In [1]:

```
#libraries to be used
import numpy as np
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from sklearn.metrics import f1_score
import matplotlib.pyplot as plt
```

In [2]:

```
#Custom Activation Functions
initializer0 = keras.initializers. RandomUniform(minval = -1, maxval =2)
initializer1 = keras.initializers. RandomUniform(minval = -1, maxval =2)

class Adaact(keras.layers.Layer):
    def __init__(self):
        super(Adaact, self).__init__()
        self.k0 = self.add_weight(name='k0', shape = (), initializer=initializer0, trainable=T
        self.k1 = self.add_weight(name='k1', shape = (), initializer=initializer1, trainable=T

    def call(self, inputs):
        return self.k0 + tf.multiply(inputs, self.k1)
```

In [3]:

```
num_classes = 10
input_shape = (28, 28, 1)

# the data, split between train and test sets
(x_train, y_train), (x_test, y_test) = keras.datasets.mnist.load_data()

# Scale images to the [0, 1] range
x_train = x_train.astype("float32") / 255
x_test = x_test.astype("float32") / 255
# Make sure images have shape (28, 28, 1)
x_train = np.expand_dims(x_train, -1)
x_test = np.expand_dims(x_test, -1)
print("x_train shape:", x_train.shape)
print(x_train.shape[0], "train samples")
print(x_test.shape[0], "test samples")


# convert class vectors to binary class matrices
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)
```

```
x_train shape: (60000, 28, 28, 1)
60000 train samples
10000 test samples
```

In [4]:

```
inputs = keras.Input(shape=input_shape)
x = layers.Conv2D(32, kernel_size=(3, 3))(inputs)
adaact = Adaact()
x = adaact(x)
x = layers.MaxPooling2D(pool_size=(2, 2))(x)
x = layers.Conv2D(64, kernel_size=(3, 3))(x)
x = adaact(x)
x = layers.MaxPooling2D(pool_size=(2, 2))(x)
x = layers.Flatten()(x)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(num_classes, activation="softmax")(x)

model = keras.Model(inputs=inputs, outputs=outputs, name="mnist_adaact")
model.summary()
```

```
Model: "mnist_adaact"
________________________________________________________________________________
__________________
 Layer (type)                   Output Shape          Param #     Connected to
================================================================================
==================
 input_1 (InputLayer)           [(None, 28, 28, 1)]  0           []

 conv2d (Conv2D)                (None, 26, 26, 32)   320         ['input_1[0]
[0]']

 adaact (Adaact)                multiple             2           ['conv2d[0]
[0]',
                                                                  'conv2d_1[0]
[0]']

 max_pooling2d (MaxPooling2D)   (None, 13, 13, 32)   0           ['adaact[0]
[0]']

 conv2d_1 (Conv2D)              (None, 11, 11, 64)   18496       ['max_pooling2d
[0][0]']

 max_pooling2d_1 (MaxPooling2D)  (None, 5, 5, 64)    0           ['adaact[1]
[0]']

 flatten (Flatten)              (None, 1600)         0           ['max_pooling2d
_1[0][0]']

 dropout (Dropout)              (None, 1600)         0           ['flatten[0]
[0]']

 dense (Dense)                  (None, 10)           16010       ['dropout[0]
[0]']

================================================================================
==================
Total params: 34,828
Trainable params: 34,828
Non-trainable params: 0
________________________________________________________________________________
__________________
```

In [5]:

```
batch_size = 128
epochs = 15

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

history=model.fit(x_train, y_train, batch_size=batch_size, epochs=epochs, validation_split=0.1
```

```
Epoch 1/15
422/422 [==============================] - 37s 80ms/step - loss: 0.7713 - accura
cy: 0.7494 - val_loss: 0.1874 - val_accuracy: 0.9457
Epoch 2/15
422/422 [==============================] - 34s 81ms/step - loss: 0.2251 - accura
cy: 0.9341 - val_loss: 0.1082 - val_accuracy: 0.9695
Epoch 3/15
422/422 [==============================] - 35s 83ms/step - loss: 0.1501 - accura
cy: 0.9544 - val_loss: 0.0785 - val_accuracy: 0.9798
Epoch 4/15
422/422 [==============================] - 37s 89ms/step - loss: 0.1165 - accura
cy: 0.9648 - val_loss: 0.0653 - val_accuracy: 0.9812
Epoch 5/15
422/422 [==============================] - 38s 89ms/step - loss: 0.0986 - accura
cy: 0.9700 - val_loss: 0.0591 - val_accuracy: 0.9833
Epoch 6/15
422/422 [==============================] - 34s 80ms/step - loss: 0.0875 - accura
cy: 0.9731 - val_loss: 0.0574 - val_accuracy: 0.9842
Epoch 7/15
422/422 [==============================] - 35s 82ms/step - loss: 0.0795 - accura
cy: 0.9750 - val_loss: 0.0521 - val_accuracy: 0.9855
Epoch 8/15
422/422 [==============================] - 37s 88ms/step - loss: 0.0760 - accura
cy: 0.9766 - val_loss: 0.0503 - val_accuracy: 0.9862
Epoch 9/15
422/422 [==============================] - 36s 85ms/step - loss: 0.0685 - accura
cy: 0.9785 - val_loss: 0.0487 - val_accuracy: 0.9860
Epoch 10/15
422/422 [==============================] - 36s 86ms/step - loss: 0.0661 - accura
cy: 0.9791 - val_loss: 0.0484 - val_accuracy: 0.9863
Epoch 11/15
422/422 [==============================] - 38s 90ms/step - loss: 0.0630 - accura
cy: 0.9802 - val_loss: 0.0464 - val_accuracy: 0.9870
Epoch 12/15
422/422 [==============================] - 41s 96ms/step - loss: 0.0591 - accura
cy: 0.9817 - val_loss: 0.0469 - val_accuracy: 0.9870
Epoch 13/15
422/422 [==============================] - 38s 91ms/step - loss: 0.0569 - accura
cy: 0.9822 - val_loss: 0.0479 - val_accuracy: 0.9877
Epoch 14/15
422/422 [==============================] - 36s 85ms/step - loss: 0.0550 - accura
cy: 0.9821 - val_loss: 0.0487 - val_accuracy: 0.9882
Epoch 15/15
422/422 [==============================] - 39s 91ms/step - loss: 0.0527 - accura
cy: 0.9832 - val_loss: 0.0441 - val_accuracy: 0.9883
```

In [6]:

```
# final parameter values at the end of training
final_parameter_values = model.get_weights()
print(final_parameter_values)
```

```
[array([[[[-0.28193203,  0.19224311, -0.02100194, -0.02192995,
           -0.29359892, -0.00308298, -0.00941813, -0.12579012,
            0.12277672,  0.06369605, -0.12851134,  0.00565888,
           -0.2460288 , -0.13639517,  0.22445598, -0.16994   ,
            0.22921774,  0.410502  , -0.03735056, -0.10685643,
           -0.4109105 ,  0.06070343, -0.29490903,  0.36812273,
           -0.22057325, -0.12388425,  0.02961864,  0.05827512,
            0.327816  , -0.02785207,  0.06403695,  0.26367396]],

         [[-0.33996472,  0.05464279,  0.16835459, -0.1951495 ,
           -0.03713529,  0.05467163, -0.13825217,  0.17349787,
           -0.46944255,  0.07936981, -0.13371399, -0.23376897,
            0.17610985, -0.18018048,  0.51445377, -0.2923995 ,
           -0.12890233, -0.09360644, -0.05877862, -0.29042983,
           -0.05745468, -0.00774323, -0.11454602,  0.53620803,
            0.1127165 ,  0.07334656,  0.122214  ,  0.00642166,
            0.51766187,  0.31699735,  0.09343468, -0.18147331]],

         [[-0.37229544,  0.06385107,  0.37445962,  0.11154728,
```

In [7]:

```
print("AF coefficients (weights) {}".format(adaact.get_weights()))
```

```
AF coefficients (weights) [0.09615643, -0.90013105]
```

In [8]:

```
# Obtain test loss and test accuracy
score = model.evaluate(x_test, y_test, verbose=0)
print("Test loss:", score[0])
print("Test accuracy:", score[1])
```

```
Test loss: 0.03889978677034378
Test accuracy: 0.9868000149726868
```

In [9]:

```
# Obtain train loss and train accuracy
score = model.evaluate(x_train, y_train, verbose=0)
print("train loss:", score[0])
print("train accuracy:", score[1])
```

```
train loss: 0.028460074216127396
train accuracy: 0.9917500019073486
```

In [10]:

```
# Evaluate the model on the test set
y_pred = model.predict(x_test)
y_pred = np.argmax(y_pred, axis=1)
y_true = np.argmax(y_test, axis=1)
```

```
313/313 [==============================] - 2s 5ms/step
```

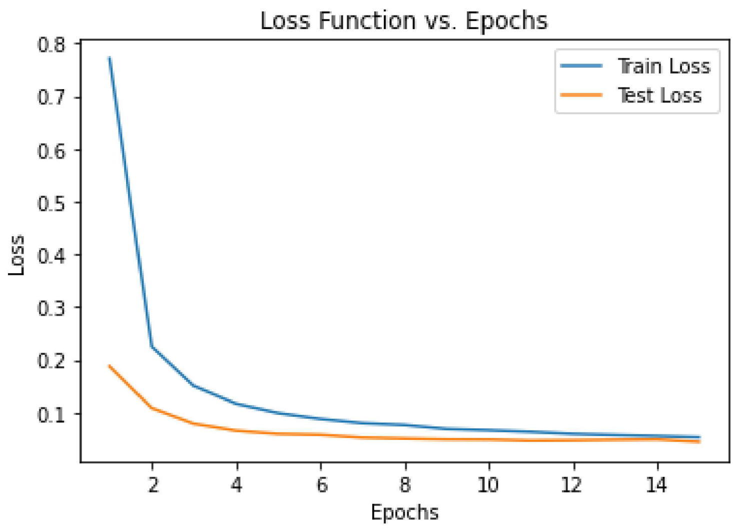
In [11]:

```
# Compute the F1-score
f1 = f1_score(y_true, y_pred, average='weighted')
print("f1 score is {}".format(f1))
```

```
f1 score is 0.9867963904416119
```

In [12]:

```
# Plot the loss function vs. epochs
train_loss = history.history['loss']
test_loss = history.history['val_loss']
plt.plot(range(1, epochs+1), train_loss, label='Train Loss')
plt.plot(range(1, epochs+1), test_loss, label='Test Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Loss Function vs. Epochs')
plt.legend()
plt.show()
```



In [ ]: