loss: 0.0102 - accuracy: 0.9572 - val_loss: 0.0107 - val_accuracy: 0.9545
Epoch 13/20
1875/1875 [==============================] - 1s 532us/step - loss: 0.0099 - accuracy: 0.9588 - val_loss: 0.0104 - val_accuracy: 0.9549
Epoch 14/20
1875/1875 [==============================] - 1s 531us/step - loss: 0.0097 - accuracy: 0.9597 - val_loss: 0.0103 - val_accuracy: 0.9561
Epoch 15/20
1875/1875 [==============================] - 1s 535us/step - loss: 0.0095 - accuracy: 0.9605 - val_loss: 0.0100 - val_accuracy: 0.9579
Epoch 16/20
1875/1875 [==============================] - 1s 565us/step - loss: 0.0093 - accuracy: 0.9617 - val_loss: 0.0100 - val_accuracy: 0.9575
Epoch 17/20
1875/1875 [==============================] - 1s 533us/step - loss: 0.0092 - accuracy: 0.9623 - val_loss: 0.0099 - val_accuracy: 0.9584
Epoch 18/20
1875/1875 [==============================] - 1s 532us/step - loss: 0.0090 - accuracy: 0.9628 - val_loss: 0.0097 - val_accuracy: 0.9597
Epoch 19/20
1875/1875 [==============================] - 1s 536us/step - loss: 0.0089 - accuracy: 0.9633 - val_loss: 0.0096 - val_accuracy: 0.9594
Epoch 20/20
1875/1875 [==============================] - 1s 531us/step - loss: 0.0087 - accuracy: 0.9645 - val_loss: 0.0095 - val_accuracy: 0.9607
WARNING:absl:At this time, the v2.11+ optimizer `tf.keras.optimizers.SGD` runs slowly on M1/M2 Macs, please use the legacy Keras optimizer instead, loc
ated at `tf.keras.optimizers.legacy.SGD`.
```

```
Epoch 1/20
1875/1875 [==============================] - 1s 573us/step - loss: 0.0466 - accuracy: 0.6934 - val_loss: 0.0373 - val_accuracy: 0.7828
Epoch 2/20
1875/1875 [==============================] - 1s 531us/step - loss: 0.0366 - accuracy: 0.7856 - val_loss: 0.0343 - val_accuracy: 0.8025
Epoch 3/20
1875/1875 [==============================] - 1s 547us/step - loss: 0.0342 - accuracy: 0.8022 - val_loss: 0.0328 - val_accuracy: 0.8085
Epoch 4/20
1875/1875 [==============================] - 1s 531us/step - loss: 0.0328 - accuracy: 0.8122 - val_loss: 0.0319 - val_accuracy: 0.8196
Epoch 5/20
1875/1875 [==============================] - 1s 530us/step - loss: 0.0318 - accuracy: 0.8195 - val_loss: 0.0311 - val_accuracy: 0.8254
Epoch 6/20
1875/1875 [==============================] - 1s 530us/step - loss: 0.0311 - accuracy: 0.8243 - val_loss: 0.0306 - val_accuracy: 0.8298
Epoch 7/20
1875/1875 [==============================] - 1s 530us/step - loss: 0.0306 - accuracy: 0.8277 - val_loss: 0.0302 - val_accuracy: 0.8299
Epoch 8/20
1875/1875 [==============================] - 1s 531us/step - loss: 0.0301 - accuracy: 0.8310 - val_loss: 0.0299 - val_accuracy: 0.8334
Epoch 9/20
1875/1875 [==============================] - 1s 530us/step - loss: 0.0298 - accuracy: 0.8328 - val_loss: 0.0295 - val_accuracy: 0.8378
Epoch 10/20
1875/1875 [==============================] - 1s 530us/step - loss: 0.0294 - accuracy: 0.8350 - val_loss: 0.0293 - val_accuracy: 0.8377
Epoch 11/20
1875/1875 [==============================] - 1s 551us/step - loss: 0.0292 - accuracy: 0.8368 - val_loss: 0.0292 - val_accuracy: 0.8406
Epoch 12/20
1875/1875 [==============================] - 1s 544us/step - loss: 0.0289 - accuracy: 0.8387 - val_loss: 0.0289 - val_accuracy: 0.8395
Epoch 13/20
1875/1875 [==============================] - 1s 538us/step - loss: 0.0287 - accuracy: 0.8397 - val_loss: 0.0287 - val_accuracy: 0.8394
Epoch 14/20
1875/1875 [==============================] - 1s 539us/step - loss: 0.0285 - accuracy: 0.8411 - val_loss: 0.0286 - val_accuracy: 0.8422
Epoch 15/20
1875/1875 [==============================] - 1s 530us/step - loss: 0.0284 - accuracy: 0.8426 - val_loss: 0.0284 - val_accuracy: 0.8416
Epoch 16/20
1875/1875 [==============================] - 1s 530us/step - loss: 0.0282 - accuracy: 0.8432 - val_loss: 0.0283 - val_accuracy: 0.8446
Epoch 17/20
1875/1875 [==============================] - 1s 533us/step - loss: 0.0281 - accuracy: 0.8444 - val_loss: 0.0282 - val_accuracy: 0.8435
Epoch 18/20
1875/1875 [==============================] - 1s 531us/step - loss: 0.0279 - accuracy: 0.8450 - val_loss: 0.0282 - val_accuracy: 0.8460
Epoch 19/20
1875/1875 [==============================] - 1s 531us/step - loss: 0.0278 - accuracy: 0.8462 - val_loss: 0.0280 - val_accuracy: 0.8477
Epoch 20/20
1875/1875 [==============================] - 1s 532us/step - loss: 0.0277 - accuracy: 0.8470 - val_loss: 0.0279 - val_accuracy: 0.8472
WARNING:absl:At this time, the v2.11+ optimizer `tf.keras.optimizers.SGD` runs slowly on M1/M2 Macs, please use the legacy Keras optimizer instead, loc
ated at `tf.keras.optimizers.legacy.SGD`.
Epoch 1/20
1875/1875 [==============================] - 1s 569us/step - loss: 0.0396 - accuracy: 0.7338 - val_loss: 0.0289 - val_accuracy: 0.8171
Epoch 2/20
1875/1875 [==============================] - 1s 534us/step - loss: 0.0273 - accuracy: 0.8261 - val_loss: 0.0255 - val_accuracy: 0.8322
Epoch 3/20
1875/1875 [==============================] - 1s 533us/step - loss: 0.0247 - accuracy: 0.8386 - val_loss: 0.0239 - val_accuracy: 0.8422
Epoch 4/20
1875/1875 [==============================] - 1s 533us/step - loss: 0.0233 - accuracy: 0.8447 - val_loss: 0.0228 - val_accuracy: 0.8458
Epoch 5/20
1875/1875 [==============================] - 1s 539us/step - loss: 0.0223 - accuracy: 0.8497 - val_loss: 0.0223 - val_accuracy: 0.8497
Epoch 6/20
1875/1875 [==============================] - 1s 540us/step - loss: 0.0216 - accuracy: 0.8530 - val_loss: 0.0216 - val_accuracy: 0.8520
Epoch 7/20
1875/1875 [==============================] - 1s 558us/step - loss: 0.0210 - accuracy: 0.8562 - val_loss: 0.0211 - val_accuracy: 0.8543
Epoch 8/20
1875/1875 [==============================] - 1s 533us/step - loss: 0.0205 - accuracy: 0.8574 - val_loss: 0.0209 - val_accuracy: 0.8570
Epoch 9/20
1875/1875 [==============================] - 1s 533us/step - loss: 0.0201 - accuracy: 0.8602 - val_loss: 0.0206 - val_accuracy: 0.8578
Epoch 10/20
1875/1875 [==============================] - 1s 533us/step - loss: 0.0198 - accuracy: 0.8615 - val_loss: 0.0203 - val_accuracy: 0.8582
Epoch 11/20
1875/1875 [==============================] - 1s 533us/step - loss: 0.0195 - accuracy: 0.8632 - val_loss: 0.0202 - val_accuracy: 0.8592
Epoch 12/20
1875/1875 [==============================] - 1s 532us/step - loss: 0.0192 - accuracy: 0.8648 - val_loss: 0.0199 - val_accuracy: 0.8603
Epoch 13/20
1875/1875 [==============================] - 1s 533us/step - loss: 0.0190 - accuracy: 0.8656 - val_loss: 0.0197 - val_accuracy: 0.8609
Epoch 14/20
1875/1875 [==============================] - 1s 533us/step - loss: 0.0188 - accuracy: 0.8672 - val_loss: 0.0196 - val_accuracy: 0.8606
Epoch 15/20
1875/1875 [==============================] - 1s 533us/step - loss: 0.0187 - accuracy: 0.8678 - val_loss: 0.0194 - val_accuracy: 0.8622
Epoch 16/20
1875/1875 [==============================] - 1s 533us/step - loss: 0.0185 - accuracy: 0.8687 - val_loss: 0.0193 - val_accuracy: 0.8631
Epoch 17/20
1875/1875 [==============================] - 1s 538us/step - loss: 0.0183 - accuracy: 0.8695 - val_loss: 0.0192 - val_accuracy: 0.8644
Epoch 18/20
1875/1875 [==============================] - 1s 532us/step - loss: 0.0182 - accuracy: 0.8702 - val_loss: 0.0191 - val_accuracy: 0.8634
Epoch 19/20
1875/1875 [==============================] - 1s 532us/step - loss: 0.0180 - accuracy: 0.8708 - val_loss: 0.0191 - val_accuracy: 0.8647
Epoch 20/20
1875/1875 [==============================] - 1s 531us/step - loss: 0.0179 - accuracy: 0.8713 - val_loss: 0.0189 - val_accuracy: 0.8647
```





The testing error and training errors seem fairly similar when comparing the two graphs. Furthermore, while each model seems to follow the same shape in how the error changes over the 20 epochs, only some of the models actually end up learning to solve the problem well. That is, each model converges to a different final training and test error. This can again be associated with the random weights that each model begins with, which leads to each model converging to a different local minima.

3. [1 mark]: Repeat question 2 (generating the same 2 plots), but with a learning rate of 0.01. How does this affect the learning performance?

In [ ]:
```
import numpy as np

train_errs = []
test_errs = []

def get_error(accuracy):
    return 1 - accuracy

err_func = np.vectorize(get_error)

num_models = 10

for i in range(num_models):
    # Create new model
    model = tf.keras.models.Sequential([
      tf.keras.layers.Flatten(input_shape=(28, 28)),   # input is a 28x28 image
      tf.keras.layers.Dense(32, activation='relu'),    # 32 neurons in the middle "hidden" layer
      tf.keras.layers.Dense(10, activation='relu')     # 10 outputs (one for each category)
    ])

    # Compile model
    model.compile(optimizer=tf.keras.optimizers.SGD(learning_rate=0.01), # use stochastic gradient descent
                  loss=my_loss,
                  metrics=['accuracy']  # in addition to the loss, also compute the categorization accuracy
                 )

    # Train each model for 20 epochs
    history = model.fit(x_train, y_train_target, epochs=20, validation_data=(x_test, y_test_target));

    # Store training and test errors
    train_err = err_func(history.history['accuracy'])
    train_errs.append(train_err)
```

```
    test_err = err_func(history.history['val_accuracy'])
    test_errs.append(test_err)

# Plot training errors
plt.figure()
for i in range(num_models):
    plt.plot(train_errs[i])
plt.title("Training Error vs Number of Epochs")
plt.ylabel("Training Error")
plt.xlabel("Number of Epochs")
plt.show()

# Plot test errors
plt.figure()
for i in range(num_models):
    plt.plot(test_errs[i])
plt.title("Test Error vs Number of Epochs")
plt.ylabel("Test Error")
plt.xlabel("Number of Epochs")
plt.show()
```

```
WARNING:absl:At this time, the v2.11+ optimizer `tf.keras.optimizers.SGD` runs slowly on M1/M2 Macs, please use the legacy Keras optimizer instead, loc
ated at `tf.keras.optimizers.legacy.SGD`.
Epoch 1/20
1875/1875 [==============================] - 2s 737us/step - loss: 0.0746 - accuracy: 0.4463 - val_loss: 0.0551 - val_accuracy: 0.6352
Epoch 2/20
1875/1875 [==============================] - 1s 541us/step - loss: 0.0461 - accuracy: 0.7174 - val_loss: 0.0383 - val_accuracy: 0.7803
Epoch 3/20
1875/1875 [==============================] - 1s 549us/step - loss: 0.0366 - accuracy: 0.7879 - val_loss: 0.0333 - val_accuracy: 0.8054
Epoch 4/20
1875/1875 [==============================] - 1s 539us/step - loss: 0.0328 - accuracy: 0.8120 - val_loss: 0.0307 - val_accuracy: 0.8249
Epoch 5/20
1875/1875 [==============================] - 1s 538us/step - loss: 0.0305 - accuracy: 0.8279 - val_loss: 0.0289 - val_accuracy: 0.8382
Epoch 6/20
1875/1875 [==============================] - 1s 537us/step - loss: 0.0288 - accuracy: 0.8395 - val_loss: 0.0274 - val_accuracy: 0.8490
Epoch 7/20
1875/1875 [==============================] - 1s 570us/step - loss: 0.0274 - accuracy: 0.8515 - val_loss: 0.0259 - val_accuracy: 0.8620
Epoch 8/20
1875/1875 [==============================] - 1s 682us/step - loss: 0.0257 - accuracy: 0.8676 - val_loss: 0.0241 - val_accuracy: 0.8795
Epoch 9/20
1875/1875 [==============================] - 1s 538us/step - loss: 0.0241 - accuracy: 0.8820 - val_loss: 0.0228 - val_accuracy: 0.8882
Epoch 10/20
1875/1875 [==============================] - 1s 537us/step - loss: 0.0230 - accuracy: 0.8892 - val_loss: 0.0218 - val_accuracy: 0.8942
Epoch 11/20
1875/1875 [==============================] - 1s 552us/step - loss: 0.0222 - accuracy: 0.8950 - val_loss: 0.0211 - val_accuracy: 0.8977
Epoch 12/20
1875/1875 [==============================] - 1s 535us/step - loss: 0.0215 - accuracy: 0.8985 - val_loss: 0.0204 - val_accuracy: 0.9043
Epoch 13/20
1875/1875 [==============================] - 1s 536us/step - loss: 0.0209 - accuracy: 0.9025 - val_loss: 0.0199 - val_accuracy: 0.9079
Epoch 14/20
1875/1875 [==============================] - 1s 544us/step - loss: 0.0203 - accuracy: 0.9056 - val_loss: 0.0194 - val_accuracy: 0.9114
Epoch 15/20
1875/1875 [==============================] - 1s 537us/step - loss: 0.0198 - accuracy: 0.9090 - val_loss: 0.0189 - val_accuracy: 0.9142
Epoch 16/20
1875/1875 [==============================] - 1s 535us/step - loss: 0.0194 - accuracy: 0.9120 - val_loss: 0.0185 - val_accuracy: 0.9162
Epoch 17/20
1875/1875 [==============================] - 1s 537us/step - loss: 0.0190 - accuracy: 0.9141 - val_loss: 0.0182 - val_accuracy: 0.9180
Epoch 18/20
1875/1875 [==============================] - 1s 536us/step - loss: 0.0186 - accuracy: 0.9162 - val_loss: 0.0179 - val_accuracy: 0.9186
Epoch 19/20
1875/1875 [==============================] - 1s 536us/step - loss: 0.0183 - accuracy: 0.9181 - val_loss: 0.0176 - val_accuracy: 0.9211
Epoch 20/20
1875/1875 [==============================] - 1s 534us/step - loss: 0.0180 - accuracy: 0.9193 - val_loss: 0.0173 - val_accuracy: 0.9224
WARNING:absl:At this time, the v2.11+ optimizer `tf.keras.optimizers.SGD` runs slowly on M1/M2 Macs, please use the legacy Keras optimizer instead, loc
ated at `tf.keras.optimizers.legacy.SGD`.
```

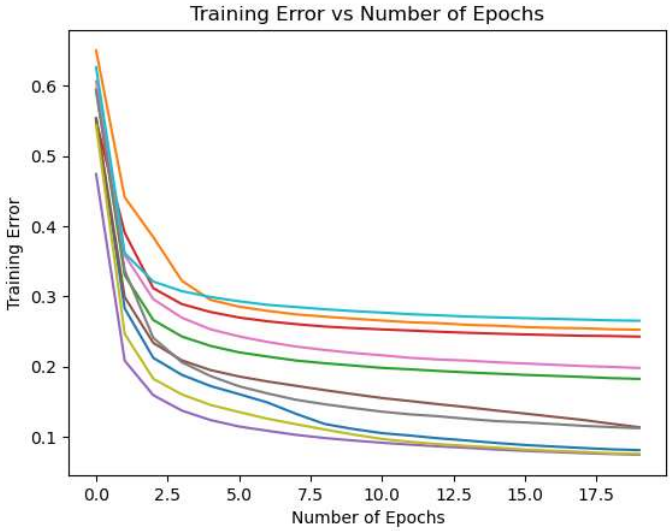
```
Epoch 1/20
1875/1875 [==============================] - 1s 581us/step - loss: 0.0812 - accuracy: 0.3498 - val_loss: 0.0663 - val_accuracy: 0.5131
Epoch 2/20
1875/1875 [==============================] - 1s 552us/step - loss: 0.0606 - accuracy: 0.5591 - val_loss: 0.0551 - val_accuracy: 0.6040
Epoch 3/20
1875/1875 [==============================] - 1s 601us/step - loss: 0.0540 - accuracy: 0.6158 - val_loss: 0.0509 - val_accuracy: 0.6459
Epoch 4/20
1875/1875 [==============================] - 1s 537us/step - loss: 0.0464 - accuracy: 0.6779 - val_loss: 0.0423 - val_accuracy: 0.7052
Epoch 5/20
1875/1875 [==============================] - 1s 544us/step - loss: 0.0421 - accuracy: 0.7052 - val_loss: 0.0400 - val_accuracy: 0.7190
Epoch 6/20
1875/1875 [==============================] - 1s 534us/step - loss: 0.0402 - accuracy: 0.7149 - val_loss: 0.0385 - val_accuracy: 0.7284
Epoch 7/20
1875/1875 [==============================] - 1s 536us/step - loss: 0.0390 - accuracy: 0.7208 - val_loss: 0.0375 - val_accuracy: 0.7327
Epoch 8/20
1875/1875 [==============================] - 1s 540us/step - loss: 0.0380 - accuracy: 0.7259 - val_loss: 0.0367 - val_accuracy: 0.7361
Epoch 9/20
1875/1875 [==============================] - 1s 558us/step - loss: 0.0373 - accuracy: 0.7291 - val_loss: 0.0361 - val_accuracy: 0.7399
Epoch 10/20
1875/1875 [==============================] - 1s 537us/step - loss: 0.0367 - accuracy: 0.7319 - val_loss: 0.0356 - val_accuracy: 0.7443
Epoch 11/20
1875/1875 [==============================] - 1s 537us/step - loss: 0.0362 - accuracy: 0.7346 - val_loss: 0.0351 - val_accuracy: 0.7452
Epoch 12/20
1875/1875 [==============================] - 1s 537us/step - loss: 0.0357 - accuracy: 0.7372 - val_loss: 0.0347 - val_accuracy: 0.7472
Epoch 13/20
1875/1875 [==============================] - 1s 536us/step - loss: 0.0353 - accuracy: 0.7384 - val_loss: 0.0344 - val_accuracy: 0.7493
Epoch 14/20
1875/1875 [==============================] - 1s 564us/step - loss: 0.0349 - accuracy: 0.7408 - val_loss: 0.0341 - val_accuracy: 0.7496
Epoch 15/20
1875/1875 [==============================] - 1s 537us/step - loss: 0.0346 - accuracy: 0.7420 - val_loss: 0.0338 - val_accuracy: 0.7511
Epoch 16/20
1875/1875 [==============================] - 1s 537us/step - loss: 0.0343 - accuracy: 0.7439 - val_loss: 0.0335 - val_accuracy: 0.7526
Epoch 17/20
1875/1875 [==============================] - 1s 543us/step - loss: 0.0341 - accuracy: 0.7451 - val_loss: 0.0333 - val_accuracy: 0.7534
Epoch 18/20
1875/1875 [==============================] - 1s 572us/step - loss: 0.0338 - accuracy: 0.7456 - val_loss: 0.0330 - val_accuracy: 0.7541
Epoch 19/20
1875/1875 [==============================] - 1s 535us/step - loss: 0.0336 - accuracy: 0.7471 - val_loss: 0.0328 - val_accuracy: 0.7554
Epoch 20/20
1875/1875 [==============================] - 1s 535us/step - loss: 0.0333 - accuracy: 0.7477 - val_loss: 0.0326 - val_accuracy: 0.7567
WARNING:absl:At this time, the v2.11+ optimizer `tf.keras.optimizers.SGD` runs slowly on M1/M2 Macs, please use the legacy Keras optimizer instead, loc
ated at `tf.keras.optimizers.legacy.SGD`.
Epoch 1/20
1875/1875 [==============================] - 1s 578us/step - loss: 0.0766 - accuracy: 0.4056 - val_loss: 0.0587 - val_accuracy: 0.5883
Epoch 2/20
1875/1875 [==============================] - 1s 535us/step - loss: 0.0502 - accuracy: 0.6701 - val_loss: 0.0436 - val_accuracy: 0.7219
Epoch 3/20
1875/1875 [==============================] - 1s 534us/step - loss: 0.0417 - accuracy: 0.7339 - val_loss: 0.0388 - val_accuracy: 0.7533
Epoch 4/20
1875/1875 [==============================] - 1s 536us/step - loss: 0.0380 - accuracy: 0.7574 - val_loss: 0.0361 - val_accuracy: 0.7696
Epoch 5/20
1875/1875 [==============================] - 1s 536us/step - loss: 0.0358 - accuracy: 0.7707 - val_loss: 0.0343 - val_accuracy: 0.7792
Epoch 6/20
1875/1875 [==============================] - 1s 535us/step - loss: 0.0343 - accuracy: 0.7800 - val_loss: 0.0331 - val_accuracy: 0.7841
Epoch 7/20
1875/1875 [==============================] - 1s 535us/step - loss: 0.0331 - accuracy: 0.7860 - val_loss: 0.0321 - val_accuracy: 0.7903
Epoch 8/20
1875/1875 [==============================] - 1s 536us/step - loss: 0.0322 - accuracy: 0.7916 - val_loss: 0.0313 - val_accuracy: 0.7940
Epoch 9/20
1875/1875 [==============================] - 1s 534us/step - loss: 0.0315 - accuracy: 0.7955 - val_loss: 0.0307 - val_accuracy: 0.7978
Epoch 10/20
1875/1875 [==============================] - 1s 542us/step - loss: 0.0308 - accuracy: 0.7987 - val_loss: 0.0301 - val_accuracy: 0.8014
Epoch 11/20
1875/1875 [==============================] - 1s 536us/step - loss: 0.0303 - accuracy: 0.8021 - val_loss: 0.0296 - val_accuracy: 0.8051
Epoch 12/20
1875/1875 [==============================] - 1s 548us/step - loss: 0.0298 - accuracy: 0.8042 - val_loss: 0.0292 - val_accuracy: 0.8068
Epoch 13/20
1875/1875 [==============================] - 1s 559us/step - loss: 0.0294 - accuracy: 0.8065 - val_loss: 0.0288 - val_accuracy: 0.8081
Epoch 14/20
1875/1875 [==============================] - 1s 545us/step - loss: 0.0290 - accuracy: 0.8086 - val_loss: 0.0285 - val_accuracy: 0.8094
Epoch 15/20
1875/1875 [==============================] - 1s 535us/step - loss: 0.0286 - accuracy: 0.8103 - val_loss: 0.0281 - val_accuracy: 0.8120
Epoch 16/20
1875/1875 [==============================] - 1s 534us/step - loss: 0.0283 - accuracy: 0.8121 - val_loss: 0.0278 - val_accuracy: 0.8140
Epoch 17/20
1875/1875 [==============================] - 1s 535us/step - loss: 0.0279 - accuracy: 0.8134 - val_loss: 0.0276 - val_accuracy: 0.8148
Epoch 18/20
1875/1875 [==============================] - 1s 546us/step - loss: 0.0277 - accuracy: 0.8150 - val_loss: 0.0273 - val_accuracy: 0.8166
Epoch 19/20
1875/1875 [==============================] - 1s 535us/step - loss: 0.0274 - accuracy: 0.8166 - val_loss: 0.0271 - val_accuracy: 0.8181
Epoch 20/20
1875/1875 [==============================] - 1s 535us/step - loss: 0.0271 - accuracy: 0.8178 - val_loss: 0.0268 - val_accuracy: 0.8188
WARNING:absl:At this time, the v2.11+ optimizer `tf.keras.optimizers.SGD` runs slowly on M1/M2 Macs, please use the legacy Keras optimizer instead, loc
ated at `tf.keras.optimizers.legacy.SGD`.
```

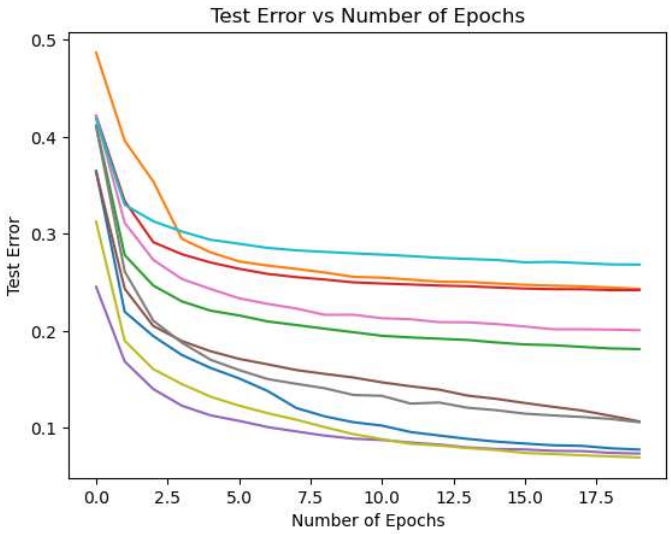
```
Epoch 1/20
1875/1875 [==============================] - 1s 579us/step - loss: 0.0720 - accuracy: 0.4503 - val_loss: 0.0578 - val_accuracy: 0.5784
Epoch 2/20
1875/1875 [==============================] - 1s 537us/step - loss: 0.0533 - accuracy: 0.6096 - val_loss: 0.0473 - val_accuracy: 0.6662
Epoch 3/20
1875/1875 [==============================] - 1s 534us/step - loss: 0.0442 - accuracy: 0.6886 - val_loss: 0.0412 - val_accuracy: 0.7086
Epoch 4/20
1875/1875 [==============================] - 1s 533us/step - loss: 0.0405 - accuracy: 0.7112 - val_loss: 0.0389 - val_accuracy: 0.7210
Epoch 5/20
1875/1875 [==============================] - 1s 537us/step - loss: 0.0386 - accuracy: 0.7225 - val_loss: 0.0374 - val_accuracy: 0.7295
Epoch 6/20
1875/1875 [==============================] - 1s 532us/step - loss: 0.0373 - accuracy: 0.7303 - val_loss: 0.0364 - val_accuracy: 0.7360
Epoch 7/20
1875/1875 [==============================] - 1s 532us/step - loss: 0.0364 - accuracy: 0.7357 - val_loss: 0.0356 - val_accuracy: 0.7413
Epoch 8/20
1875/1875 [==============================] - 1s 544us/step - loss: 0.0356 - accuracy: 0.7398 - val_loss: 0.0350 - val_accuracy: 0.7446
Epoch 9/20
1875/1875 [==============================] - 1s 557us/step - loss: 0.0350 - accuracy: 0.7432 - val_loss: 0.0344 - val_accuracy: 0.7471
Epoch 10/20
1875/1875 [==============================] - 1s 533us/step - loss: 0.0345 - accuracy: 0.7454 - val_loss: 0.0340 - val_accuracy: 0.7500
Epoch 11/20
1875/1875 [==============================] - 1s 540us/step - loss: 0.0341 - accuracy: 0.7473 - val_loss: 0.0337 - val_accuracy: 0.7512
Epoch 12/20
1875/1875 [==============================] - 1s 546us/step - loss: 0.0337 - accuracy: 0.7489 - val_loss: 0.0333 - val_accuracy: 0.7522
Epoch 13/20
1875/1875 [==============================] - 1s 537us/step - loss: 0.0334 - accuracy: 0.7506 - val_loss: 0.0330 - val_accuracy: 0.7533
Epoch 14/20
1875/1875 [==============================] - 1s 542us/step - loss: 0.0331 - accuracy: 0.7521 - val_loss: 0.0328 - val_accuracy: 0.7540
Epoch 15/20
1875/1875 [==============================] - 1s 568us/step - loss: 0.0329 - accuracy: 0.7533 - val_loss: 0.0326 - val_accuracy: 0.7553
Epoch 16/20
1875/1875 [==============================] - 1s 539us/step - loss: 0.0327 - accuracy: 0.7544 - val_loss: 0.0324 - val_accuracy: 0.7564
Epoch 17/20
1875/1875 [==============================] - 1s 538us/step - loss: 0.0324 - accuracy: 0.7554 - val_loss: 0.0322 - val_accuracy: 0.7570
Epoch 18/20
1875/1875 [==============================] - 1s 533us/step - loss: 0.0323 - accuracy: 0.7563 - val_loss: 0.0320 - val_accuracy: 0.7571
Epoch 19/20
1875/1875 [==============================] - 1s 540us/step - loss: 0.0321 - accuracy: 0.7567 - val_loss: 0.0319 - val_accuracy: 0.7580
Epoch 20/20
1875/1875 [==============================] - 1s 534us/step - loss: 0.0319 - accuracy: 0.7576 - val_loss: 0.0317 - val_accuracy: 0.7580
WARNING:absl:At this time, the v2.11+ optimizer `tf.keras.optimizers.SGD` runs slowly on M1/M2 Macs, please use the legacy Keras optimizer instead, loc
ated at `tf.keras.optimizers.legacy.SGD`.
Epoch 1/20
1875/1875 [==============================] - 1s 581us/step - loss: 0.0681 - accuracy: 0.5259 - val_loss: 0.0442 - val_accuracy: 0.7546
Epoch 2/20
1875/1875 [==============================] - 1s 534us/step - loss: 0.0381 - accuracy: 0.7918 - val_loss: 0.0327 - val_accuracy: 0.8316
Epoch 3/20
1875/1875 [==============================] - 1s 542us/step - loss: 0.0308 - accuracy: 0.8410 - val_loss: 0.0282 - val_accuracy: 0.8600
Epoch 4/20
1875/1875 [==============================] - 1s 555us/step - loss: 0.0275 - accuracy: 0.8630 - val_loss: 0.0257 - val_accuracy: 0.8772
Epoch 5/20
1875/1875 [==============================] - 1s 533us/step - loss: 0.0255 - accuracy: 0.8764 - val_loss: 0.0240 - val_accuracy: 0.8871
Epoch 6/20
1875/1875 [==============================] - 1s 538us/step - loss: 0.0240 - accuracy: 0.8856 - val_loss: 0.0228 - val_accuracy: 0.8928
Epoch 7/20
1875/1875 [==============================] - 1s 547us/step - loss: 0.0230 - accuracy: 0.8917 - val_loss: 0.0219 - val_accuracy: 0.8992
Epoch 8/20
1875/1875 [==============================] - 1s 536us/step - loss: 0.0221 - accuracy: 0.8976 - val_loss: 0.0211 - val_accuracy: 0.9037
Epoch 9/20
1875/1875 [==============================] - 1s 533us/step - loss: 0.0213 - accuracy: 0.9020 - val_loss: 0.0204 - val_accuracy: 0.9080
Epoch 10/20
1875/1875 [==============================] - 1s 540us/step - loss: 0.0207 - accuracy: 0.9056 - val_loss: 0.0198 - val_accuracy: 0.9112
Epoch 11/20
1875/1875 [==============================] - 1s 567us/step - loss: 0.0202 - accuracy: 0.9087 - val_loss: 0.0193 - val_accuracy: 0.9125
Epoch 12/20
1875/1875 [==============================] - 1s 535us/step - loss: 0.0197 - accuracy: 0.9114 - val_loss: 0.0189 - val_accuracy: 0.9152
Epoch 13/20
1875/1875 [==============================] - 1s 534us/step - loss: 0.0192 - accuracy: 0.9140 - val_loss: 0.0185 - val_accuracy: 0.9172
Epoch 14/20
1875/1875 [==============================] - 1s 534us/step - loss: 0.0188 - accuracy: 0.9160 - val_loss: 0.0181 - val_accuracy: 0.9202
Epoch 15/20
1875/1875 [==============================] - 1s 534us/step - loss: 0.0185 - accuracy: 0.9184 - val_loss: 0.0178 - val_accuracy: 0.9218
Epoch 16/20
1875/1875 [==============================] - 1s 534us/step - loss: 0.0181 - accuracy: 0.9203 - val_loss: 0.0175 - val_accuracy: 0.9223
Epoch 17/20
1875/1875 [==============================] - 1s 537us/step - loss: 0.0178 - accuracy: 0.9219 - val_loss: 0.0172 - val_accuracy: 0.9237
Epoch 18/20
1875/1875 [==============================] - 1s 540us/step - loss: 0.0175 - accuracy: 0.9235 - val_loss: 0.0170 - val_accuracy: 0.9240
Epoch 19/20
1875/1875 [==============================] - 1s 548us/step - loss: 0.0172 - accuracy: 0.9248 - val_loss: 0.0167 - val_accuracy: 0.9258
Epoch 20/20
1875/1875 [==============================] - 1s 551us/step - loss: 0.0170 - accuracy: 0.9258 - val_loss: 0.0165 - val_accuracy: 0.9265
WARNING:absl:At this time, the v2.11+ optimizer `tf.keras.optimizers.SGD` runs slowly on M1/M2 Macs, please use the legacy Keras optimizer instead, loc
ated at `tf.keras.optimizers.legacy.SGD`.
```

```
Epoch 1/20
1875/1875 [==============================] - 1s 587us/step - loss: 0.0740 - accuracy: 0.4469 - val_loss: 0.0545 - val_accuracy: 0.6361
Epoch 2/20
1875/1875 [==============================] - 1s 534us/step - loss: 0.0465 - accuracy: 0.7013 - val_loss: 0.0398 - val_accuracy: 0.7566
Epoch 3/20
1875/1875 [==============================] - 1s 532us/step - loss: 0.0381 - accuracy: 0.7665 - val_loss: 0.0348 - val_accuracy: 0.7950
Epoch 4/20
1875/1875 [==============================] - 1s 534us/step - loss: 0.0344 - accuracy: 0.7914 - val_loss: 0.0322 - val_accuracy: 0.8105
Epoch 5/20
1875/1875 [==============================] - 1s 533us/step - loss: 0.0323 - accuracy: 0.8052 - val_loss: 0.0305 - val_accuracy: 0.8210
Epoch 6/20
1875/1875 [==============================] - 1s 564us/step - loss: 0.0308 - accuracy: 0.8143 - val_loss: 0.0292 - val_accuracy: 0.8289
Epoch 7/20
1875/1875 [==============================] - 1s 534us/step - loss: 0.0297 - accuracy: 0.8215 - val_loss: 0.0283 - val_accuracy: 0.8345
Epoch 8/20
1875/1875 [==============================] - 1s 534us/step - loss: 0.0288 - accuracy: 0.8277 - val_loss: 0.0275 - val_accuracy: 0.8404
Epoch 9/20
1875/1875 [==============================] - 1s 533us/step - loss: 0.0280 - accuracy: 0.8338 - val_loss: 0.0268 - val_accuracy: 0.8444
Epoch 10/20
1875/1875 [==============================] - 1s 532us/step - loss: 0.0273 - accuracy: 0.8395 - val_loss: 0.0262 - val_accuracy: 0.8481
Epoch 11/20
1875/1875 [==============================] - 1s 539us/step - loss: 0.0266 - accuracy: 0.8450 - val_loss: 0.0256 - val_accuracy: 0.8531
Epoch 12/20
1875/1875 [==============================] - 1s 533us/step - loss: 0.0259 - accuracy: 0.8494 - val_loss: 0.0249 - val_accuracy: 0.8571
Epoch 13/20
1875/1875 [==============================] - 1s 533us/step - loss: 0.0253 - accuracy: 0.8540 - val_loss: 0.0244 - val_accuracy: 0.8606
Epoch 14/20
1875/1875 [==============================] - 1s 545us/step - loss: 0.0247 - accuracy: 0.8581 - val_loss: 0.0238 - val_accuracy: 0.8668
Epoch 15/20
1875/1875 [==============================] - 1s 560us/step - loss: 0.0241 - accuracy: 0.8629 - val_loss: 0.0232 - val_accuracy: 0.8701
Epoch 16/20
1875/1875 [==============================] - 1s 545us/step - loss: 0.0235 - accuracy: 0.8672 - val_loss: 0.0226 - val_accuracy: 0.8744
Epoch 17/20
1875/1875 [==============================] - 1s 534us/step - loss: 0.0229 - accuracy: 0.8717 - val_loss: 0.0221 - val_accuracy: 0.8784
Epoch 18/20
1875/1875 [==============================] - 1s 533us/step - loss: 0.0224 - accuracy: 0.8761 - val_loss: 0.0215 - val_accuracy: 0.8822
Epoch 19/20
1875/1875 [==============================] - 1s 533us/step - loss: 0.0219 - accuracy: 0.8815 - val_loss: 0.0210 - val_accuracy: 0.8876
Epoch 20/20
1875/1875 [==============================] - 1s 533us/step - loss: 0.0214 - accuracy: 0.8867 - val_loss: 0.0205 - val_accuracy: 0.8934
WARNING:absl:At this time, the v2.11+ optimizer `tf.keras.optimizers.SGD` runs slowly on M1/M2 Macs, please use the legacy Keras optimizer instead, loc
ated at `tf.keras.optimizers.legacy.SGD`.
Epoch 1/20
1875/1875 [==============================] - 2s 862us/step - loss: 0.0776 - accuracy: 0.3936 - val_loss: 0.0612 - val_accuracy: 0.5786
Epoch 2/20
1875/1875 [==============================] - 1s 560us/step - loss: 0.0530 - accuracy: 0.6416 - val_loss: 0.0472 - val_accuracy: 0.6886
Epoch 3/20
1875/1875 [==============================] - 1s 537us/step - loss: 0.0458 - accuracy: 0.7042 - val_loss: 0.0432 - val_accuracy: 0.7268
Epoch 4/20
1875/1875 [==============================] - 1s 535us/step - loss: 0.0430 - accuracy: 0.7306 - val_loss: 0.0412 - val_accuracy: 0.7465
Epoch 5/20
1875/1875 [==============================] - 1s 535us/step - loss: 0.0413 - accuracy: 0.7471 - val_loss: 0.0399 - val_accuracy: 0.7568
Epoch 6/20
1875/1875 [==============================] - 1s 535us/step - loss: 0.0400 - accuracy: 0.7573 - val_loss: 0.0388 - val_accuracy: 0.7664
Epoch 7/20
1875/1875 [==============================] - 1s 535us/step - loss: 0.0391 - accuracy: 0.7652 - val_loss: 0.0380 - val_accuracy: 0.7722
Epoch 8/20
1875/1875 [==============================] - 1s 536us/step - loss: 0.0384 - accuracy: 0.7717 - val_loss: 0.0374 - val_accuracy: 0.7770
Epoch 9/20
1875/1875 [==============================] - 1s 558us/step - loss: 0.0378 - accuracy: 0.7766 - val_loss: 0.0368 - val_accuracy: 0.7835
Epoch 10/20
1875/1875 [==============================] - 1s 547us/step - loss: 0.0373 - accuracy: 0.7808 - val_loss: 0.0364 - val_accuracy: 0.7834
Epoch 11/20
1875/1875 [==============================] - 1s 542us/step - loss: 0.0368 - accuracy: 0.7841 - val_loss: 0.0360 - val_accuracy: 0.7870
Epoch 12/20
1875/1875 [==============================] - 1s 536us/step - loss: 0.0364 - accuracy: 0.7878 - val_loss: 0.0357 - val_accuracy: 0.7880
Epoch 13/20
1875/1875 [==============================] - 1s 533us/step - loss: 0.0361 - accuracy: 0.7902 - val_loss: 0.0354 - val_accuracy: 0.7910
Epoch 14/20
1875/1875 [==============================] - 1s 535us/step - loss: 0.0358 - accuracy: 0.7918 - val_loss: 0.0351 - val_accuracy: 0.7912
Epoch 15/20
1875/1875 [==============================] - 1s 536us/step - loss: 0.0355 - accuracy: 0.7941 - val_loss: 0.0349 - val_accuracy: 0.7930
Epoch 16/20
1875/1875 [==============================] - 1s 543us/step - loss: 0.0352 - accuracy: 0.7958 - val_loss: 0.0347 - val_accuracy: 0.7955
Epoch 17/20
1875/1875 [==============================] - 1s 537us/step - loss: 0.0350 - accuracy: 0.7976 - val_loss: 0.0344 - val_accuracy: 0.7984
Epoch 18/20
1875/1875 [==============================] - 1s 535us/step - loss: 0.0347 - accuracy: 0.7994 - val_loss: 0.0343 - val_accuracy: 0.7984
Epoch 19/20
1875/1875 [==============================] - 1s 535us/step - loss: 0.0345 - accuracy: 0.8007 - val_loss: 0.0340 - val_accuracy: 0.7987
Epoch 20/20
1875/1875 [==============================] - 1s 534us/step - loss: 0.0343 - accuracy: 0.8022 - val_loss: 0.0339 - val_accuracy: 0.7993
WARNING:absl:At this time, the v2.11+ optimizer `tf.keras.optimizers.SGD` runs slowly on M1/M2 Macs, please use the legacy Keras optimizer instead, loc
ated at `tf.keras.optimizers.legacy.SGD`.
```

```
Epoch 1/20
1875/1875 [==============================] - 1s 573us/step - loss: 0.0766 - accuracy: 0.4066 - val_loss: 0.0584 - val_accuracy: 0.5892
Epoch 2/20
1875/1875 [==============================] - 1s 535us/step - loss: 0.0509 - accuracy: 0.6632 - val_loss: 0.0429 - val_accuracy: 0.7388
Epoch 3/20
1875/1875 [==============================] - 1s 535us/step - loss: 0.0406 - accuracy: 0.7593 - val_loss: 0.0369 - val_accuracy: 0.7895
Epoch 4/20
1875/1875 [==============================] - 1s 559us/step - loss: 0.0365 - accuracy: 0.7941 - val_loss: 0.0342 - val_accuracy: 0.8122
Epoch 5/20
1875/1875 [==============================] - 1s 554us/step - loss: 0.0342 - accuracy: 0.8137 - val_loss: 0.0324 - val_accuracy: 0.8297
Epoch 6/20
1875/1875 [==============================] - 1s 535us/step - loss: 0.0328 - accuracy: 0.8283 - val_loss: 0.0312 - val_accuracy: 0.8404
Epoch 7/20
1875/1875 [==============================] - 1s 535us/step - loss: 0.0317 - accuracy: 0.8385 - val_loss: 0.0303 - val_accuracy: 0.8497
Epoch 8/20
1875/1875 [==============================] - 1s 565us/step - loss: 0.0308 - accuracy: 0.8475 - val_loss: 0.0296 - val_accuracy: 0.8546
Epoch 9/20
1875/1875 [==============================] - 1s 537us/step - loss: 0.0301 - accuracy: 0.8539 - val_loss: 0.0290 - val_accuracy: 0.8592
Epoch 10/20
1875/1875 [==============================] - 1s 546us/step - loss: 0.0295 - accuracy: 0.8594 - val_loss: 0.0285 - val_accuracy: 0.8661
Epoch 11/20
1875/1875 [==============================] - 1s 535us/step - loss: 0.0290 - accuracy: 0.8645 - val_loss: 0.0280 - val_accuracy: 0.8670
Epoch 12/20
1875/1875 [==============================] - 1s 534us/step - loss: 0.0285 - accuracy: 0.8684 - val_loss: 0.0276 - val_accuracy: 0.8750
Epoch 13/20
1875/1875 [==============================] - 1s 533us/step - loss: 0.0281 - accuracy: 0.8711 - val_loss: 0.0273 - val_accuracy: 0.8739
Epoch 14/20
1875/1875 [==============================] - 1s 533us/step - loss: 0.0277 - accuracy: 0.8747 - val_loss: 0.0269 - val_accuracy: 0.8793
Epoch 15/20
1875/1875 [==============================] - 1s 535us/step - loss: 0.0274 - accuracy: 0.8780 - val_loss: 0.0266 - val_accuracy: 0.8818
Epoch 16/20
1875/1875 [==============================] - 1s 533us/step - loss: 0.0270 - accuracy: 0.8798 - val_loss: 0.0263 - val_accuracy: 0.8854
Epoch 17/20
1875/1875 [==============================] - 1s 535us/step - loss: 0.0267 - accuracy: 0.8821 - val_loss: 0.0260 - val_accuracy: 0.8872
Epoch 18/20
1875/1875 [==============================] - 1s 535us/step - loss: 0.0265 - accuracy: 0.8845 - val_loss: 0.0258 - val_accuracy: 0.8888
Epoch 19/20
1875/1875 [==============================] - 1s 533us/step - loss: 0.0262 - accuracy: 0.8864 - val_loss: 0.0256 - val_accuracy: 0.8908
Epoch 20/20
1875/1875 [==============================] - 1s 561us/step - loss: 0.0260 - accuracy: 0.8880 - val_loss: 0.0254 - val_accuracy: 0.8942
WARNING:absl:At this time, the v2.11+ optimizer `tf.keras.optimizers.SGD` runs slowly on M1/M2 Macs, please use the legacy Keras optimizer instead, loc
ated at `tf.keras.optimizers.legacy.SGD`.
Epoch 1/20
1875/1875 [==============================] - 1s 580us/step - loss: 0.0719 - accuracy: 0.4564 - val_loss: 0.0494 - val_accuracy: 0.6874
Epoch 2/20
1875/1875 [==============================] - 1s 537us/step - loss: 0.0418 - accuracy: 0.7536 - val_loss: 0.0354 - val_accuracy: 0.8102
Epoch 3/20
1875/1875 [==============================] - 1s 535us/step - loss: 0.0338 - accuracy: 0.8176 - val_loss: 0.0306 - val_accuracy: 0.8394
Epoch 4/20
1875/1875 [==============================] - 1s 534us/step - loss: 0.0302 - accuracy: 0.8399 - val_loss: 0.0280 - val_accuracy: 0.8549
Epoch 5/20
1875/1875 [==============================] - 1s 533us/step - loss: 0.0279 - accuracy: 0.8548 - val_loss: 0.0261 - val_accuracy: 0.8678
Epoch 6/20
1875/1875 [==============================] - 1s 535us/step - loss: 0.0262 - accuracy: 0.8650 - val_loss: 0.0247 - val_accuracy: 0.8771
Epoch 7/20
1875/1875 [==============================] - 1s 535us/step - loss: 0.0248 - accuracy: 0.8746 - val_loss: 0.0234 - val_accuracy: 0.8849
Epoch 8/20
1875/1875 [==============================] - 1s 534us/step - loss: 0.0237 - accuracy: 0.8824 - val_loss: 0.0223 - val_accuracy: 0.8916
Epoch 9/20
1875/1875 [==============================] - 1s 534us/step - loss: 0.0226 - accuracy: 0.8899 - val_loss: 0.0213 - val_accuracy: 0.8993
Epoch 10/20
1875/1875 [==============================] - 1s 533us/step - loss: 0.0216 - accuracy: 0.8971 - val_loss: 0.0204 - val_accuracy: 0.9065
Epoch 11/20
1875/1875 [==============================] - 1s 534us/step - loss: 0.0208 - accuracy: 0.9033 - val_loss: 0.0197 - val_accuracy: 0.9116
Epoch 12/20
1875/1875 [==============================] - 1s 533us/step - loss: 0.0201 - accuracy: 0.9075 - val_loss: 0.0191 - val_accuracy: 0.9164
Epoch 13/20
1875/1875 [==============================] - 1s 554us/step - loss: 0.0195 - accuracy: 0.9108 - val_loss: 0.0186 - val_accuracy: 0.9182
Epoch 14/20
1875/1875 [==============================] - 1s 535us/step - loss: 0.0191 - accuracy: 0.9134 - val_loss: 0.0182 - val_accuracy: 0.9209
Epoch 15/20
1875/1875 [==============================] - 1s 548us/step - loss: 0.0186 - accuracy: 0.9157 - val_loss: 0.0178 - val_accuracy: 0.9229
Epoch 16/20
1875/1875 [==============================] - 1s 582us/step - loss: 0.0182 - accuracy: 0.9184 - val_loss: 0.0174 - val_accuracy: 0.9259
Epoch 17/20
1875/1875 [==============================] - 1s 538us/step - loss: 0.0179 - accuracy: 0.9203 - val_loss: 0.0171 - val_accuracy: 0.9270
Epoch 18/20
1875/1875 [==============================] - 1s 533us/step - loss: 0.0176 - accuracy: 0.9218 - val_loss: 0.0169 - val_accuracy: 0.9283
Epoch 19/20
1875/1875 [==============================] - 1s 534us/step - loss: 0.0173 - accuracy: 0.9238 - val_loss: 0.0166 - val_accuracy: 0.9294
Epoch 20/20
1875/1875 [==============================] - 1s 538us/step - loss: 0.0170 - accuracy: 0.9247 - val_loss: 0.0163 - val_accuracy: 0.9306
WARNING:absl:At this time, the v2.11+ optimizer `tf.keras.optimizers.SGD` runs slowly on M1/M2 Macs, please use the legacy Keras optimizer instead, loc
ated at `tf.keras.optimizers.legacy.SGD`.
```

```
Epoch 1/20
1875/1875 [==============================] - 1s 583us/step - loss: 0.0800 - accuracy: 0.3736 - val_loss: 0.0597 - val_accuracy: 0.5816
Epoch 2/20
1875/1875 [==============================] - 1s 537us/step - loss: 0.0521 - accuracy: 0.6387 - val_loss: 0.0469 - val_accuracy: 0.6700
Epoch 3/20
1875/1875 [==============================] - 1s 537us/step - loss: 0.0456 - accuracy: 0.6791 - val_loss: 0.0436 - val_accuracy: 0.6870
Epoch 4/20
1875/1875 [==============================] - 1s 541us/step - loss: 0.0432 - accuracy: 0.6928 - val_loss: 0.0420 - val_accuracy: 0.6976
Epoch 5/20
1875/1875 [==============================] - 1s 538us/step - loss: 0.0417 - accuracy: 0.7011 - val_loss: 0.0409 - val_accuracy: 0.7061
Epoch 6/20
1875/1875 [==============================] - 1s 537us/step - loss: 0.0407 - accuracy: 0.7072 - val_loss: 0.0400 - val_accuracy: 0.7102
Epoch 7/20
1875/1875 [==============================] - 1s 537us/step - loss: 0.0399 - accuracy: 0.7123 - val_loss: 0.0394 - val_accuracy: 0.7145
Epoch 8/20
1875/1875 [==============================] - 1s 539us/step - loss: 0.0392 - accuracy: 0.7154 - val_loss: 0.0388 - val_accuracy: 0.7170
Epoch 9/20
1875/1875 [==============================] - 1s 539us/step - loss: 0.0387 - accuracy: 0.7186 - val_loss: 0.0383 - val_accuracy: 0.7186
Epoch 10/20
1875/1875 [==============================] - 1s 555us/step - loss: 0.0382 - accuracy: 0.7212 - val_loss: 0.0379 - val_accuracy: 0.7201
Epoch 11/20
1875/1875 [==============================] - 1s 561us/step - loss: 0.0378 - accuracy: 0.7233 - val_loss: 0.0376 - val_accuracy: 0.7214
Epoch 12/20
1875/1875 [==============================] - 1s 543us/step - loss: 0.0374 - accuracy: 0.7257 - val_loss: 0.0372 - val_accuracy: 0.7229
Epoch 13/20
1875/1875 [==============================] - 1s 539us/step - loss: 0.0371 - accuracy: 0.7271 - val_loss: 0.0369 - val_accuracy: 0.7247
Epoch 14/20
1875/1875 [==============================] - 1s 537us/step - loss: 0.0368 - accuracy: 0.7290 - val_loss: 0.0367 - val_accuracy: 0.7259
Epoch 15/20
1875/1875 [==============================] - 1s 544us/step - loss: 0.0365 - accuracy: 0.7301 - val_loss: 0.0365 - val_accuracy: 0.7269
Epoch 16/20
1875/1875 [==============================] - 1s 538us/step - loss: 0.0362 - accuracy: 0.7313 - val_loss: 0.0362 - val_accuracy: 0.7293
Epoch 17/20
1875/1875 [==============================] - 1s 539us/step - loss: 0.0360 - accuracy: 0.7322 - val_loss: 0.0360 - val_accuracy: 0.7289
Epoch 18/20
1875/1875 [==============================] - 1s 537us/step - loss: 0.0358 - accuracy: 0.7332 - val_loss: 0.0358 - val_accuracy: 0.7302
Epoch 19/20
1875/1875 [==============================] - 1s 539us/step - loss: 0.0355 - accuracy: 0.7344 - val_loss: 0.0356 - val_accuracy: 0.7315
Epoch 20/20
1875/1875 [==============================] - 1s 540us/step - loss: 0.0353 - accuracy: 0.7349 - val_loss: 0.0355 - val_accuracy: 0.7317
```



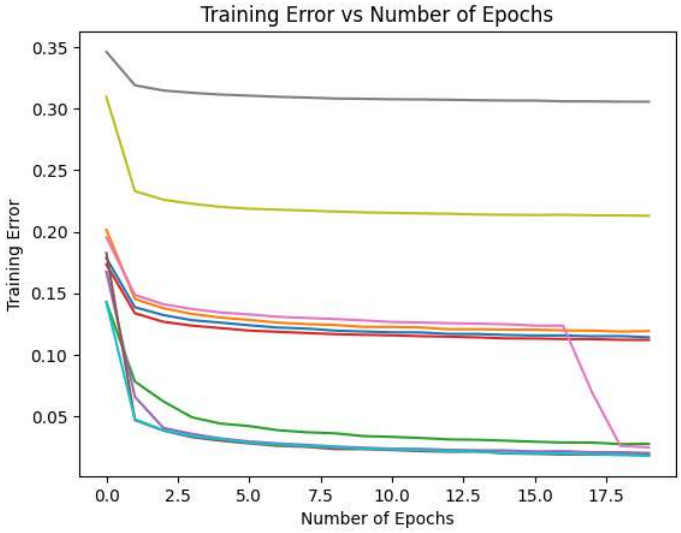


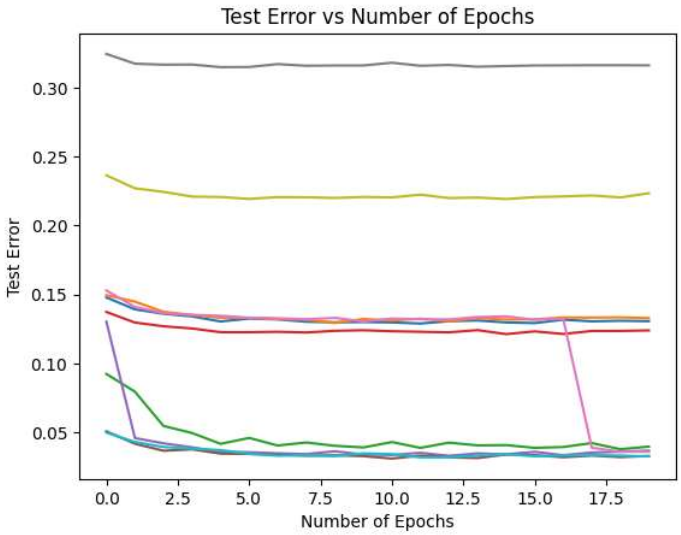
While each model still converges to a relatively different final training and test error, the initial training errors for each model are significantly higher. The final training errors for each model seem to be within the same range as in question 2, meaning that the drop in training error after the first few epochs is much steeper with a learning rate of 0.01 when compared to a learning rate of 0.1.

Furthermore, the errors seem to converge after about 6-7 epochs when the learning rate is 0.01, while the errors seemed to converge after about 3-4 epochs when the learning rate was 0.1. This makes sense, as the learning rate determines how fast the model will update its weights. A larger learning rate can lead to faster convergence, which is evident in this case.

4. [1 mark]: Instead of standard Stochastic Gradient Descent, most modern machine learning researchers use a variation of it called Adam that automatically adjusts the learning rate and does a better local estimate of the gradient. We can switch to this by changing `optimizer=tf.keras.optimizers.SGD(learning_rate=0.1)` to `optimizer="adam"`. Repeat question 2 (generating the same 2 plots) using the Adam optimizer. How does this affect the learning performance?

In [ ]:
```
import numpy as np

train_errs = []
test_errs = []

def get_error(accuracy):
    return 1 - accuracy

err_func = np.vectorize(get_error)

num_models = 10

for i in range(num_models):
    # Create new model
    model = tf.keras.models.Sequential([
      tf.keras.layers.Flatten(input_shape=(28, 28)),   # input is a 28x28 image
      tf.keras.layers.Dense(32, activation='relu'),    # 32 neurons in the middle "hidden" layer
      tf.keras.layers.Dense(10, activation='relu')     # 10 outputs (one for each category)
    ])

    # Compile model
    model.compile(optimizer="adam", # use stochastic gradient descent
                  loss=my_loss,
                  metrics=['accuracy']  # in addition to the loss, also compute the categorization accuracy
                 )

    # Train each model for 20 epochs
    history = model.fit(x_train, y_train_target, epochs=20, validation_data=(x_test, y_test_target), verbose=0);

    # Store training and test errors
    train_err = err_func(history.history['accuracy'])
    train_errs.append(train_err)

    test_err = err_func(history.history['val_accuracy'])
    test_errs.append(test_err)

# Plot training errors
plt.figure()
for i in range(num_models):
    plt.plot(train_errs[i])
plt.title("Training Error vs Number of Epochs")
plt.ylabel("Training Error")
plt.xlabel("Number of Epochs")
plt.show()

# Plot test errors
plt.figure()
for i in range(num_models):
    plt.plot(test_errs[i])
plt.title("Test Error vs Number of Epochs")
plt.ylabel("Test Error")
plt.xlabel("Number of Epochs")
plt.show()
```





With the Adam optimizer, the training and test errors seem to converge much quicker, after about 2 epochs. However, the final training and test errors seem to be in the same range as in question 2, so changing the optimizer does not seem to improve the final performance of the model.

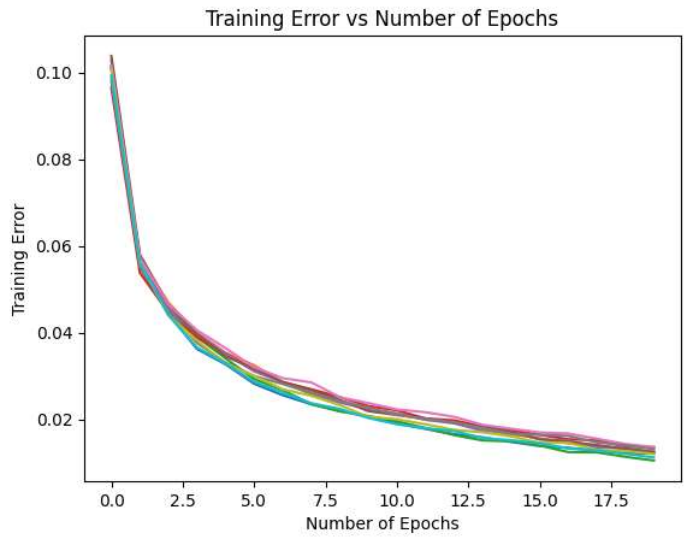
5. [1 mark]: To improve performance, we can also change the output layer of neurons to be a `softmax` layer. This means that they will automatically be normalized with the softmax function (so the output of neuron i will be $e^{a_i} / \sum_j e^{a_j}$). This should push the output value closer to the sort of output we want (all zeros with a single 1). Furthermore, instead of doing the error as $(t-y)^2$, we can instead choose an error function that is only concerned about categorization. After all, the only thing we care about when categorizing is which of the outputs is largest, not whether they are exactly 0 or 1. Conveniently, tensorflow (and many other deep learning libraries) also contains a version of this function that works directly on category labels, so we don't have to do the `y_train_target = np.eye(10)[y_train]` trick in the code above. The resulting network looks like this:

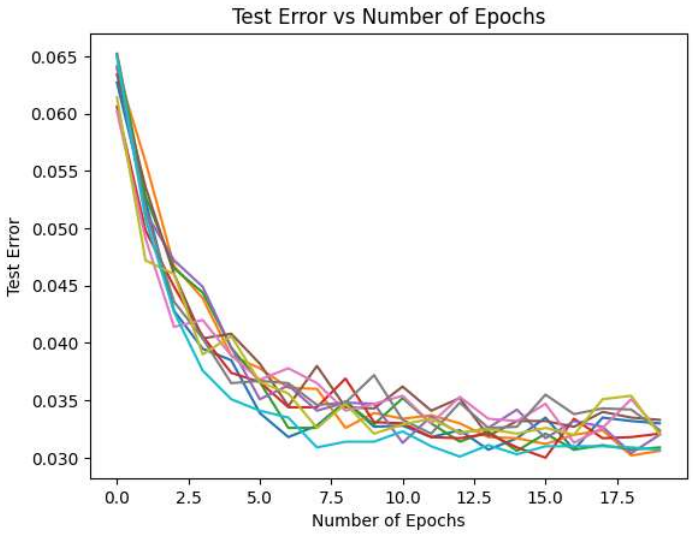
In [ ]:
```
model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),   # input is a 28x28 image
  tf.keras.layers.Dense(32, activation='relu'),    # 32 neurons in the middle "hidden" layer
  tf.keras.layers.Dense(10, activation='softmax')  # 10 outputs (one for each category)
])

model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy']  # in addition to the loss, also compute the categorization accuracy
             )

model.fit(x_train, y_train, epochs=20, validation_data=(x_test, y_test))   # note that we now use y_train, not y_train_target
```

Generate the same 2 plots as in question 2. How do these changes affect the learning performance?

In [ ]:
```
import numpy as np

train_errs = []
test_errs = []

def get_error(accuracy):
    return 1 - accuracy

err_func = np.vectorize(get_error)
```

```
num_models = 10

for i in range(num_models):
    # Create new model
    model = tf.keras.models.Sequential([
              tf.keras.layers.Flatten(input_shape=(28, 28)),   # input is a 28x28 image
              tf.keras.layers.Dense(32, activation='relu'),    # 32 neurons in the middle "hidden" layer
              tf.keras.layers.Dense(10, activation='softmax')  # 10 outputs (one for each category)
             ])

    # Compile model
    model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy']  # in addition to the loss, also compute the categorization accuracy
             )

    # Train each model for 20 epochs
    history = model.fit(x_train, y_train, epochs=20, validation_data=(x_test, y_test), verbose=0);

    # Store training and test errors
    train_err = err_func(history.history['accuracy'])
    train_errs.append(train_err)

    test_err = err_func(history.history['val_accuracy'])
    test_errs.append(test_err)

# Plot training errors
plt.figure()
for i in range(num_models):
    plt.plot(train_errs[i])
plt.title("Training Error vs Number of Epochs")
plt.ylabel("Training Error")
plt.xlabel("Number of Epochs")
plt.show()

# Plot test errors
plt.figure()
for i in range(num_models):
    plt.plot(test_errs[i])
plt.title("Test Error vs Number of Epochs")
plt.ylabel("Test Error")
plt.xlabel("Number of Epochs")
plt.show()
```





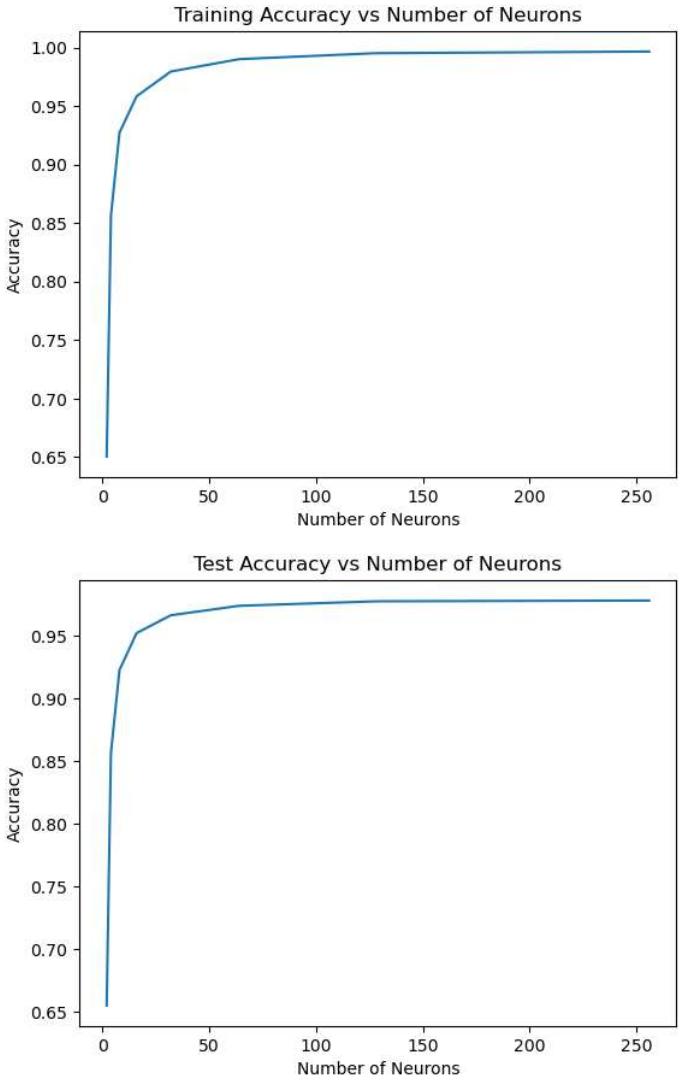
After the changes were made, all the models perform significantly better, with very low final training and test errors. In addition, they have very similar training and test errors throughout all the epochs, as opposed to before, when each model had relatively different initial and final training and test errors.

This is likely attributed to the `softmax` activation function that is used in the output layer. While the ReLU activation function is commonly used in hidden layers of neural networks, the `softmax` activation function is often more appropriate in the output layer as it produces a probability distribution over the classes that sum up to 1. This allows us to easily interpret the output as probabilities, which is why multi-class classification problems like this one often have the `softmax` activation function in the output layer.

6. [2 marks]: Using the final version of the model (from question 5), explore the performance of the model as you change the number of neurons in the hidden layer. Try 2, 4, 8, 16, 32, 64, 128, and 256. For each value, train 10 models for 10 epochs each. We do not need the testing accuracy over the epochs, so you can switch back to setting `epochs=10` rather than using the `for` loop. Plot the average training accuracy vs different numbers of neurons. On a separate plot, plot the average testing accuracy vs different numbers of neurons. What trend do you observe?

In [ ]:
```
neurons = [2, 4, 8, 16, 32, 64, 128, 256]

final_train_accs = []
final_test_accs = []

plt.figure()

for i in range(len(neurons)):
    total_train_acc = 0
    total_test_acc = 0
    # Train 10 models
    for j in range(10):
        # Create new model
        model = tf.keras.models.Sequential([
                tf.keras.layers.Flatten(input_shape=(28, 28)),   # input is a 28x28 image
                tf.keras.layers.Dense(neurons[i], activation='relu'),    # 32 neurons in the middle "hidden" layer
                tf.keras.layers.Dense(10, activation='softmax')  # 10 outputs (one for each category)
                ])

        # Compile model
        model.compile(optimizer='adam',
                loss='sparse_categorical_crossentropy',
                metrics=['accuracy']  # in addition to the loss, also compute the categorization accuracy
                )

        # Train each model for 10 epochs
        history = model.fit(x_train, y_train, epochs=10, validation_data=(x_test, y_test), verbose=0);

        # Store training and test errors
        total_train_acc += history.history['accuracy'][-1]
        total_test_acc += history.history['val_accuracy'][-1]

    # Calculate average accuracy across 10 models
    avg_train_acc = total_train_acc / 10
    avg_test_acc = total_test_acc / 10

    # Plot training accuracies
    final_train_accs.append(avg_train_acc)
    final_test_accs.append(avg_test_acc)

plt.plot(neurons, final_train_accs)
plt.title("Training Accuracy vs Number of Neurons")
plt.ylabel("Accuracy")
plt.xlabel("Number of Neurons")
plt.show()

plt.figure()
plt.plot(neurons, final_test_accs)
plt.title("Test Accuracy vs Number of Neurons")
plt.ylabel("Accuracy")
```

```
plt.xlabel("Number of Neurons")
plt.show()
```

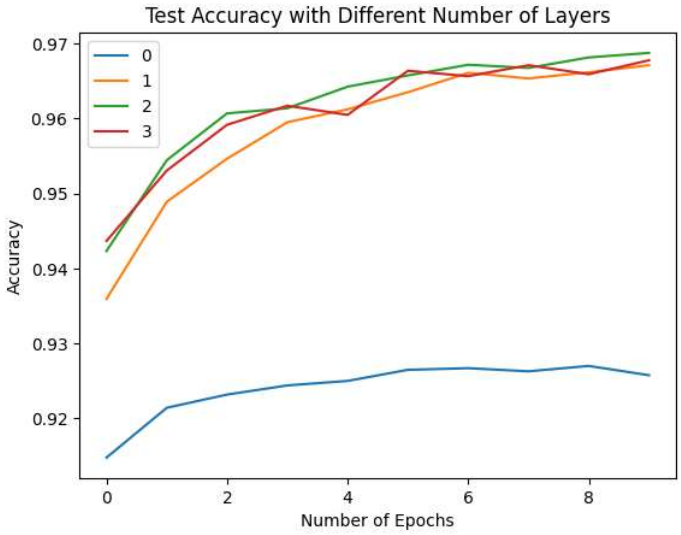


From the above graphs, we clearly see that as the number of neurons increases, the training and test accuracies increase as well. However, the relationship is not linear, and in fact, we see diminishing returns as we increase the number of neurons. This then becomes tradeoff, as increasing the number of neurons increases the accuracy at the cost of computational efficiency. When processing large amounts of data, it may be beneficial to use a lower number of neurons, such as 16 or 32, as that will still allow the model to have a final accuracy above 90%.

7. [2 marks]: The overall goal is to make the network perform as well as possible on the testing data. What would you do if your job was to make the best possible network for this task? Try some things and report your results. For example, try adding a second hidden layer to the model. Try making it different sizes. What about having no hidden layer? Or more than 2? Report the results of your exploration.

In [ ]:
```
# We first try varying the number of hidden layers
layers = [0, 1, 2, 3]

for i in range(len(layers)):
    total_acc = np.zeros(10)

    # Train 5 models
    for j in range(5):
        # Create new model
        model = tf.keras.models.Sequential()

        model.add(tf.keras.layers.Flatten(input_shape=(28, 28)))

        for k in range(layers[i]):
            model.add(tf.keras.layers.Dense(32, activation='relu'))

        model.add(tf.keras.layers.Dense(10, activation='softmax'))

        # Compile model
        model.compile(optimizer='adam',
                loss='sparse_categorical_crossentropy',
                metrics=['accuracy']  # in addition to the loss, also compute the categorization accuracy
                )

        # Train each model for 10 epochs
```

```
        history = model.fit(x_train, y_train, epochs=10, validation_data=(x_test, y_test), verbose=0);

        # Store test errors
        total_acc = np.add(total_acc, history.history['val_accuracy'])

    # Calculate average accuracy across 5 models
    avg_acc = np.divide(total_acc, 5)

    # Plot training accuracies
    plt.plot(avg_acc)

plt.title("Test Accuracy with Different Number of Layers")
plt.legend(layers)
plt.ylabel("Accuracy")
plt.xlabel("Number of Epochs")
plt.show()
```



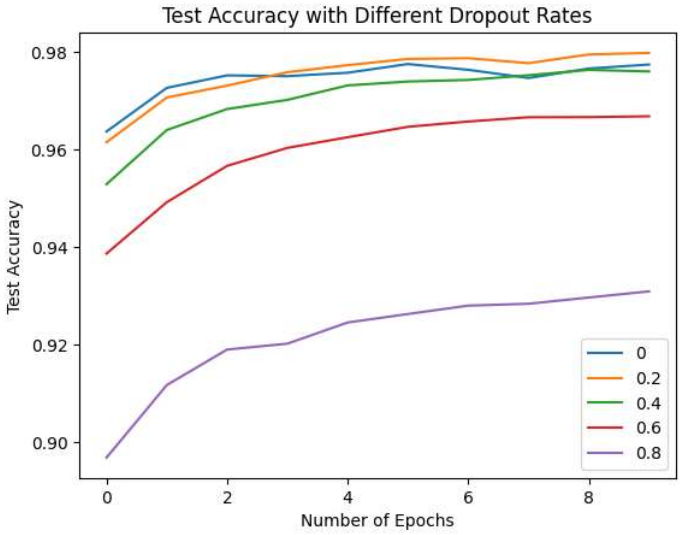
We see that having no hidden layers greatly impacts the final test accuracy. However, after adding 1 hidden layer, any additional layers do not impact the final test accuracy as much. From the above graph, it seems that having 2 hidden layers has the highest accuracy throughout the epochs, and so, we will choose to have 2 hidden layers in the final model.

In [ ]:
```
# We now optimize the number of neurons in each layer
neurons = [8, 32, 64, 128]
num_neurons = len(neurons)

# Store the maximum accuracy and the best number of neurons for each layer as (max_acc, layer1, layer2)
max_acc = (0, 0, 0)

for i in range(num_neurons):
    for j in range(num_neurons):
        total_acc = 0
        # Train 5 neurons
        for k in range(5):
            model = tf.keras.models.Sequential([
                    tf.keras.layers.Flatten(input_shape=(28, 28)),   # input is a 28x28 image
                    tf.keras.layers.Dense(neurons[i], activation='relu'),    # 32 neurons in the middle "hidden" layer
                    tf.keras.layers.Dense(neurons[j], activation='relu'),    # 32 neurons in the middle "hidden" layer
                    tf.keras.layers.Dense(10, activation='softmax')  # 10 outputs (one for each category)
                    ])
            model.compile(optimizer='adam',
                loss='sparse_categorical_crossentropy',
                metrics=['accuracy']  # in addition to the loss, also compute the categorization accuracy
                )
            history = model.fit(x_train, y_train, epochs=10, validation_data=(x_test, y_test), verbose=0);
            total_acc += history.history['val_accuracy'][-1]
        avg_acc = total_acc/5
        if avg_acc > max_acc[0]:
            max_acc = (avg_acc, neurons[i], neurons[j])

print(f"The maximum final test accuracy is {max_acc[0]}, with {max_acc[1]} neurons in the first layer and {max_acc[2]} neurons in the second layer.")
```

```
The maximum final test accuracy is 0.978059995174408, with 128 neurons in the first layer and 128 neurons in the second layer.
```

In [ ]:
```
# We now add dropout and vary the dropout rate
dropouts = [0, 0.2, 0.4, 0.6, 0.8]

for i in range(len(dropouts)):
    total_acc = np.zeros(10)

    # Train 5 models
    for j in range(5):
        # Create new model
        model = tf.keras.models.Sequential([
            tf.keras.layers.Flatten(input_shape=(28, 28)),   # input is a 28x28 image
            tf.keras.layers.Dense(128, activation='relu'),    # 32 neurons in the middle "hidden" layer
```

```
            tf.keras.layers.Dropout(dropouts[i]),
            tf.keras.layers.Dense(128, activation='relu'),    # 32 neurons in the middle "hidden" layer
            tf.keras.layers.Dropout(dropouts[i]),
            tf.keras.layers.Dense(10, activation='softmax')  # 10 outputs (one for each category)
            ])

        # Compile model
        model.compile(optimizer='adam',
            loss='sparse_categorical_crossentropy',
            metrics=['accuracy']  # in addition to the loss, also compute the categorization accuracy
            )

        # Train each model for 10 epochs
        history = model.fit(x_train, y_train, epochs=10, validation_data=(x_test, y_test), verbose=0);

        # Store test errors
        total_acc = np.add(total_acc, history.history['val_accuracy'])

    # Calculate average accuracy across 5 models
    avg_acc = np.divide(total_acc, 5)

    # Plot training accuracies
    plt.plot(avg_acc)

plt.title("Test Accuracy with Different Dropout Rates")
plt.legend(dropouts)
plt.ylabel("Test Accuracy")
plt.xlabel("Number of Epochs")
plt.show()
```



Having dropout layers in the model adds regularization to the training data. This helps to make the model generalize better to test data and prevent overfitting. As seen in the above graph, the model with the best test accuracy has a dropout rate of 0.2, so in the final model, we choose to have a dropout rate of 0.2.

In [ ]:
```
# Final model
model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),   # input is a 28x28 image
    tf.keras.layers.Dense(128, activation='relu'),    # 32 neurons in the middle "hidden" layer
    tf.keras.layers.Dropout(0.2), # Dropout rate of 0.2
    tf.keras.layers.Dense(128, activation='relu'),    # 32 neurons in the middle "hidden" layer
    tf.keras.layers.Dropout(0.2), # Dropout rate of 0.2
    tf.keras.layers.Dense(10, activation='softmax')  # 10 outputs (one for each category)
    ])

# Compile model
model.compile(optimizer='adam',
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy']  # in addition to the loss, also compute the categorization accuracy
    )

# Train each model for 10 epochs
history = model.fit(x_train, y_train, epochs=10, validation_data=(x_test, y_test), verbose=0);

# Print final test accuracy
print(f"The final test accuracy is {history.history['val_accuracy'][-1]*100:.2f}%.")
```

```
The final test accuracy is 98.05%.
```

In the final model, we implement two hidden layers with 128 neurons each. Each hidden layer is followed by a dropout layer with a dropout rate of 0.2. After training this model, we get a final test accuracy of 98.05%.

Part 2: CIFAR-10

The CIFAR-10 dataset is a bit harder than the MNIST dataset. Like MNIST, it consists of 10 categories of images, but now they are colour images and they are of different types of objects.

In [ ]:
```
import tensorflow as tf
import matplotlib.pyplot as plt

(x_train, y_train), (x_test, y_test) = tf.keras.datasets.cifar10.load_data()

names = ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck']

# Scale pixel values to be between 0 and 1
x_train = x_train / 255.0
x_test = x_test / 255.0
y_test = y_test[:,0]
```

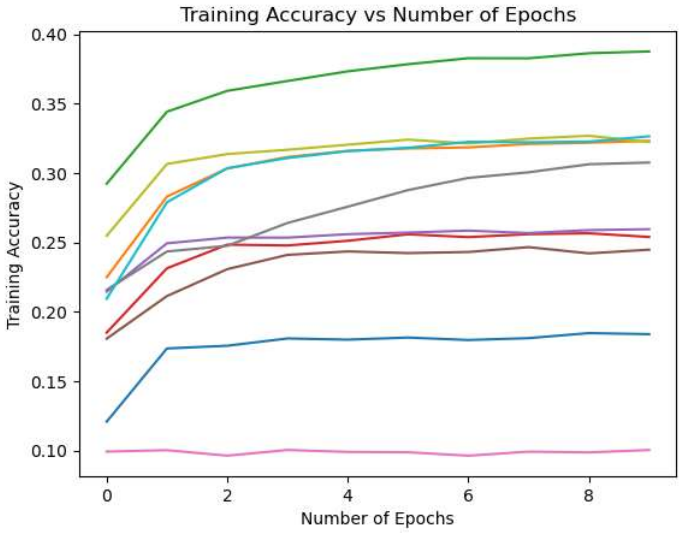
In [ ]:
```
plt.figure(figsize=(14,6))
for i in range(10):
    plt.subplot(2, 5, i+1)
    plt.imshow(x_train[i])
    plt.xticks([])
    plt.yticks([])
    plt.title(names[int(y_train[i])])
plt.show()
```

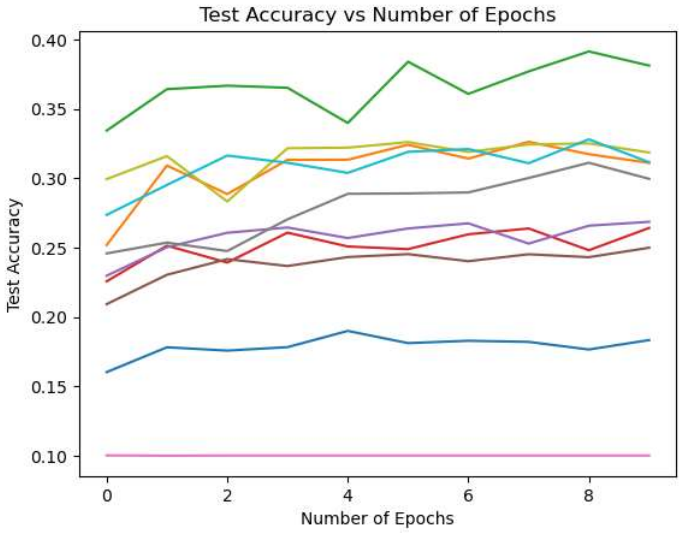
8. [2 marks]: Try using the same model as we used for the MNIST (question 5) task to categorize these images. You will need to change `input_shape=(28, 28)` to `input_shape=(32, 32, 3)`, as these are now colour images of a different size. Use 32 neurons in the hidden layer and train for 10 epochs. Do this 10 times and plot the training accuracy and testing accuracy. How well does the model perform? Try increasing the number of hidden layer neurons (pick a number of neurons that seems reasonable given how much computer processing power you have). How much does the system improve?

We first train the model with one hidden layer and 32 neurons for 10 epochs.

In [ ]:
```
import numpy as np

train_accs = []
test_accs = []

num_models = 10

for i in range(num_models):
    # Create new model
    model = tf.keras.models.Sequential([
              tf.keras.layers.Flatten(input_shape=(32, 32, 3)),   # input is a 28x28 image
              tf.keras.layers.Dense(32, activation='relu'),    # 32 neurons in the middle "hidden" layer
              tf.keras.layers.Dense(10, activation='softmax')  # 10 outputs (one for each category)
            ])

    # Compile model
    model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy']  # in addition to the loss, also compute the categorization accuracy
             )

    # Train each model for 10 epochs
    history = model.fit(x_train, y_train, epochs=10, validation_data=(x_test, y_test), verbose=0);

    # Store training and test errors
    train_accs.append(history.history['accuracy'])

    test_accs.append(history.history['val_accuracy'])

# Plot training errors
plt.figure()
```

```
for i in range(num_models):
    plt.plot(train_accs[i])
plt.title("Training Accuracy vs Number of Epochs")
plt.ylabel("Training Accuracy")
plt.xlabel("Number of Epochs")
plt.show()

# Plot test errors
plt.figure()
for i in range(num_models):
    plt.plot(test_accs[i])
plt.title("Test Accuracy vs Number of Epochs")
plt.ylabel("Test Accuracy")
plt.xlabel("Number of Epochs")
plt.show()
```





In [ ]:
```
# Calculate average final accuracies

total_train = np.sum(train_accs, axis=0)
avg_train_acc = total_train[-1]/10
print(f"The average final training accuracy is {avg_train_acc * 100:.2f}%.")

total_test = np.sum(test_accs, axis=0)
avg_test_acc = total_test[-1]/10
print(f"The average final test accuracy is {avg_test_acc * 100:.2f}%.")
```

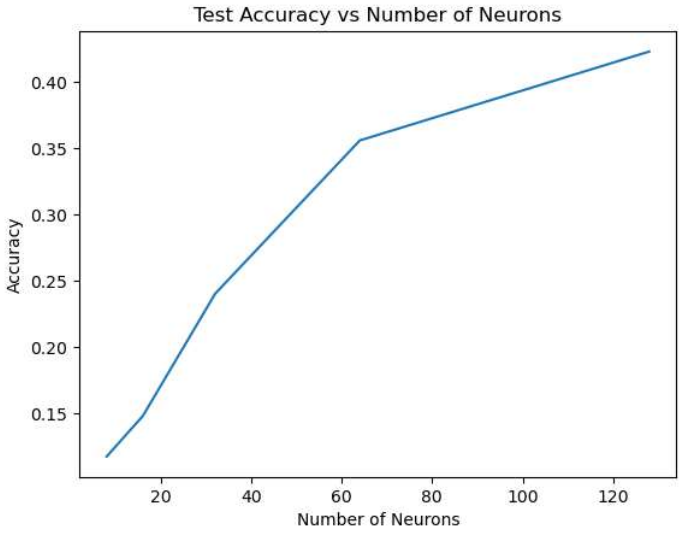
```
The average final training accuracy is 27.10%.
The average final test accuracy is 26.88%.
```

The current model has extremely low training and test accuracy, with the best model achieving just over 35% final test accuracy, with the majority of the models with less than 30% final test accuracy. The average final test accuracy is 26.88%. Clearly, this model performs very poorly. We can try improving the performance of the model by increasing the number of hidden layer neurons.

In [ ]:
```
neurons = [8, 16, 32, 64, 128]

final_train_accs = []
final_test_accs = []
```

```
plt.figure()

for i in range(len(neurons)):
    total_train_acc = 0
    total_test_acc = 0
    # Train 10 models
    for j in range(10):
        # Create new model
        model = tf.keras.models.Sequential([
              tf.keras.layers.Flatten(input_shape=(32, 32, 3)),   # input is a 28x28 image
              tf.keras.layers.Dense(neurons[i], activation='relu'),    # 32 neurons in the middle "hidden" layer
              tf.keras.layers.Dense(10, activation='softmax')  # 10 outputs (one for each category)
            ])

        # Compile model
        model.compile(optimizer='adam',
                loss='sparse_categorical_crossentropy',
                metrics=['accuracy']  # in addition to the loss, also compute the categorization accuracy
                )

        # Train each model for 10 epochs
        history = model.fit(x_train, y_train, epochs=10, validation_data=(x_test, y_test), verbose=0);

        # Store training and test errors
        total_train_acc += history.history['accuracy'][-1]
        total_test_acc += history.history['val_accuracy'][-1]

    # Calculate average accuracy across 10 models
    avg_train_acc = total_train_acc / 10
    avg_test_acc = total_test_acc / 10

    # Plot training accuracies
    final_train_accs.append(avg_train_acc)
    final_test_accs.append(avg_test_acc)

plt.figure()
plt.plot(neurons, final_test_accs)
plt.title("Test Accuracy vs Number of Neurons")
plt.ylabel("Accuracy")
plt.xlabel("Number of Neurons")
plt.show()
```

```
<Figure size 640x480 with 0 Axes>
```



From the graph above, we see that increasing the number of neurons does lead to an increase in test accuracy. However, the test accuracy does not increase significantly enough to say the model is performing well, as the model with 128 neurons in the hidden layer still fails to get above 50% final test accuracy. Other changes will have to be done to the model to make it perform better against the CIFAR-10 data.

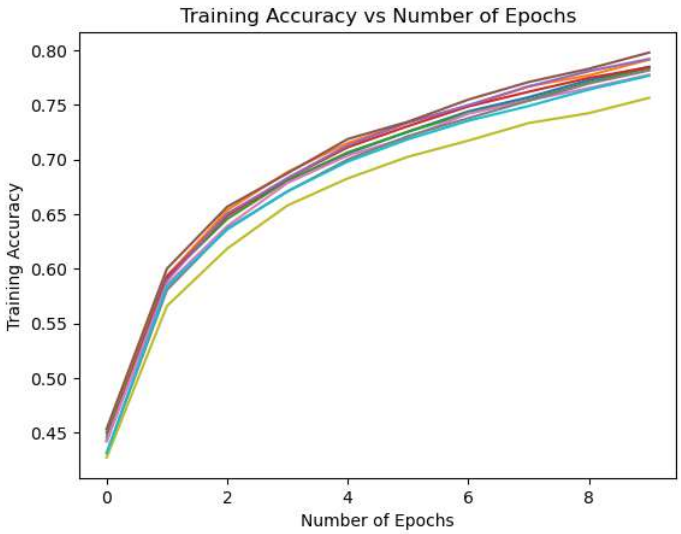
9. [2 marks]: To solve this task, let's try adding more layers. In particular, let's add convolutional layers. Here is a good network structure to start with:

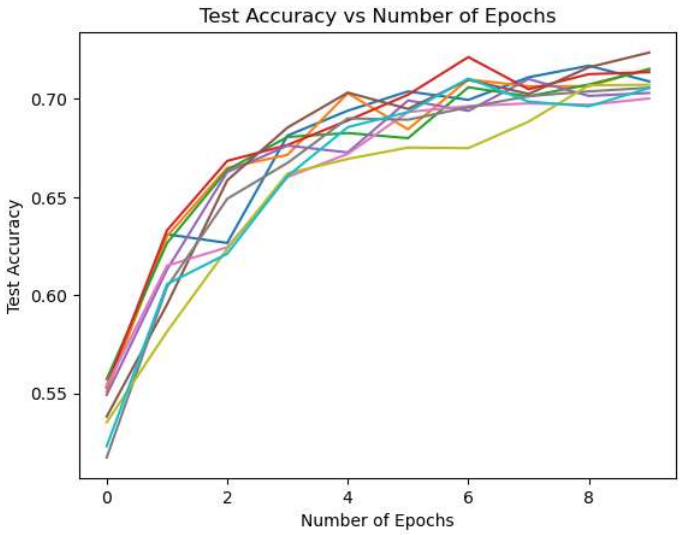
In [ ]:
```
model = tf.keras.models.Sequential([
  tf.keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)),
  tf.keras.layers.MaxPooling2D((2, 2)),
  tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),
  tf.keras.layers.MaxPooling2D((2, 2)),
  tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),
  tf.keras.layers.Flatten(),
  tf.keras.layers.Dense(64, activation='relu'),
  tf.keras.layers.Dense(10, activation='softmax')
])
```

Train this network for 10 epochs, and do this 10 times and plot the training and testing accorcay. How well does the model perform?

In [ ]:
```
import numpy as np
```

```
train_accs = []
test_accs = []

num_models = 10

for i in range(num_models):
    # Create new model
    model = tf.keras.models.Sequential([
              tf.keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)),
              tf.keras.layers.MaxPooling2D((2, 2)),
              tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),
              tf.keras.layers.MaxPooling2D((2, 2)),
              tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),
              tf.keras.layers.Flatten(),
              tf.keras.layers.Dense(64, activation='relu'),
              tf.keras.layers.Dense(10, activation='softmax')
             ])

    # Compile model
    model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy']  # in addition to the loss, also compute the categorization accuracy
             )

    # Train each model for 10 epochs
    history = model.fit(x_train, y_train, epochs=10, validation_data=(x_test, y_test), verbose=0);

    # Store training and test errors
    train_accs.append(history.history['accuracy'])

    test_accs.append(history.history['val_accuracy'])

# Plot training errors
plt.figure()
for i in range(num_models):
    plt.plot(train_accs[i])
plt.title("Training Accuracy vs Number of Epochs")
plt.ylabel("Training Accuracy")
plt.xlabel("Number of Epochs")
plt.show()

# Plot test errors
plt.figure()
for i in range(num_models):
    plt.plot(test_accs[i])
plt.title("Test Accuracy vs Number of Epochs")
plt.ylabel("Test Accuracy")
plt.xlabel("Number of Epochs")
plt.show()
```



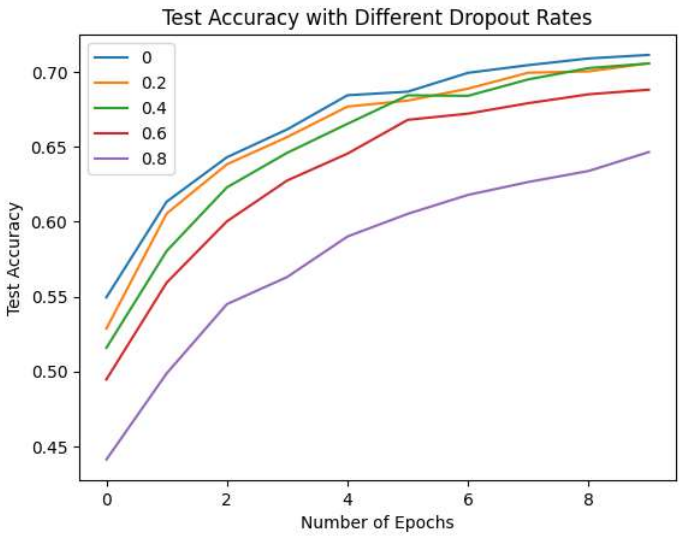


This new network with convolutional layers performs significantly better than the model with only fully connected layers. This is why image classification tasks often use networks involving convolutional layers.

10. [2 marks]: Improve the model as much as you can. Remember that the goal is to increase the testing accuracy, not the training accuracy (after all, if we wanted perfect training accuracy, we could just write code that just memorizes all the training data and uses it as a lookup table). Report what things you tried and how much they helped (or did not help).

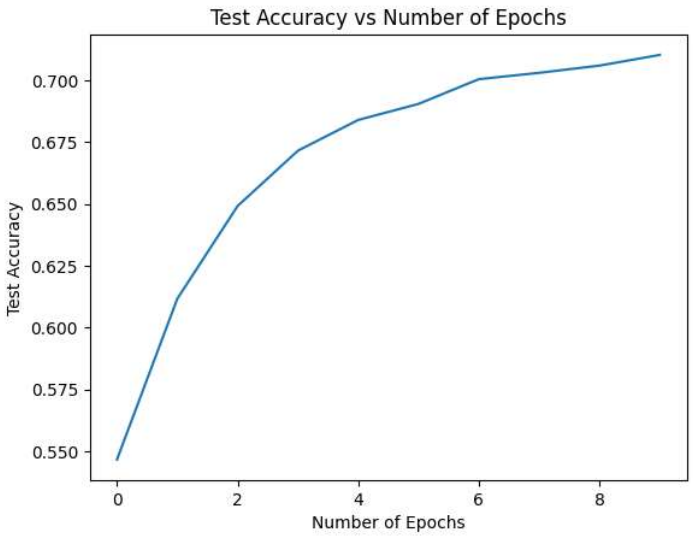
We first try adding a dropout layer and varying the dropout rate.

In [ ]:
```
# We now add dropout and vary the dropout rate
dropouts = [0, 0.2, 0.4, 0.6, 0.8]

for i in range(len(dropouts)):
    total_acc = np.zeros(10)

    # Train 5 models
    for j in range(5):
        # Create new model
        model = tf.keras.models.Sequential([
              tf.keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)),
              tf.keras.layers.MaxPooling2D((2, 2)),
              tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),
              tf.keras.layers.MaxPooling2D((2, 2)),
              tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),
              tf.keras.layers.Flatten(),
              tf.keras.layers.Dense(64, activation='relu'),
              tf.keras.layers.Dropout(dropouts[i]),
              tf.keras.layers.Dense(10, activation='softmax')
            ])

        # Compile model
        model.compile(optimizer='adam',
            loss='sparse_categorical_crossentropy',
            metrics=['accuracy']  # in addition to the loss, also compute the categorization accuracy
            )

        # Train each model for 10 epochs
        history = model.fit(x_train, y_train, epochs=10, validation_data=(x_test, y_test), verbose=0);

        # Store test errors
        total_acc = np.add(total_acc, history.history['val_accuracy'])

    # Calculate average accuracy across 5 models
    avg_acc = np.divide(total_acc, 5)

    # Plot training accuracies
    plt.plot(avg_acc)

plt.title("Test Accuracy with Different Dropout Rates")
plt.legend(dropouts)
plt.ylabel("Test Accuracy")
plt.xlabel("Number of Epochs")
plt.show()
```



Surprisingly, we see that having no dropout leads to the highest test accuracy. We rerun the model with no dropout to get the current test accuracy.

In [ ]:
```
total_acc = np.zeros(10)

# Train 5 models
for i in range(5):
    # Create new model
    model = tf.keras.models.Sequential([
            tf.keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)),
            tf.keras.layers.MaxPooling2D((2, 2)),
            tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),
            tf.keras.layers.MaxPooling2D((2, 2)),
            tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),
            tf.keras.layers.Flatten(),
            tf.keras.layers.Dense(64, activation='relu'),
            tf.keras.layers.Dense(10, activation='softmax')
        ])

    # Compile model
    model.compile(optimizer='adam',
        loss='sparse_categorical_crossentropy',
        metrics=['accuracy']  # in addition to the loss, also compute the categorization accuracy
        )

    # Train each model for 10 epochs
    history = model.fit(x_train, y_train, epochs=10, validation_data=(x_test, y_test), verbose=0);

    # Store test errors
    total_acc = np.add(total_acc, history.history['val_accuracy'])

# Calculate average accuracy across 5 models
avg_acc = np.divide(total_acc, 5)

# Plot training accuracies
plt.plot(avg_acc)

plt.title("Test Accuracy vs Number of Epochs")
plt.ylabel("Test Accuracy")
plt.xlabel("Number of Epochs")
plt.show()

print(f"The final test accuracy of the model is {avg_acc[-1]}.")
```




```
The final test accuracy of the model is 0.7102999985218048.
```

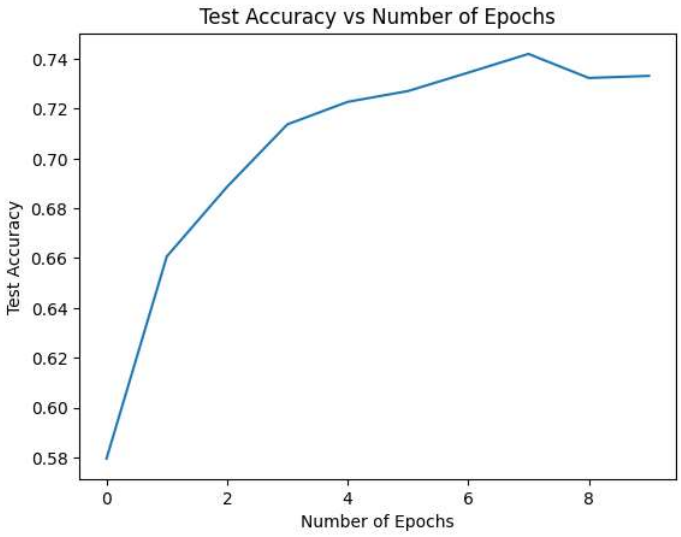
The final test accuracy of the model with no dropout is 71.03%. We will attempt to improve the test accuracy above 71.03%. We try adding an additional convolutional layer.

In [ ]:

```
import numpy as np

total_acc = np.zeros(10)

num_models = 5
for i in range(num_models):
    # Create new model
    model = tf.keras.models.Sequential([
              tf.keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)),
              tf.keras.layers.MaxPooling2D((2, 2)),

              tf.keras.layers.Conv2D(64, (3, 3), activation='relu', padding='same'),
              tf.keras.layers.MaxPooling2D((2, 2)),

              tf.keras.layers.Conv2D(128, (3, 3), activation='relu', padding='same'),
              tf.keras.layers.MaxPooling2D((2, 2)),

              tf.keras.layers.Conv2D(128, (3, 3), activation='relu', padding='same'),
              tf.keras.layers.MaxPooling2D((2, 2)),

              tf.keras.layers.Flatten(),
              tf.keras.layers.Dense(64, activation='relu'),
              tf.keras.layers.Dense(10, activation='softmax')
            ])


    # Compile model
    model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy']  # in addition to the loss, also compute the categorization accuracy
             )

    # Train each model for 10 epochs
    history = model.fit(x_train, y_train, epochs=10, validation_data=(x_test, y_test));

    # Store training and test errors
    total_acc = np.add(total_acc, history.history['val_accuracy'])

# Calculate average accuracy across 5 models
avg_acc = np.divide(total_acc, 5)

# Plot training accuracies
plt.plot(avg_acc)

plt.title("Test Accuracy vs Number of Epochs")
plt.ylabel("Test Accuracy")
plt.xlabel("Number of Epochs")
plt.show()

print(f"The final test accuracy of the model is {avg_acc[-1]}.")
```

```
Epoch 1/10
1563/1563 [==============================] - 27s 16ms/step - loss: 1.4729 - accuracy: 0.4581 - val_loss: 1.1353 - val_accuracy: 0.5929
Epoch 2/10
1563/1563 [==============================] - 24s 16ms/step - loss: 1.0405 - accuracy: 0.6320 - val_loss: 0.9251 - val_accuracy: 0.6768
Epoch 3/10
1563/1563 [==============================] - 24s 15ms/step - loss: 0.8566 - accuracy: 0.7005 - val_loss: 0.8833 - val_accuracy: 0.6970
Epoch 4/10
1563/1563 [==============================] - 24s 15ms/step - loss: 0.7419 - accuracy: 0.7414 - val_loss: 0.8437 - val_accuracy: 0.7094
Epoch 5/10
1563/1563 [==============================] - 24s 15ms/step - loss: 0.6492 - accuracy: 0.7725 - val_loss: 0.8195 - val_accuracy: 0.7252
Epoch 6/10
1563/1563 [==============================] - 24s 15ms/step - loss: 0.5675 - accuracy: 0.8023 - val_loss: 0.8076 - val_accuracy: 0.7340
Epoch 7/10
1563/1563 [==============================] - 24s 15ms/step - loss: 0.4999 - accuracy: 0.8227 - val_loss: 0.8141 - val_accuracy: 0.7320
Epoch 8/10
1563/1563 [==============================] - 24s 15ms/step - loss: 0.4357 - accuracy: 0.8469 - val_loss: 0.8397 - val_accuracy: 0.7361
Epoch 9/10
1563/1563 [==============================] - 23s 15ms/step - loss: 0.3867 - accuracy: 0.8630 - val_loss: 0.9233 - val_accuracy: 0.7292
Epoch 10/10
1563/1563 [==============================] - 23s 15ms/step - loss: 0.3363 - accuracy: 0.8816 - val_loss: 0.9265 - val_accuracy: 0.7439
Epoch 1/10
1563/1563 [==============================] - 26s 16ms/step - loss: 1.4504 - accuracy: 0.4719 - val_loss: 1.2113 - val_accuracy: 0.5740
Epoch 2/10
1563/1563 [==============================] - 25s 16ms/step - loss: 1.0214 - accuracy: 0.6397 - val_loss: 0.9615 - val_accuracy: 0.6637
Epoch 3/10
1563/1563 [==============================] - 25s 16ms/step - loss: 0.8401 - accuracy: 0.7048 - val_loss: 0.8937 - val_accuracy: 0.6929
Epoch 4/10
1563/1563 [==============================] - 23s 15ms/step - loss: 0.7144 - accuracy: 0.7491 - val_loss: 0.8045 - val_accuracy: 0.7266
Epoch 5/10
1563/1563 [==============================] - 23s 15ms/step - loss: 0.6239 - accuracy: 0.7824 - val_loss: 0.7601 - val_accuracy: 0.7361
Epoch 6/10
1563/1563 [==============================] - 23s 15ms/step - loss: 0.5488 - accuracy: 0.8079 - val_loss: 0.8134 - val_accuracy: 0.7296
Epoch 7/10
1563/1563 [==============================] - 24s 15ms/step - loss: 0.4877 - accuracy: 0.8285 - val_loss: 0.7895 - val_accuracy: 0.7473
Epoch 8/10
1563/1563 [==============================] - 24s 15ms/step - loss: 0.4192 - accuracy: 0.8522 - val_loss: 0.8014 - val_accuracy: 0.7473
Epoch 9/10
1563/1563 [==============================] - 24s 15ms/step - loss: 0.3762 - accuracy: 0.8666 - val_loss: 0.8338 - val_accuracy: 0.7472
Epoch 10/10
1563/1563 [==============================] - 23s 15ms/step - loss: 0.3229 - accuracy: 0.8865 - val_loss: 1.0059 - val_accuracy: 0.7259
Epoch 1/10
1563/1563 [==============================] - 25s 15ms/step - loss: 1.5152 - accuracy: 0.4452 - val_loss: 1.1791 - val_accuracy: 0.5787
Epoch 2/10
1563/1563 [==============================] - 23s 15ms/step - loss: 1.0658 - accuracy: 0.6243 - val_loss: 0.9786 - val_accuracy: 0.6553
Epoch 3/10
1563/1563 [==============================] - 23s 15ms/step - loss: 0.8773 - accuracy: 0.6951 - val_loss: 0.9079 - val_accuracy: 0.6878
Epoch 4/10
1563/1563 [==============================] - 23s 15ms/step - loss: 0.7550 - accuracy: 0.7356 - val_loss: 0.8631 - val_accuracy: 0.6997
Epoch 5/10
1563/1563 [==============================] - 23s 15ms/step - loss: 0.6629 - accuracy: 0.7688 - val_loss: 0.8341 - val_accuracy: 0.7159
Epoch 6/10
1563/1563 [==============================] - 23s 15ms/step - loss: 0.5915 - accuracy: 0.7922 - val_loss: 0.8695 - val_accuracy: 0.7109
Epoch 7/10
1563/1563 [==============================] - 24s 15ms/step - loss: 0.5201 - accuracy: 0.8156 - val_loss: 0.8171 - val_accuracy: 0.7298
Epoch 8/10
1563/1563 [==============================] - 23s 15ms/step - loss: 0.4673 - accuracy: 0.8356 - val_loss: 0.8509 - val_accuracy: 0.7299
Epoch 9/10
1563/1563 [==============================] - 23s 15ms/step - loss: 0.4135 - accuracy: 0.8547 - val_loss: 0.9538 - val_accuracy: 0.7137
Epoch 10/10
1563/1563 [==============================] - 23s 15ms/step - loss: 0.3659 - accuracy: 0.8703 - val_loss: 0.9043 - val_accuracy: 0.7329
Epoch 1/10
1563/1563 [==============================] - 27s 16ms/step - loss: 1.4729 - accuracy: 0.4598 - val_loss: 1.1706 - val_accuracy: 0.5835
Epoch 2/10
1563/1563 [==============================] - 24s 16ms/step - loss: 1.0446 - accuracy: 0.6296 - val_loss: 0.9584 - val_accuracy: 0.6637
Epoch 3/10
1563/1563 [==============================] - 24s 15ms/step - loss: 0.8734 - accuracy: 0.6925 - val_loss: 0.9086 - val_accuracy: 0.6879
Epoch 4/10
1563/1563 [==============================] - 24s 15ms/step - loss: 0.7582 - accuracy: 0.7332 - val_loss: 0.8434 - val_accuracy: 0.7122
Epoch 5/10
1563/1563 [==============================] - 24s 16ms/step - loss: 0.6662 - accuracy: 0.7669 - val_loss: 0.8222 - val_accuracy: 0.7190
Epoch 6/10
1563/1563 [==============================] - 24s 16ms/step - loss: 0.5927 - accuracy: 0.7913 - val_loss: 0.8282 - val_accuracy: 0.7232
Epoch 7/10
1563/1563 [==============================] - 24s 16ms/step - loss: 0.5253 - accuracy: 0.8163 - val_loss: 0.8751 - val_accuracy: 0.7247
Epoch 8/10
1563/1563 [==============================] - 24s 16ms/step - loss: 0.4705 - accuracy: 0.8330 - val_loss: 0.8123 - val_accuracy: 0.7448
Epoch 9/10
1563/1563 [==============================] - 24s 15ms/step - loss: 0.4166 - accuracy: 0.8521 - val_loss: 0.8968 - val_accuracy: 0.7291
Epoch 10/10
1563/1563 [==============================] - 25s 16ms/step - loss: 0.3720 - accuracy: 0.8690 - val_loss: 0.9503 - val_accuracy: 0.7276
Epoch 1/10
1563/1563 [==============================] - 27s 17ms/step - loss: 1.4918 - accuracy: 0.4535 - val_loss: 1.2001 - val_accuracy: 0.5684
Epoch 2/10
1563/1563 [==============================] - 24s 16ms/step - loss: 1.0548 - accuracy: 0.6256 - val_loss: 1.0301 - val_accuracy: 0.6439
Epoch 3/10
1563/1563 [==============================] - 24s 15ms/step - loss: 0.8679 - accuracy: 0.6962 - val_loss: 0.9318 - val_accuracy: 0.6776
Epoch 4/10
1563/1563 [==============================] - 24s 15ms/step - loss: 0.7417 - accuracy: 0.7416 - val_loss: 0.8269 - val_accuracy: 0.7207
Epoch 5/10
1563/1563 [==============================] - 24s 15ms/step - loss: 0.6500 - accuracy: 0.7715 - val_loss: 0.8191 - val_accuracy: 0.7176
Epoch 6/10
1563/1563 [==============================] - 24s 15ms/step - loss: 0.5723 - accuracy: 0.8005 - val_loss: 0.7966 - val_accuracy: 0.7378
Epoch 7/10
1563/1563 [==============================] - 24s 15ms/step - loss: 0.5094 - accuracy: 0.8198 - val_loss: 0.8044 - val_accuracy: 0.7387
```

```
Epoch 8/10
1563/1563 [==============================] - 24s 15ms/step - loss: 0.4487 - accuracy: 0.8411 - val_loss: 0.8005 - val_accuracy: 0.7520
Epoch 9/10
1563/1563 [==============================] - 24s 15ms/step - loss: 0.4014 - accuracy: 0.8577 - val_loss: 0.8705 - val_accuracy: 0.7426
Epoch 10/10
1563/1563 [==============================] - 24s 15ms/step - loss: 0.3536 - accuracy: 0.8746 - val_loss: 0.9248 - val_accuracy: 0.7357
```



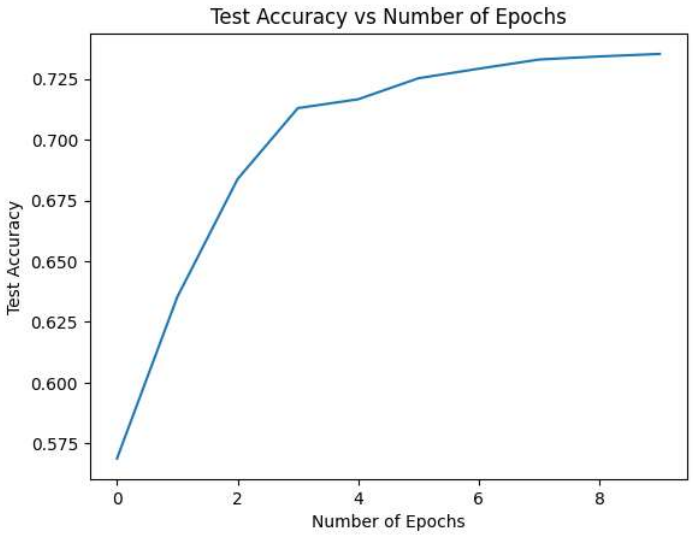

```
The final test accuracy of the model is 0.7332000017166138.
```

As seen above, adding an additional convolutional layer increased the final test accuracy by a small margin (approximately 2.29%). However, this led to a 48% increase in processing time. In some cases, this tradeoff may not be beneficial, but since we're only trying to maximumize final test accuracy, we will continue with this model. We can then try adding additional hidden fully-connected layers and increasing the number of neurons in each hidden layer to 128 from 64.

In [ ]:
```python
import numpy as np

total_acc = np.zeros(10)

num_models = 5
for i in range(num_models):
    # Create new model
    model = tf.keras.models.Sequential([
              tf.keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)),
              tf.keras.layers.MaxPooling2D((2, 2)),

              tf.keras.layers.Conv2D(64, (3, 3), activation='relu', padding='same'),
              tf.keras.layers.MaxPooling2D((2, 2)),

              tf.keras.layers.Conv2D(128, (3, 3), activation='relu', padding='same'),
              tf.keras.layers.MaxPooling2D((2, 2)),

              tf.keras.layers.Conv2D(128, (3, 3), activation='relu', padding='same'),
              tf.keras.layers.MaxPooling2D((2, 2)),

              tf.keras.layers.Flatten(),
              tf.keras.layers.Dense(128, activation='relu'),
              tf.keras.layers.Dense(128, activation='relu'),
              tf.keras.layers.Dense(10, activation='softmax')
            ])


    # Compile model
    model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy']  # in addition to the loss, also compute the categorization accuracy
             )

    # Train each model for 10 epochs
    history = model.fit(x_train, y_train, epochs=10, validation_data=(x_test, y_test));

    # Store training and test errors
    total_acc = np.add(total_acc, history.history['val_accuracy'])

# Calculate average accuracy across 5 models
avg_acc = np.divide(total_acc, 5)

# Plot training accuracies
plt.plot(avg_acc)

plt.title("Test Accuracy vs Number of Epochs")
plt.ylabel("Test Accuracy")
plt.xlabel("Number of Epochs")
plt.show()

print(f"The final test accuracy of the model is {avg_acc[-1]}.")
```

```
Epoch 1/10
1563/1563 [==============================] - 25s 15ms/step - loss: 1.5041 - accuracy: 0.4436 - val_loss: 1.1703 - val_accuracy: 0.5737
Epoch 2/10
1563/1563 [==============================] - 24s 15ms/step - loss: 1.0497 - accuracy: 0.6277 - val_loss: 0.9782 - val_accuracy: 0.6544
Epoch 3/10
1563/1563 [==============================] - 24s 15ms/step - loss: 0.8543 - accuracy: 0.7016 - val_loss: 0.8703 - val_accuracy: 0.6991
Epoch 4/10
1563/1563 [==============================] - 24s 15ms/step - loss: 0.7254 - accuracy: 0.7477 - val_loss: 0.8035 - val_accuracy: 0.7272
Epoch 5/10
1563/1563 [==============================] - 24s 15ms/step - loss: 0.6426 - accuracy: 0.7765 - val_loss: 0.8457 - val_accuracy: 0.7217
Epoch 6/10
1563/1563 [==============================] - 24s 15ms/step - loss: 0.5727 - accuracy: 0.8001 - val_loss: 0.7956 - val_accuracy: 0.7342
Epoch 7/10
1563/1563 [==============================] - 24s 15ms/step - loss: 0.5131 - accuracy: 0.8203 - val_loss: 0.7911 - val_accuracy: 0.7365
Epoch 8/10
1563/1563 [==============================] - 25s 16ms/step - loss: 0.4565 - accuracy: 0.8417 - val_loss: 0.8401 - val_accuracy: 0.7398
Epoch 9/10
1563/1563 [==============================] - 25s 16ms/step - loss: 0.4114 - accuracy: 0.8555 - val_loss: 0.8814 - val_accuracy: 0.7379
Epoch 10/10
1563/1563 [==============================] - 25s 16ms/step - loss: 0.3664 - accuracy: 0.8708 - val_loss: 0.8416 - val_accuracy: 0.7441
Epoch 1/10
1563/1563 [==============================] - 27s 16ms/step - loss: 1.5359 - accuracy: 0.4298 - val_loss: 1.2147 - val_accuracy: 0.5702
Epoch 2/10
1563/1563 [==============================] - 24s 16ms/step - loss: 1.0892 - accuracy: 0.6130 - val_loss: 1.1848 - val_accuracy: 0.5837
Epoch 3/10
1563/1563 [==============================] - 24s 15ms/step - loss: 0.8924 - accuracy: 0.6852 - val_loss: 0.8853 - val_accuracy: 0.6912
Epoch 4/10
1563/1563 [==============================] - 24s 16ms/step - loss: 0.7687 - accuracy: 0.7319 - val_loss: 0.8549 - val_accuracy: 0.7021
Epoch 5/10
1563/1563 [==============================] - 25s 16ms/step - loss: 0.6728 - accuracy: 0.7654 - val_loss: 0.8310 - val_accuracy: 0.7161
Epoch 6/10
1563/1563 [==============================] - 25s 16ms/step - loss: 0.5904 - accuracy: 0.7931 - val_loss: 0.8210 - val_accuracy: 0.7290
Epoch 7/10
1563/1563 [==============================] - 24s 15ms/step - loss: 0.5216 - accuracy: 0.8184 - val_loss: 0.7968 - val_accuracy: 0.7346
Epoch 8/10
1563/1563 [==============================] - 23s 15ms/step - loss: 0.4654 - accuracy: 0.8367 - val_loss: 0.8873 - val_accuracy: 0.7255
Epoch 9/10
1563/1563 [==============================] - 23s 15ms/step - loss: 0.4173 - accuracy: 0.8527 - val_loss: 0.8808 - val_accuracy: 0.7380
Epoch 10/10
1563/1563 [==============================] - 23s 15ms/step - loss: 0.3763 - accuracy: 0.8681 - val_loss: 0.9122 - val_accuracy: 0.7271
Epoch 1/10
1563/1563 [==============================] - 26s 16ms/step - loss: 1.5293 - accuracy: 0.4337 - val_loss: 1.2010 - val_accuracy: 0.5674
Epoch 2/10
1563/1563 [==============================] - 24s 16ms/step - loss: 1.0764 - accuracy: 0.6171 - val_loss: 1.0802 - val_accuracy: 0.6297
Epoch 3/10
1563/1563 [==============================] - 24s 15ms/step - loss: 0.8947 - accuracy: 0.6854 - val_loss: 0.9457 - val_accuracy: 0.6761
Epoch 4/10
1563/1563 [==============================] - 25s 16ms/step - loss: 0.7727 - accuracy: 0.7304 - val_loss: 0.8679 - val_accuracy: 0.7053
Epoch 5/10
1563/1563 [==============================] - 24s 16ms/step - loss: 0.6862 - accuracy: 0.7601 - val_loss: 0.8500 - val_accuracy: 0.7167
Epoch 6/10
1563/1563 [==============================] - 24s 15ms/step - loss: 0.6080 - accuracy: 0.7872 - val_loss: 0.8138 - val_accuracy: 0.7322
Epoch 7/10
1563/1563 [==============================] - 24s 15ms/step - loss: 0.5507 - accuracy: 0.8072 - val_loss: 0.8194 - val_accuracy: 0.7337
Epoch 8/10
1563/1563 [==============================] - 24s 15ms/step - loss: 0.4931 - accuracy: 0.8295 - val_loss: 0.8283 - val_accuracy: 0.7362
Epoch 9/10
1563/1563 [==============================] - 26s 17ms/step - loss: 0.4388 - accuracy: 0.8452 - val_loss: 0.9269 - val_accuracy: 0.7255
Epoch 10/10
1563/1563 [==============================] - 25s 16ms/step - loss: 0.4025 - accuracy: 0.8598 - val_loss: 0.8996 - val_accuracy: 0.7356
Epoch 1/10
1563/1563 [==============================] - 27s 16ms/step - loss: 1.5350 - accuracy: 0.4294 - val_loss: 1.1797 - val_accuracy: 0.5710
Epoch 2/10
1563/1563 [==============================] - 26s 16ms/step - loss: 1.0856 - accuracy: 0.6123 - val_loss: 1.0208 - val_accuracy: 0.6385
Epoch 3/10
1563/1563 [==============================] - 25s 16ms/step - loss: 0.8908 - accuracy: 0.6855 - val_loss: 0.9177 - val_accuracy: 0.6839
Epoch 4/10
1563/1563 [==============================] - 26s 17ms/step - loss: 0.7601 - accuracy: 0.7374 - val_loss: 0.8589 - val_accuracy: 0.7025
Epoch 5/10
1563/1563 [==============================] - 25s 16ms/step - loss: 0.6763 - accuracy: 0.7652 - val_loss: 0.8455 - val_accuracy: 0.7141
Epoch 6/10
1563/1563 [==============================] - 25s 16ms/step - loss: 0.6062 - accuracy: 0.7890 - val_loss: 0.9023 - val_accuracy: 0.7015
Epoch 7/10
1563/1563 [==============================] - 25s 16ms/step - loss: 0.5376 - accuracy: 0.8143 - val_loss: 0.8632 - val_accuracy: 0.7200
Epoch 8/10
1563/1563 [==============================] - 25s 16ms/step - loss: 0.4891 - accuracy: 0.8300 - val_loss: 0.8662 - val_accuracy: 0.7287
Epoch 9/10
1563/1563 [==============================] - 24s 15ms/step - loss: 0.4400 - accuracy: 0.8456 - val_loss: 0.8927 - val_accuracy: 0.7319
Epoch 10/10
1563/1563 [==============================] - 25s 16ms/step - loss: 0.3953 - accuracy: 0.8615 - val_loss: 0.9089 - val_accuracy: 0.7354
Epoch 1/10
1563/1563 [==============================] - 25s 16ms/step - loss: 1.5232 - accuracy: 0.4379 - val_loss: 1.2197 - val_accuracy: 0.5615
Epoch 2/10
1563/1563 [==============================] - 24s 15ms/step - loss: 1.0527 - accuracy: 0.6259 - val_loss: 0.9434 - val_accuracy: 0.6699
Epoch 3/10
1563/1563 [==============================] - 24s 15ms/step - loss: 0.8666 - accuracy: 0.6975 - val_loss: 0.9494 - val_accuracy: 0.6685
Epoch 4/10
1563/1563 [==============================] - 24s 15ms/step - loss: 0.7494 - accuracy: 0.7388 - val_loss: 0.7879 - val_accuracy: 0.7278
Epoch 5/10
1563/1563 [==============================] - 24s 15ms/step - loss: 0.6576 - accuracy: 0.7698 - val_loss: 0.8264 - val_accuracy: 0.7145
Epoch 6/10
1563/1563 [==============================] - 24s 15ms/step - loss: 0.5827 - accuracy: 0.7962 - val_loss: 0.7974 - val_accuracy: 0.7295
Epoch 7/10
1563/1563 [==============================] - 24s 16ms/step - loss: 0.5224 - accuracy: 0.8175 - val_loss: 0.8631 - val_accuracy: 0.7212
```

```
Epoch 8/10
1563/1563 [==============================] - 24s 15ms/step - loss: 0.4654 - accuracy: 0.8365 - val_loss: 0.8132 - val_accuracy: 0.7347
Epoch 9/10
1563/1563 [==============================] - 25s 16ms/step - loss: 0.4142 - accuracy: 0.8546 - val_loss: 0.8412 - val_accuracy: 0.7379
Epoch 10/10
1563/1563 [==============================] - 25s 16ms/step - loss: 0.3718 - accuracy: 0.8702 - val_loss: 0.9304 - val_accuracy: 0.7342
```




```
The final test accuracy of the model is 0.7352800011634827.
```

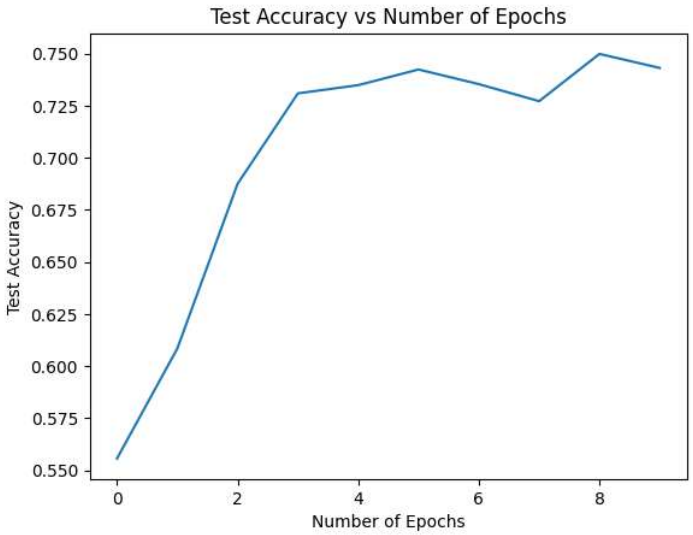
Adding additional hidden fully-connected layers led to an increase of only 0.21% in the final test accuracy. This was surprising, as there was one additional fully-connected layer added and all the fully-connected layers had 128 neurons, which was double the number of neurons originally. Next, we try adding batch normalization, to normalize the data.

In [ ]:
```python
# Add BatchNormalization

import numpy as np

total_acc = np.zeros(10)

num_models = 5
for i in range(num_models):
    # Create new model
    model = tf.keras.models.Sequential([
              tf.keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)),
              tf.keras.layers.BatchNormalization(),
              tf.keras.layers.MaxPooling2D((2, 2)),

              tf.keras.layers.Conv2D(64, (3, 3), activation='relu', padding='same'),
              tf.keras.layers.BatchNormalization(),
              tf.keras.layers.MaxPooling2D((2, 2)),

              tf.keras.layers.Conv2D(128, (3, 3), activation='relu', padding='same'),
              tf.keras.layers.BatchNormalization(),
              tf.keras.layers.MaxPooling2D((2, 2)),

              tf.keras.layers.Conv2D(128, (3, 3), activation='relu', padding='same'),
              tf.keras.layers.BatchNormalization(),
              tf.keras.layers.MaxPooling2D((2, 2)),

              tf.keras.layers.Flatten(),

              tf.keras.layers.Dense(128, activation='relu'),
              tf.keras.layers.Dense(128, activation='relu'),
              tf.keras.layers.Dense(10, activation='softmax')
            ])


    # Compile model
    model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy']  # in addition to the loss, also compute the categorization accuracy
             )

    # Train each model for 10 epochs
    history = model.fit(x_train, y_train, epochs=10, validation_data=(x_test, y_test));

    # Store training and test errors
    total_acc = np.add(total_acc, history.history['val_accuracy'])

# Calculate average accuracy across 5 models
avg_acc = np.divide(total_acc, 5)

# Plot training accuracies
plt.plot(avg_acc)

plt.title("Test Accuracy vs Number of Epochs")
plt.ylabel("Test Accuracy")
```

```
plt.xlabel("Number of Epochs")
plt.show()

print(f"The final test accuracy of the model is {avg_acc[-1]}.")
```

```
Epoch 1/10
1563/1563 [==============================] - 35s 22ms/step - loss: 1.2545 - accuracy: 0.5489 - val_loss: 1.5178 - val_accuracy: 0.4656
Epoch 2/10
1563/1563 [==============================] - 34s 22ms/step - loss: 0.8478 - accuracy: 0.7040 - val_loss: 1.4170 - val_accuracy: 0.5605
Epoch 3/10
1563/1563 [==============================] - 35s 22ms/step - loss: 0.6808 - accuracy: 0.7628 - val_loss: 0.9987 - val_accuracy: 0.6621
Epoch 4/10
1563/1563 [==============================] - 34s 22ms/step - loss: 0.5739 - accuracy: 0.8000 - val_loss: 0.8567 - val_accuracy: 0.7165
Epoch 5/10
1563/1563 [==============================] - 34s 22ms/step - loss: 0.4853 - accuracy: 0.8303 - val_loss: 0.8368 - val_accuracy: 0.7318
Epoch 6/10
1563/1563 [==============================] - 34s 22ms/step - loss: 0.4106 - accuracy: 0.8569 - val_loss: 0.8090 - val_accuracy: 0.7469
Epoch 7/10
1563/1563 [==============================] - 34s 22ms/step - loss: 0.3453 - accuracy: 0.8791 - val_loss: 0.7723 - val_accuracy: 0.7599
Epoch 8/10
1563/1563 [==============================] - 34s 22ms/step - loss: 0.2898 - accuracy: 0.8983 - val_loss: 0.9924 - val_accuracy: 0.7161
Epoch 9/10
1563/1563 [==============================] - 34s 22ms/step - loss: 0.2551 - accuracy: 0.9103 - val_loss: 0.8794 - val_accuracy: 0.7565
Epoch 10/10
1563/1563 [==============================] - 33s 21ms/step - loss: 0.2170 - accuracy: 0.9237 - val_loss: 1.0453 - val_accuracy: 0.7251
Epoch 1/10
1563/1563 [==============================] - 34s 21ms/step - loss: 1.2305 - accuracy: 0.5616 - val_loss: 1.1104 - val_accuracy: 0.6124
Epoch 2/10
1563/1563 [==============================] - 32s 21ms/step - loss: 0.8546 - accuracy: 0.6999 - val_loss: 1.1909 - val_accuracy: 0.6004
Epoch 3/10
1563/1563 [==============================] - 32s 21ms/step - loss: 0.6882 - accuracy: 0.7601 - val_loss: 0.8235 - val_accuracy: 0.7290
Epoch 4/10
1563/1563 [==============================] - 32s 21ms/step - loss: 0.5766 - accuracy: 0.7999 - val_loss: 0.7402 - val_accuracy: 0.7512
Epoch 5/10
1563/1563 [==============================] - 32s 21ms/step - loss: 0.4915 - accuracy: 0.8303 - val_loss: 0.8840 - val_accuracy: 0.7223
Epoch 6/10
1563/1563 [==============================] - 33s 21ms/step - loss: 0.4125 - accuracy: 0.8545 - val_loss: 0.8199 - val_accuracy: 0.7447
Epoch 7/10
1563/1563 [==============================] - 33s 21ms/step - loss: 0.3457 - accuracy: 0.8790 - val_loss: 0.8705 - val_accuracy: 0.7435
Epoch 8/10
1563/1563 [==============================] - 32s 20ms/step - loss: 0.2944 - accuracy: 0.8952 - val_loss: 0.8702 - val_accuracy: 0.7478
Epoch 9/10
1563/1563 [==============================] - 32s 20ms/step - loss: 0.2550 - accuracy: 0.9111 - val_loss: 1.0441 - val_accuracy: 0.7239
Epoch 10/10
1563/1563 [==============================] - 32s 21ms/step - loss: 0.2176 - accuracy: 0.9229 - val_loss: 0.9853 - val_accuracy: 0.7427
Epoch 1/10
1563/1563 [==============================] - 34s 21ms/step - loss: 1.2411 - accuracy: 0.5555 - val_loss: 1.2473 - val_accuracy: 0.5804
Epoch 2/10
1563/1563 [==============================] - 33s 21ms/step - loss: 0.8439 - accuracy: 0.7014 - val_loss: 0.9084 - val_accuracy: 0.6877
Epoch 3/10
1563/1563 [==============================] - 33s 21ms/step - loss: 0.6843 - accuracy: 0.7595 - val_loss: 0.8089 - val_accuracy: 0.7209
Epoch 4/10
1563/1563 [==============================] - 33s 21ms/step - loss: 0.5687 - accuracy: 0.8036 - val_loss: 0.8483 - val_accuracy: 0.7214
Epoch 5/10
1563/1563 [==============================] - 33s 21ms/step - loss: 0.4762 - accuracy: 0.8327 - val_loss: 0.8287 - val_accuracy: 0.7292
Epoch 6/10
1563/1563 [==============================] - 34s 22ms/step - loss: 0.4017 - accuracy: 0.8594 - val_loss: 0.8817 - val_accuracy: 0.7322
Epoch 7/10
1563/1563 [==============================] - 33s 21ms/step - loss: 0.3417 - accuracy: 0.8806 - val_loss: 0.9115 - val_accuracy: 0.7286
Epoch 8/10
1563/1563 [==============================] - 33s 21ms/step - loss: 0.2940 - accuracy: 0.8976 - val_loss: 0.9820 - val_accuracy: 0.7287
Epoch 9/10
1563/1563 [==============================] - 32s 21ms/step - loss: 0.2459 - accuracy: 0.9133 - val_loss: 0.8840 - val_accuracy: 0.7653
Epoch 10/10
1563/1563 [==============================] - 32s 21ms/step - loss: 0.2112 - accuracy: 0.9269 - val_loss: 1.0521 - val_accuracy: 0.7352
Epoch 1/10
1563/1563 [==============================] - 34s 21ms/step - loss: 1.2430 - accuracy: 0.5557 - val_loss: 1.2625 - val_accuracy: 0.5571
Epoch 2/10
1563/1563 [==============================] - 33s 21ms/step - loss: 0.8404 - accuracy: 0.7073 - val_loss: 1.3189 - val_accuracy: 0.5860
Epoch 3/10
1563/1563 [==============================] - 33s 21ms/step - loss: 0.6817 - accuracy: 0.7624 - val_loss: 0.8543 - val_accuracy: 0.7036
Epoch 4/10
1563/1563 [==============================] - 33s 21ms/step - loss: 0.5720 - accuracy: 0.7999 - val_loss: 0.7635 - val_accuracy: 0.7423
Epoch 5/10
1563/1563 [==============================] - 34s 22ms/step - loss: 0.4816 - accuracy: 0.8312 - val_loss: 0.7739 - val_accuracy: 0.7404
Epoch 6/10
1563/1563 [==============================] - 34s 22ms/step - loss: 0.4108 - accuracy: 0.8582 - val_loss: 0.8249 - val_accuracy: 0.7416
Epoch 7/10
1563/1563 [==============================] - 34s 22ms/step - loss: 0.3452 - accuracy: 0.8780 - val_loss: 0.9823 - val_accuracy: 0.7129
Epoch 8/10
1563/1563 [==============================] - 35s 23ms/step - loss: 0.2957 - accuracy: 0.8955 - val_loss: 0.9436 - val_accuracy: 0.7316
Epoch 9/10
1563/1563 [==============================] - 36s 23ms/step - loss: 0.2537 - accuracy: 0.9119 - val_loss: 0.8746 - val_accuracy: 0.7584
Epoch 10/10
1563/1563 [==============================] - 35s 23ms/step - loss: 0.2152 - accuracy: 0.9258 - val_loss: 0.8707 - val_accuracy: 0.7672
Epoch 1/10
1563/1563 [==============================] - 36s 22ms/step - loss: 1.2415 - accuracy: 0.5549 - val_loss: 1.2843 - val_accuracy: 0.5625
Epoch 2/10
1563/1563 [==============================] - 34s 22ms/step - loss: 0.8574 - accuracy: 0.7001 - val_loss: 1.2020 - val_accuracy: 0.6074
Epoch 3/10
1563/1563 [==============================] - 34s 22ms/step - loss: 0.7011 - accuracy: 0.7560 - val_loss: 1.1780 - val_accuracy: 0.6220
Epoch 4/10
1563/1563 [==============================] - 35s 23ms/step - loss: 0.5833 - accuracy: 0.7958 - val_loss: 0.8205 - val_accuracy: 0.7232
Epoch 5/10
1563/1563 [==============================] - 33s 21ms/step - loss: 0.4975 - accuracy: 0.8249 - val_loss: 0.7779 - val_accuracy: 0.7506
Epoch 6/10
1563/1563 [==============================] - 32s 21ms/step - loss: 0.4201 - accuracy: 0.8530 - val_loss: 0.8078 - val_accuracy: 0.7465
Epoch 7/10
1563/1563 [==============================] - 32s 20ms/step - loss: 0.3599 - accuracy: 0.8730 - val_loss: 0.9106 - val_accuracy: 0.7319
```

```
Epoch 8/10
1563/1563 [==============================] - 32s 20ms/step - loss: 0.3032 - accuracy: 0.8942 - val_loss: 1.0443 - val_accuracy: 0.7117
Epoch 9/10
1563/1563 [==============================] - 32s 20ms/step - loss: 0.2637 - accuracy: 0.9057 - val_loss: 0.8985 - val_accuracy: 0.7452
Epoch 10/10
1563/1563 [==============================] - 32s 20ms/step - loss: 0.2302 - accuracy: 0.9194 - val_loss: 0.9776 - val_accuracy: 0.7454
```




```
The final test accuracy of the model is 0.743119990825653.
```

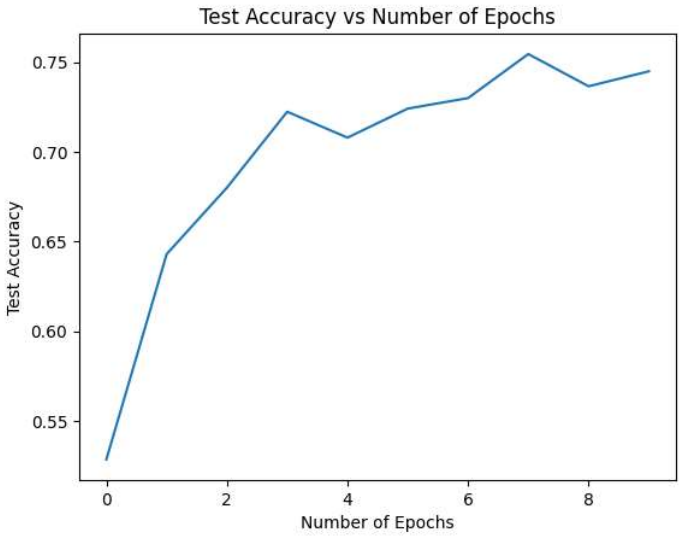
After adding batch normalization, we see from above that the training accuracy is improving significantly compared to the improvement in the test accuracy. In fact, there was an approximately 5% improvement in training accuracy, but only a 0.78% improvement in the test accuracy. This may be a sign of overfitting, so we can try adding dropout again to see if the final test accuracy improves.

In [ ]:
```python
# Add dropout

import numpy as np

total_acc = np.zeros(10)

num_models = 5
for i in range(num_models):
    # Create new model
    model = tf.keras.models.Sequential([
              tf.keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)),
              tf.keras.layers.BatchNormalization(),
              tf.keras.layers.MaxPooling2D((2, 2)),

              tf.keras.layers.Conv2D(64, (3, 3), activation='relu', padding='same'),
              tf.keras.layers.BatchNormalization(),
              tf.keras.layers.MaxPooling2D((2, 2)),

              tf.keras.layers.Conv2D(128, (3, 3), activation='relu', padding='same'),
              tf.keras.layers.BatchNormalization(),
              tf.keras.layers.MaxPooling2D((2, 2)),

              tf.keras.layers.Conv2D(128, (3, 3), activation='relu', padding='same'),
              tf.keras.layers.BatchNormalization(),
              tf.keras.layers.MaxPooling2D((2, 2)),

              tf.keras.layers.Flatten(),
              tf.keras.layers.Dropout(0.2),
              tf.keras.layers.Dense(128, activation='relu'),
              tf.keras.layers.Dropout(0.2),
              tf.keras.layers.Dense(128, activation='relu'),
              tf.keras.layers.Dropout(0.2),
              tf.keras.layers.Dense(10, activation='softmax')
            ])


    # Compile model
    model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy']  # in addition to the loss, also compute the categorization accuracy
             )

    # Train each model for 10 epochs
    history = model.fit(x_train, y_train, epochs=10, validation_data=(x_test, y_test));

    # Store training and test errors
    total_acc = np.add(total_acc, history.history['val_accuracy'])

# Calculate average accuracy across 5 models
avg_acc = np.divide(total_acc, 5)

# Plot training accuracies
plt.plot(avg_acc)
```

```
plt.title("Test Accuracy vs Number of Epochs")
plt.ylabel("Test Accuracy")
plt.xlabel("Number of Epochs")
plt.show()

print(f"The final test accuracy of the model is {avg_acc[-1]}.")
```

```
Epoch 1/10
1563/1563 [==============================] - 34s 21ms/step - loss: 1.4351 - accuracy: 0.4845 - val_loss: 1.1636 - val_accuracy: 0.5762
Epoch 2/10
1563/1563 [==============================] - 32s 21ms/step - loss: 1.0153 - accuracy: 0.6480 - val_loss: 0.9916 - val_accuracy: 0.6609
Epoch 3/10
1563/1563 [==============================] - 32s 21ms/step - loss: 0.8501 - accuracy: 0.7092 - val_loss: 0.9099 - val_accuracy: 0.6950
Epoch 4/10
1563/1563 [==============================] - 32s 21ms/step - loss: 0.7340 - accuracy: 0.7502 - val_loss: 0.8639 - val_accuracy: 0.7109
Epoch 5/10
1563/1563 [==============================] - 33s 21ms/step - loss: 0.6538 - accuracy: 0.7764 - val_loss: 0.8272 - val_accuracy: 0.7285
Epoch 6/10
1563/1563 [==============================] - 33s 21ms/step - loss: 0.5863 - accuracy: 0.7991 - val_loss: 0.9563 - val_accuracy: 0.6974
Epoch 7/10
1563/1563 [==============================] - 32s 21ms/step - loss: 0.5291 - accuracy: 0.8206 - val_loss: 1.0825 - val_accuracy: 0.6613
Epoch 8/10
1563/1563 [==============================] - 32s 21ms/step - loss: 0.4738 - accuracy: 0.8361 - val_loss: 0.7759 - val_accuracy: 0.7514
Epoch 9/10
1563/1563 [==============================] - 32s 21ms/step - loss: 0.4361 - accuracy: 0.8496 - val_loss: 0.8047 - val_accuracy: 0.7558
Epoch 10/10
1563/1563 [==============================] - 32s 21ms/step - loss: 0.3979 - accuracy: 0.8655 - val_loss: 0.8002 - val_accuracy: 0.7517
Epoch 1/10
1563/1563 [==============================] - 43s 22ms/step - loss: 1.4405 - accuracy: 0.4832 - val_loss: 1.4261 - val_accuracy: 0.4987
Epoch 2/10
1563/1563 [==============================] - 34s 22ms/step - loss: 1.0270 - accuracy: 0.6399 - val_loss: 1.0204 - val_accuracy: 0.6346
Epoch 3/10
1563/1563 [==============================] - 34s 22ms/step - loss: 0.8538 - accuracy: 0.7059 - val_loss: 0.9552 - val_accuracy: 0.6754
Epoch 4/10
1563/1563 [==============================] - 34s 22ms/step - loss: 0.7374 - accuracy: 0.7490 - val_loss: 0.7931 - val_accuracy: 0.7234
Epoch 5/10
1563/1563 [==============================] - 34s 22ms/step - loss: 0.6590 - accuracy: 0.7760 - val_loss: 0.9425 - val_accuracy: 0.6759
Epoch 6/10
1563/1563 [==============================] - 34s 22ms/step - loss: 0.5886 - accuracy: 0.7990 - val_loss: 0.7929 - val_accuracy: 0.7395
Epoch 7/10
1563/1563 [==============================] - 35s 23ms/step - loss: 0.5334 - accuracy: 0.8180 - val_loss: 0.6944 - val_accuracy: 0.7646
Epoch 8/10
1563/1563 [==============================] - 35s 22ms/step - loss: 0.4845 - accuracy: 0.8347 - val_loss: 0.7965 - val_accuracy: 0.7498
Epoch 9/10
1563/1563 [==============================] - 35s 22ms/step - loss: 0.4391 - accuracy: 0.8503 - val_loss: 0.7872 - val_accuracy: 0.7539
Epoch 10/10
1563/1563 [==============================] - 35s 22ms/step - loss: 0.4012 - accuracy: 0.8637 - val_loss: 1.0400 - val_accuracy: 0.7018
Epoch 1/10
1563/1563 [==============================] - 37s 23ms/step - loss: 1.4555 - accuracy: 0.4779 - val_loss: 1.1581 - val_accuracy: 0.5939
Epoch 2/10
1563/1563 [==============================] - 35s 22ms/step - loss: 1.0198 - accuracy: 0.6454 - val_loss: 1.1294 - val_accuracy: 0.6086
Epoch 3/10
1563/1563 [==============================] - 35s 22ms/step - loss: 0.8484 - accuracy: 0.7093 - val_loss: 0.8366 - val_accuracy: 0.7099
Epoch 4/10
1563/1563 [==============================] - 34s 22ms/step - loss: 0.7357 - accuracy: 0.7486 - val_loss: 0.7877 - val_accuracy: 0.7310
Epoch 5/10
1563/1563 [==============================] - 35s 22ms/step - loss: 0.6449 - accuracy: 0.7801 - val_loss: 0.8166 - val_accuracy: 0.7267
Epoch 6/10
1563/1563 [==============================] - 34s 22ms/step - loss: 0.5843 - accuracy: 0.8011 - val_loss: 0.8012 - val_accuracy: 0.7301
Epoch 7/10
1563/1563 [==============================] - 34s 22ms/step - loss: 0.5225 - accuracy: 0.8226 - val_loss: 0.7816 - val_accuracy: 0.7382
Epoch 8/10
1563/1563 [==============================] - 34s 22ms/step - loss: 0.4754 - accuracy: 0.8383 - val_loss: 0.7231 - val_accuracy: 0.7647
Epoch 9/10
1563/1563 [==============================] - 34s 22ms/step - loss: 0.4281 - accuracy: 0.8544 - val_loss: 0.8063 - val_accuracy: 0.7406
Epoch 10/10
1563/1563 [==============================] - 34s 22ms/step - loss: 0.3882 - accuracy: 0.8677 - val_loss: 0.7449 - val_accuracy: 0.7638
Epoch 1/10
1563/1563 [==============================] - 35s 22ms/step - loss: 1.4493 - accuracy: 0.4791 - val_loss: 1.4899 - val_accuracy: 0.4870
Epoch 2/10
1563/1563 [==============================] - 34s 22ms/step - loss: 1.0284 - accuracy: 0.6458 - val_loss: 1.0262 - val_accuracy: 0.6539
Epoch 3/10
1563/1563 [==============================] - 34s 22ms/step - loss: 0.8559 - accuracy: 0.7072 - val_loss: 0.9242 - val_accuracy: 0.6792
Epoch 4/10
1563/1563 [==============================] - 35s 22ms/step - loss: 0.7425 - accuracy: 0.7483 - val_loss: 0.8249 - val_accuracy: 0.7158
Epoch 5/10
1563/1563 [==============================] - 34s 22ms/step - loss: 0.6534 - accuracy: 0.7786 - val_loss: 1.0155 - val_accuracy: 0.6639
Epoch 6/10
1563/1563 [==============================] - 34s 22ms/step - loss: 0.5848 - accuracy: 0.8025 - val_loss: 0.9102 - val_accuracy: 0.7102
Epoch 7/10
1563/1563 [==============================] - 34s 22ms/step - loss: 0.5308 - accuracy: 0.8199 - val_loss: 0.7926 - val_accuracy: 0.7505
Epoch 8/10
1563/1563 [==============================] - 34s 22ms/step - loss: 0.4801 - accuracy: 0.8369 - val_loss: 0.7967 - val_accuracy: 0.7415
Epoch 9/10
1563/1563 [==============================] - 34s 22ms/step - loss: 0.4353 - accuracy: 0.8526 - val_loss: 0.7951 - val_accuracy: 0.7439
Epoch 10/10
1563/1563 [==============================] - 34s 22ms/step - loss: 0.3937 - accuracy: 0.8648 - val_loss: 0.8313 - val_accuracy: 0.7441
Epoch 1/10
1563/1563 [==============================] - 36s 22ms/step - loss: 1.4424 - accuracy: 0.4821 - val_loss: 1.4610 - val_accuracy: 0.4865
Epoch 2/10
1563/1563 [==============================] - 34s 22ms/step - loss: 1.0214 - accuracy: 0.6475 - val_loss: 0.9806 - val_accuracy: 0.6569
Epoch 3/10
1563/1563 [==============================] - 34s 22ms/step - loss: 0.8466 - accuracy: 0.7079 - val_loss: 1.0810 - val_accuracy: 0.6409
Epoch 4/10
1563/1563 [==============================] - 34s 22ms/step - loss: 0.7347 - accuracy: 0.7506 - val_loss: 0.7880 - val_accuracy: 0.7310
Epoch 5/10
1563/1563 [==============================] - 34s 22ms/step - loss: 0.6555 - accuracy: 0.7749 - val_loss: 0.7430 - val_accuracy: 0.7449
Epoch 6/10
1563/1563 [==============================] - 34s 22ms/step - loss: 0.5795 - accuracy: 0.8032 - val_loss: 0.7726 - val_accuracy: 0.7437
Epoch 7/10
1563/1563 [==============================] - 34s 22ms/step - loss: 0.5265 - accuracy: 0.8208 - val_loss: 0.8153 - val_accuracy: 0.7356
```

```
Epoch 8/10
1563/1563 [==============================] - 34s 22ms/step - loss: 0.4771 - accuracy: 0.8371 - val_loss: 0.7412 - val_accuracy: 0.7657
Epoch 9/10
1563/1563 [==============================] - 34s 22ms/step - loss: 0.4320 - accuracy: 0.8497 - val_loss: 1.0076 - val_accuracy: 0.6889
Epoch 10/10
1563/1563 [==============================] - 34s 22ms/step - loss: 0.3896 - accuracy: 0.8669 - val_loss: 0.7651 - val_accuracy: 0.7635
```




```
The final test accuracy of the model is 0.744979989528656.
```

The addition of dropout layers with a dropout rate of 0.2 led to a 0.19% improvement in the final test accuracy. Thus, after adding an additional convolutional layer, an additional hidden fully-connected layer with more neurons, batch normalization and dropout, the final test accuracy of the model is 74.50%.

In [ ]: