Experiment - 3

1 Aim: Design a CNN architecture to implement the image classification task over an image dataset. Perform the Hyper-parameter tuning and record the results.

2 Dataset Description

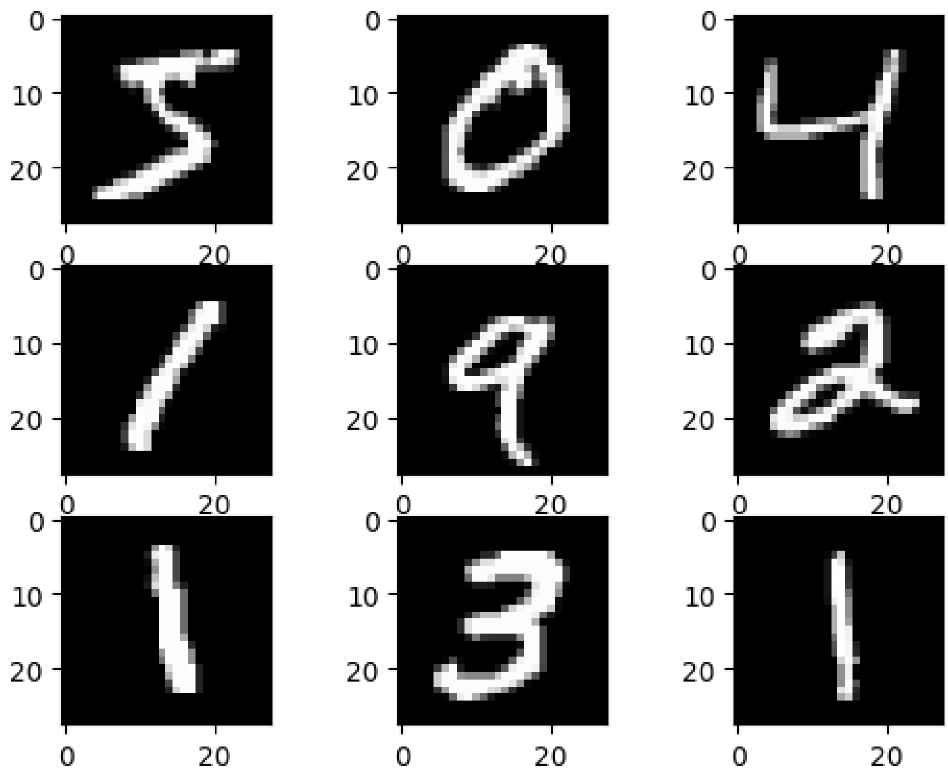
- The data that will be incorporated is the **MNIST database** which contains 60,000 images for training and 10,000 test images.
- The dataset consists of small square 28×28 pixel grayscale images of handwritten single digits between 0 and 9
- The MNIST dataset is conveniently bundled within Keras, and we can easily analyze some of its features in Python.

[1]:
```
from tensorflow import keras
from keras.datasets import mnist     # MNIST dataset is included in Keras
(X_train, y_train), (X_test, y_test) = mnist.load_data()

print("X_train shape", X_train.shape)
print("y_train shape", y_train.shape)
print("X_test shape", X_test.shape)
print("y_test shape", y_test.shape)
```

```
WARNING:tensorflow:From c:\MAGNUS\files\GAI\aiml\Lib\site-
packages\keras\src\losses.py:2976: The name
tf.losses.sparse_softmax_cross_entropy is deprecated. Please use
tf.compat.v1.losses.sparse_softmax_cross_entropy instead.

X_train shape (60000, 28, 28)
y_train shape (60000,)
X_test shape (10000, 28, 28)
y_test shape (10000,)
```

[2]:
```
# Visualize any random image
# Plot first few images
import matplotlib.pyplot as plt
```

```
for i in range(9):
        # define subplot
        plt.subplot(3,3,i+1) # 3 rows, 3 col, pos
        # plot raw pixel data
        plt.imshow(X_train[i],cmap='gray')
# show the figure
plt.show()
```



2.0.1 Formatting the Input

[3]:
```
# Single-channel input data (grey-scale)
# First apply convolutions then flatten

X_train = X_train.reshape(60000, 28, 28, 1) # single-channel input
X_test = X_test.reshape(10000, 28, 28, 1)

X_train = X_train.astype('float32')         # change integers to 32-bit floating point numbers
X_test = X_test.astype('float32')

X_train /= 255                              # min-max normalization
```

```
X_test /= 255

print("Training matrix shape", X_train.shape)
print("Testing matrix shape", X_test.shape)
```

```
Training matrix shape (60000, 28, 28, 1)
Testing matrix shape (10000, 28, 28, 1)
```

3 Convolutional Neural Network

- Convolution applies **kernels** (filters) that traverse through each image and generate **feature maps**
- keras Conv2D: https://keras.io/api/layers/convolution_layers/convolution2d/
- Each kernel in a CNN learns a different characteristic of an image.
- **max pooling** helps in reducing the number of learnable parameters, and decreasing the computational cost (e.g. system memory)

3.1 Building a Convolutional Neural Network

[4]:
```
from keras import backend as K
from keras import __version__

print('Using Keras version:', __version__, 'backend:', K.backend())
```

```
Using Keras version: 2.15.0 backend: tensorflow
```

[5]:
```
# import cnn layers
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Dropout, Flatten, Dense
import tensorflow as tf
```

[6]:
```
model = Sequential()                                    # Linear stacking of layers

# Convolution Layer 1: 8 filters, kernel size 3x3, relu activation, valid␣
↪padding, stride 1
model.add(Conv2D(8,kernel_size=(3,3),activation='relu',strides=1,␣
↪padding='valid',input_shape=(28, 28, 1)))
# MaxPooling: pool size 2, stride 2
model.add(MaxPooling2D(pool_size=2,strides=2))
# Convolution Layer 2: 16 filters, kernel size 3x3, relu activation, valid␣
↪padding, stride 1
model.add(Conv2D(16,kernel_size=(3,3),activation='relu',strides=1,␣
↪padding='valid'))
# MaxPooling: pool size 2, stride 2
model.add(MaxPooling2D(pool_size=2,strides=2))
# Flatten final feature matrix into a 1d array
model.add(Flatten(input_shape=(28,28)))
```

```python
# Fully Connected Layer: 64 units and relu activation
model.add(Dense(64,activation='relu'))
# Dropout layer, 0.2 rate
model.add(Dropout(rate=0.2))
# Final output dense Layer
model.add(Dense(10, activation='softmax'))

#Compile the model with sparse_categorical_crossentropy loss
model.
 ↪compile(loss='sparse_categorical_crossentropy',optimizer='adam',metrics=['accuracy'])
```

```
WARNING:tensorflow:From c:\MAGNUS\files\GAI\aiml\Lib\site-
packages\keras\src\backend.py:873: The name tf.get_default_graph is deprecated.
Please use tf.compat.v1.get_default_graph instead.

WARNING:tensorflow:From c:\MAGNUS\files\GAI\aiml\Lib\site-
packages\keras\src\layers\pooling\max_pooling2d.py:161: The name tf.nn.max_pool
is deprecated. Please use tf.nn.max_pool2d instead.

WARNING:tensorflow:From c:\MAGNUS\files\GAI\aiml\Lib\site-
packages\keras\src\optimizers\__init__.py:309: The name tf.train.Optimizer is
deprecated. Please use tf.compat.v1.train.Optimizer instead.
```

[7]:
```python
model.build()
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 26, 26, 8)         80

 max_pooling2d (MaxPooling2  (None, 13, 13, 8)         0
 D)

 conv2d_1 (Conv2D)           (None, 11, 11, 16)        1168

 max_pooling2d_1 (MaxPoolin  (None, 5, 5, 16)          0
 g2D)

 flatten (Flatten)           (None, 400)               0

 dense (Dense)               (None, 64)                25664

 dropout (Dropout)           (None, 64)                0

 dense_1 (Dense)             (None, 10)                650
```

```
=================================================================
Total params: 27562 (107.66 KB)
Trainable params: 27562 (107.66 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

[8]:

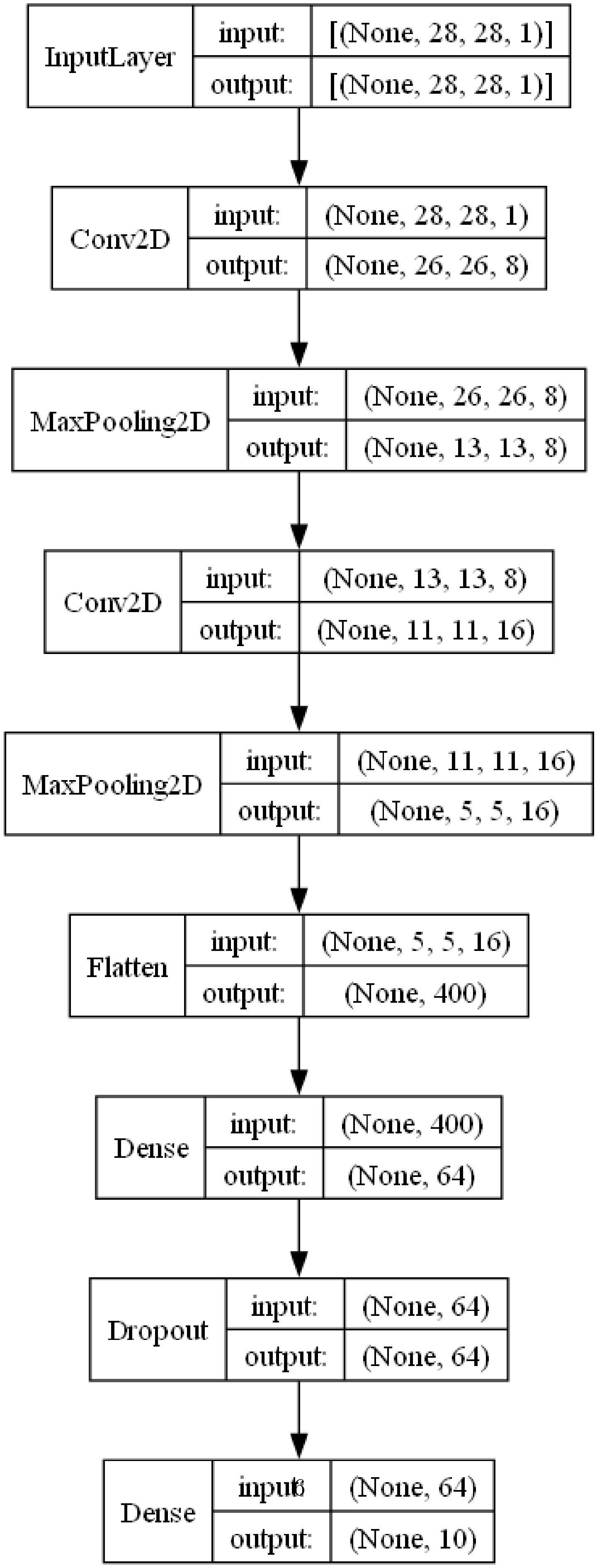
```python
# Conv1: 3x3 kernels, one for each the single channel, 8 such filters and 8␣
↪biases
print('Conv1: ',3*3*1*8 + 8)
# Conv2: 3x3 kernels, one for each of the 8 channels, 16 such filters and 16␣
↪biases
print('Conv2: ',3*3*8*16 + 16)
# input to dense layer
print('Flatten:', 5*5*16)
# 400 inputs, 1 bias connected to each of 64 units in dense layer
print('Dense1: ',400*64+64)
# 64 inputs, 1 bias connected to each of 10 units in output layer
print('Dense2: ',64*10+10)
```

```
Conv1:  80
Conv2:  1168
Flatten: 400
Dense1:  25664
Dense2:  650
```

[9]:

```python
# Visualize the model
from keras.utils import plot_model
plot_model(model, show_shapes=True, show_layer_names=False)
```

[9]:


InputLayer
input:
[(None, 28, 28, 1)]
output:
[(None, 28, 28, 1)]
Conv2D
input:
(None, 28, 28, 1)
output:
(None, 26, 26, 8)
MaxPooling2D
input:
(None, 26, 26, 8)
output:
(None, 13, 13, 8)
Conv2D
input:
(None, 13, 13, 8)
output:
(None, 11, 11, 16)
MaxPooling2D
input:
(None, 11, 11, 16)
output:
(None, 5, 5, 16)
Flatten
input:
(None, 5, 5, 16)
output:
(None, 400)
Dense
input:
(None, 400)
output:
(None, 64)
Dropout
input:
(None, 64)
output:
(None, 64)
Dense
input:
(None, 64)
output:
(None, 10)

Train the model

- Validation data =0.2*60,000 = 12,000
- Batch size = 128
- Number of batches during training are (60000-12000)/128 = 48000/128 = 375

[10]:
```
# Train the model
batch_size=128
epochs=10
hist = model.fit(X_train,␣
 ↪y_train,epochs=epochs,batch_size=batch_size,verbose=1,validation_split=0.2)
```

```
Epoch 1/10
WARNING:tensorflow:From c:\MAGNUS\files\GAI\aiml\Lib\site-
packages\keras\src\utils\tf_utils.py:492: The name tf.ragged.RaggedTensorValue
is deprecated. Please use tf.compat.v1.ragged.RaggedTensorValue instead.

WARNING:tensorflow:From c:\MAGNUS\files\GAI\aiml\Lib\site-
packages\keras\src\engine\base_layer_utils.py:384: The name
tf.executing_eagerly_outside_functions is deprecated. Please use
tf.compat.v1.executing_eagerly_outside_functions instead.

375/375 [==============================] - 5s 7ms/step - loss: 0.5493 -
accuracy: 0.8431 - val_loss: 0.1316 - val_accuracy: 0.9622
Epoch 2/10
375/375 [==============================] - 2s 6ms/step - loss: 0.1364 -
accuracy: 0.9595 - val_loss: 0.0811 - val_accuracy: 0.9765
Epoch 3/10
375/375 [==============================] - 2s 6ms/step - loss: 0.0980 -
accuracy: 0.9702 - val_loss: 0.0675 - val_accuracy: 0.9797
Epoch 4/10
375/375 [==============================] - 2s 6ms/step - loss: 0.0806 -
accuracy: 0.9753 - val_loss: 0.0629 - val_accuracy: 0.9816
Epoch 5/10
375/375 [==============================] - 3s 8ms/step - loss: 0.0676 -
accuracy: 0.9788 - val_loss: 0.0538 - val_accuracy: 0.9852
Epoch 6/10
375/375 [==============================] - 3s 8ms/step - loss: 0.0591 -
accuracy: 0.9822 - val_loss: 0.0495 - val_accuracy: 0.9861
Epoch 7/10
375/375 [==============================] - 3s 8ms/step - loss: 0.0535 -
accuracy: 0.9833 - val_loss: 0.0509 - val_accuracy: 0.9857
Epoch 8/10
375/375 [==============================] - 3s 8ms/step - loss: 0.0470 -
accuracy: 0.9856 - val_loss: 0.0479 - val_accuracy: 0.9870
Epoch 9/10
```

```
375/375 [==============================] - 3s 7ms/step - loss: 0.0424 -
accuracy: 0.9867 - val_loss: 0.0448 - val_accuracy: 0.9877
Epoch 10/10
375/375 [==============================] - 2s 6ms/step - loss: 0.0403 -
accuracy: 0.9876 - val_loss: 0.0434 - val_accuracy: 0.9884
```

3.1.1 Evaluate Model

[11]:
```
score = model.evaluate(X_test, y_test, verbose = 0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

```
Test loss: 0.036152534186840006
Test accuracy: 0.9889000058174133
```

[12]:
```
# make one prediction
print('Actual class:',y_test[0])
print('Class Probabilities:')
model.predict(X_test[0].reshape(1,28,28,1))
```

```
Actual class: 7
Class Probabilities:
1/1 [==============================] - 0s 103ms/step
```

[12]:
```
array([[2.8043439e-06, 1.2572475e-07, 4.6315331e-06, 1.4247337e-05,
        9.9764907e-10, 6.4714554e-09, 8.9647369e-12, 9.9997783e-01,
        1.2663826e-08, 3.6554039e-07]], dtype=float32)
```

[13]:
```
import numpy as np
yhat_test = np.argmax(model.predict(X_test),axis=-1)
print(yhat_test[0:10])
print(y_test[0:10])
```

```
313/313 [==============================] - 1s 2ms/step
[7 2 1 0 4 1 4 9 5 9]
[7 2 1 0 4 1 4 9 5 9]
```

[14]:
```
from sklearn.metrics import accuracy_score
print('Accuracy:')
print(float(accuracy_score(y_test, yhat_test))*100,'%')
```

```
Accuracy:
98.89 %
```

[15]:
```
from sklearn.metrics import confusion_matrix
print('Confusion Matrix:')
print(confusion_matrix(y_test, yhat_test))
```

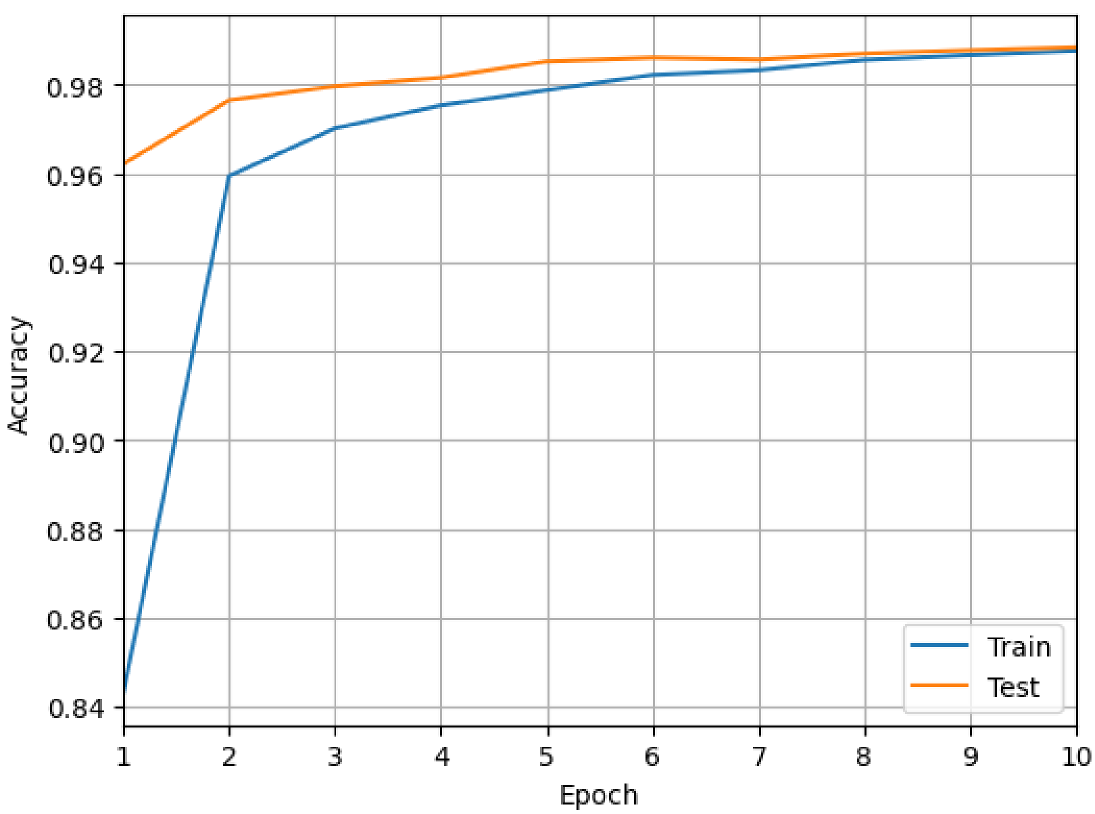
```
Confusion Matrix:
[[ 977    0    0    0    1    0    0    1    1    0]
```

```
[   0 1132    1    0    0    0    1    0    1    0]
[   2    2 1022    0    2    0    0    2    2    0]
[   1    0    0  997    0    4    0    1    5    2]
[   0    0    0    0  978    0    0    0    1    3]
[   2    0    1    5    0  879    2    1    1    1]
[   5    2    0    0    3    2  942    0    4    0]
[   0    3    8    1    1    0    0 1009    0    6]
[   3    0    0    1    1    1    0    2  962    4]
[   3    3    0    0    8    1    0    0    3  991]]
```

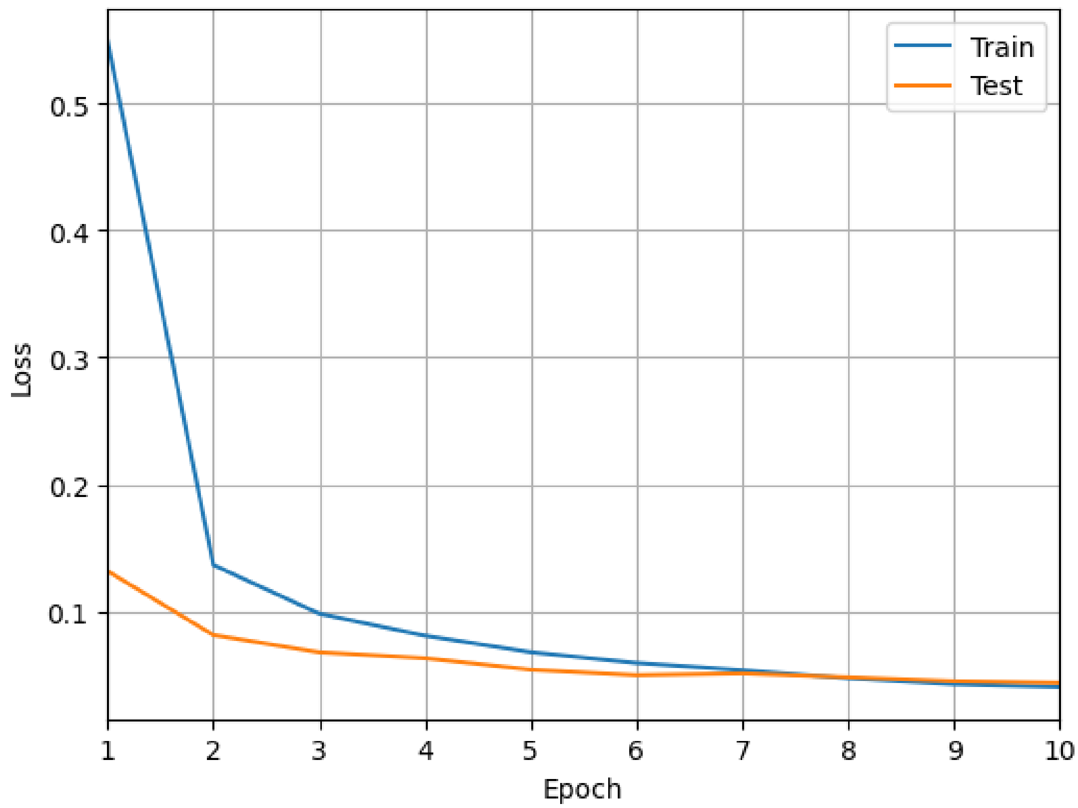
3.1.2 Plot Learning curves

[16]:
```
hist.history.keys()
epochRange = range(1,epochs+1)
```

[17]:
```
# Plot Accuracy vs epochs (DIY)
plt.plot(epochRange,hist.history['accuracy'])
plt.plot(epochRange,hist.history['val_accuracy'])
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.grid()
plt.xlim((1,epochs))
plt.legend(['Train','Test'])
plt.show()
```



[18]:
```
# Plot Loss vs epochs (DIY)
plt.plot(epochRange,hist.history['loss'])
plt.plot(epochRange,hist.history['val_loss'])
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.grid()
plt.xlim((1,epochs))
plt.legend(['Train','Test'])
plt.show()
```



3.2 Removing the dropout layer

[19]:
```
model_2 = Sequential() # Linear stacking of layers

# Convolution Layer 1: 8 filters, kernel size 3x3, relu activation, valid␣
↪padding, stride 1
model_2.add(Conv2D(8,kernel_size=(3,3),activation='relu',strides=1,␣
↪padding='valid',input_shape=(28, 28, 1)))
```

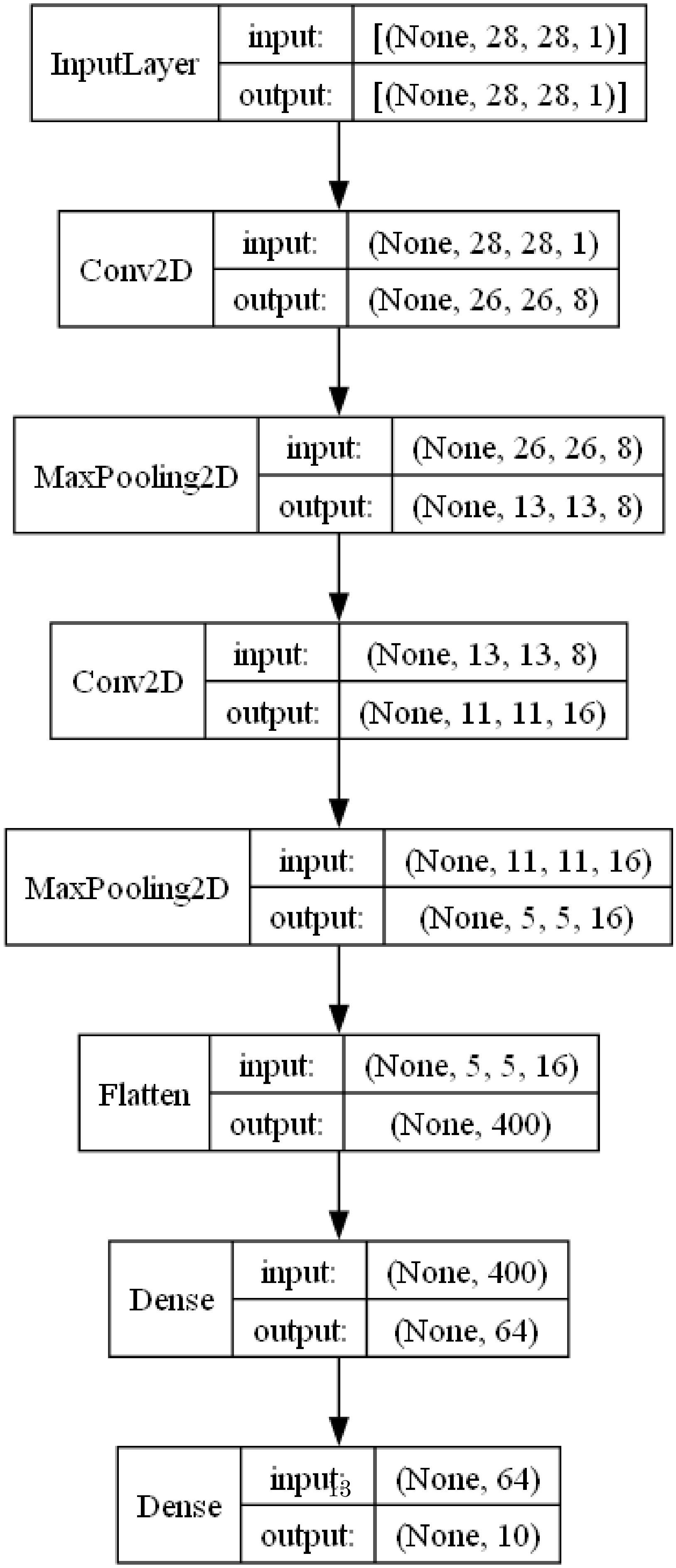
```
# MaxPooling: pool size 2, stride 2
model_2.add(MaxPooling2D(pool_size=2,strides=2))
# Convolution Layer 2: 16 filters, kernel size 3x3, relu activation, valid␣
↪padding, stride 1
model_2.add(Conv2D(16,kernel_size=(3,3),activation='relu',strides=1,␣
↪padding='valid'))
# MaxPooling: pool size 2, stride 2
model_2.add(MaxPooling2D(pool_size=2,strides=2))
# Flatten final feature matrix into a 1d array
model_2.add(Flatten(input_shape=(28,28)))
# Fully Connected Layer: 64 units and relu activation
model_2.add(Dense(64,activation='relu'))
# Final output dense Layer
model_2.add(Dense(10, activation='softmax'))
#Compile the new model with sparse_categorical_crossentropy loss
model_2.
↪compile(loss='sparse_categorical_crossentropy',optimizer='adam',metrics=['accuracy'])
```

[20]:
```
model_2.build()
model_2.summary()
```

```
Model: "sequential_1"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_2 (Conv2D)           (None, 26, 26, 8)         80

 max_pooling2d_2 (MaxPoolin  (None, 13, 13, 8)         0
 g2D)

 conv2d_3 (Conv2D)           (None, 11, 11, 16)        1168

 max_pooling2d_3 (MaxPoolin  (None, 5, 5, 16)          0
 g2D)

 flatten_1 (Flatten)         (None, 400)               0

 dense_2 (Dense)             (None, 64)                25664

 dense_3 (Dense)             (None, 10)                650

=================================================================
Total params: 27562 (107.66 KB)
Trainable params: 27562 (107.66 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

[21]:
```python
plot_model(model_2, show_shapes=True, show_layer_names=False)
```

[21]:


InputLayer
input:
[(None, 28, 28, 1)]
output:
[(None, 28, 28, 1)]
Conv2D
input:
(None, 28, 28, 1)
output:
(None, 26, 26, 8)
MaxPooling2D
input:
(None, 26, 26, 8)
output:
(None, 13, 13, 8)
Conv2D
input:
(None, 13, 13, 8)
output:
(None, 11, 11, 16)
MaxPooling2D
input:
(None, 11, 11, 16)
output:
(None, 5, 5, 16)
Flatten
input:
(None, 5, 5, 16)
output:
(None, 400)
Dense
input:
(None, 400)
output:
(None, 64)
Dense
input:
(None, 64)
output:
(None, 10)

[22]:
```
batch_size=128
epochs=10
hist_2 = model_2.fit(X_train,␣
↪y_train,epochs=epochs,batch_size=batch_size,verbose=1,validation_split=0.2)
```

```
Epoch 1/10
375/375 [==============================] - 5s 8ms/step - loss: 0.4212 -
accuracy: 0.8790 - val_loss: 0.1563 - val_accuracy: 0.9528
Epoch 2/10
375/375 [==============================] - 3s 7ms/step - loss: 0.1156 -
accuracy: 0.9651 - val_loss: 0.1001 - val_accuracy: 0.9708
Epoch 3/10
375/375 [==============================] - 3s 7ms/step - loss: 0.0839 -
accuracy: 0.9741 - val_loss: 0.0988 - val_accuracy: 0.9704
Epoch 4/10
375/375 [==============================] - 2s 6ms/step - loss: 0.0673 -
accuracy: 0.9788 - val_loss: 0.0672 - val_accuracy: 0.9802
Epoch 5/10
375/375 [==============================] - 2s 6ms/step - loss: 0.0564 -
accuracy: 0.9821 - val_loss: 0.0659 - val_accuracy: 0.9797
Epoch 6/10
375/375 [==============================] - 2s 6ms/step - loss: 0.0472 -
accuracy: 0.9851 - val_loss: 0.0608 - val_accuracy: 0.9828
Epoch 7/10
375/375 [==============================] - 2s 6ms/step - loss: 0.0443 -
accuracy: 0.9861 - val_loss: 0.0540 - val_accuracy: 0.9838
Epoch 8/10
375/375 [==============================] - 2s 6ms/step - loss: 0.0375 -
accuracy: 0.9878 - val_loss: 0.0580 - val_accuracy: 0.9837
Epoch 9/10
375/375 [==============================] - 2s 6ms/step - loss: 0.0343 -
accuracy: 0.9894 - val_loss: 0.0619 - val_accuracy: 0.9816
Epoch 10/10
375/375 [==============================] - 2s 6ms/step - loss: 0.0303 -
accuracy: 0.9905 - val_loss: 0.0502 - val_accuracy: 0.9853
```

[37]:
```
score_2 = model_2.evaluate(X_test, y_test, verbose = 0)
print('Test loss:', score_2[0])
print('Test accuracy:', score_2[1])
```

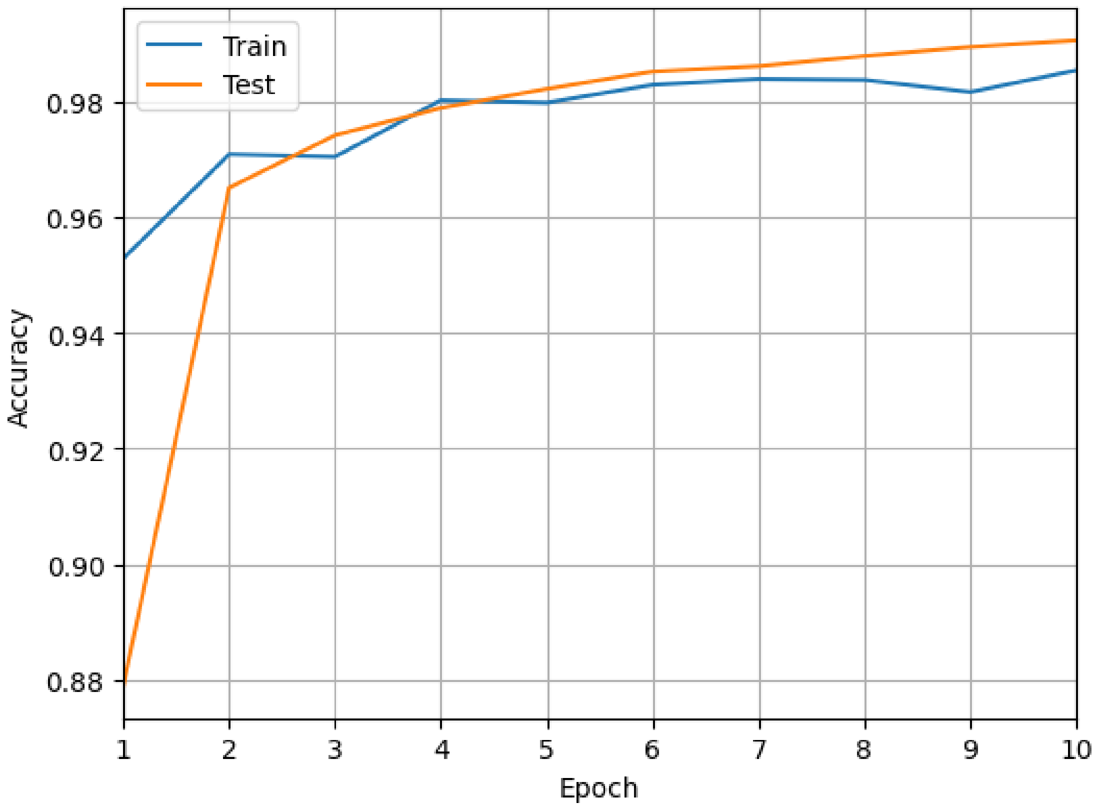
```
Test loss: 0.034381765872240067
Test accuracy: 0.9879999756813049
```

[48]:
```
yhat_test = np.argmax(model_2.predict(X_test),axis=-1)
print(yhat_test[0:10])
```

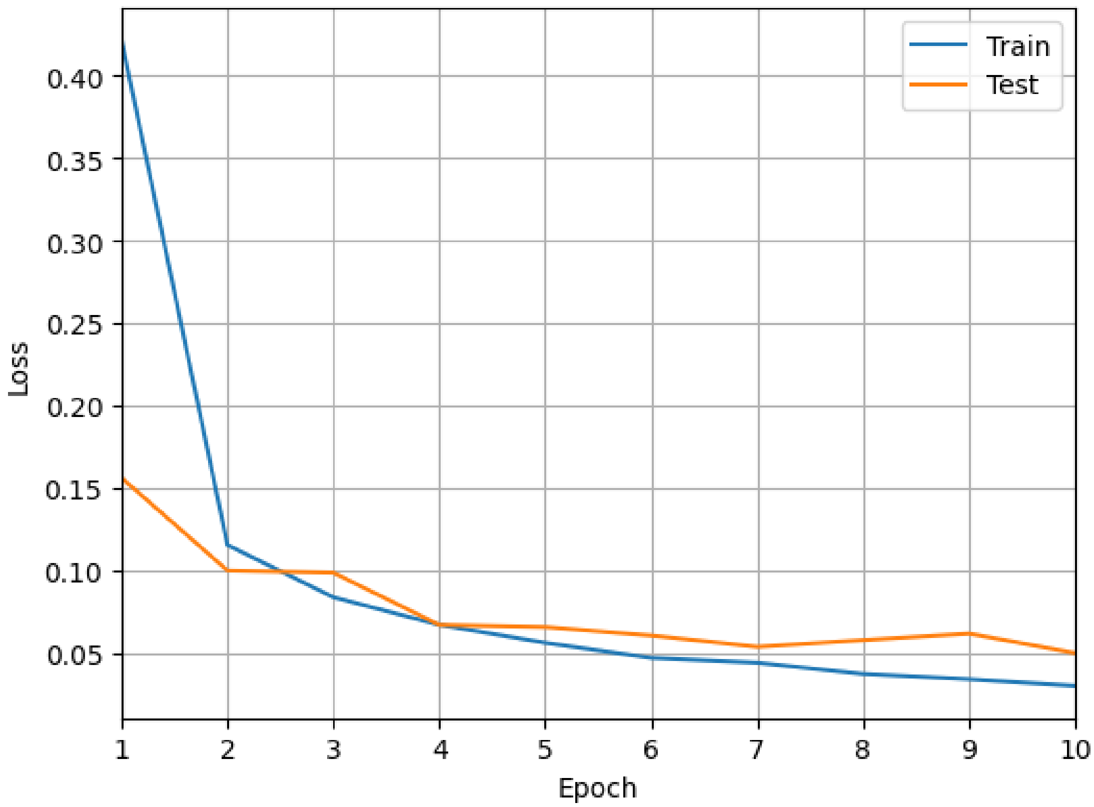
```
print(y_test[0:10])
print('Confusion Matrix:')
print(confusion_matrix(y_test, yhat_test))
print('Accuracy:')
print(float(accuracy_score(y_test, yhat_test))*100,'%')
```

```
313/313 [==============================] - 1s 3ms/step
[7 2 1 0 4 1 4 9 5 9]
[7 2 1 0 4 1 4 9 5 9]
Confusion Matrix:
[[ 972    0    0    0    0    0    3    2    3    0]
 [    0 1131    1    0    0    0    1    0    2    0]
 [    1    3 1021    0    1    1    1    1    3    0]
 [    1    0    1 1000    0    3    0    1    4    0]
 [    1    1    2    0  966    0    1    1    0   10]
 [    1    0    0    4    0  881    2    1    3    0]
 [    3    3    0    0    4    3  944    0    1    0]
 [    0    1   11    3    0    0    0 1009    1    3]
 [    2    0    3    1    1    0    0    1  964    2]
 [    1    2    0    2    5    2    0    5    0  992]]
Accuracy:
98.8 %
```

[25]:
```
plt.plot(epochRange,hist_2.history['val_accuracy'])
plt.plot(epochRange,hist_2.history['accuracy'])
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.grid()
plt.xlim((1,epochs))
plt.legend(['Train','Test'])
plt.show()
```



[26]:
```
plt.plot(epochRange,hist_2.history['loss'])
plt.plot(epochRange,hist_2.history['val_loss'])
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.grid()
plt.xlim((1,epochs))
plt.legend(['Train','Test'])
plt.show()
```



3.3 Increasing filters in Convolution layers

[38]:
```
model_3 = Sequential() # Linear stacking of layers

# Convolution Layer 1: 64 filters, kernel size 3x3, relu activation, valid␣
↪padding, stride 1
model_3.add(Conv2D(64,kernel_size=(3,3),activation='relu',strides=1,␣
↪padding='valid',input_shape=(28, 28, 1)))
# MaxPooling: pool size 2, stride 2
model_3.add(MaxPooling2D(pool_size=2,strides=2))
# Convolution Layer 2: 64 filters, kernel size 3x3, relu activation, valid␣
↪padding, stride 1
model_3.add(Conv2D(64,kernel_size=(3,3),activation='relu',strides=1,␣
↪padding='valid'))
# MaxPooling: pool size 2, stride 2
model_3.add(MaxPooling2D(pool_size=2,strides=2))
# Flatten final feature matrix into a 1d array
model_3.add(Flatten(input_shape=(28,28)))
# Fully Connected Layer: 64 units and relu activation
model_3.add(Dense(64,activation='relu'))
```

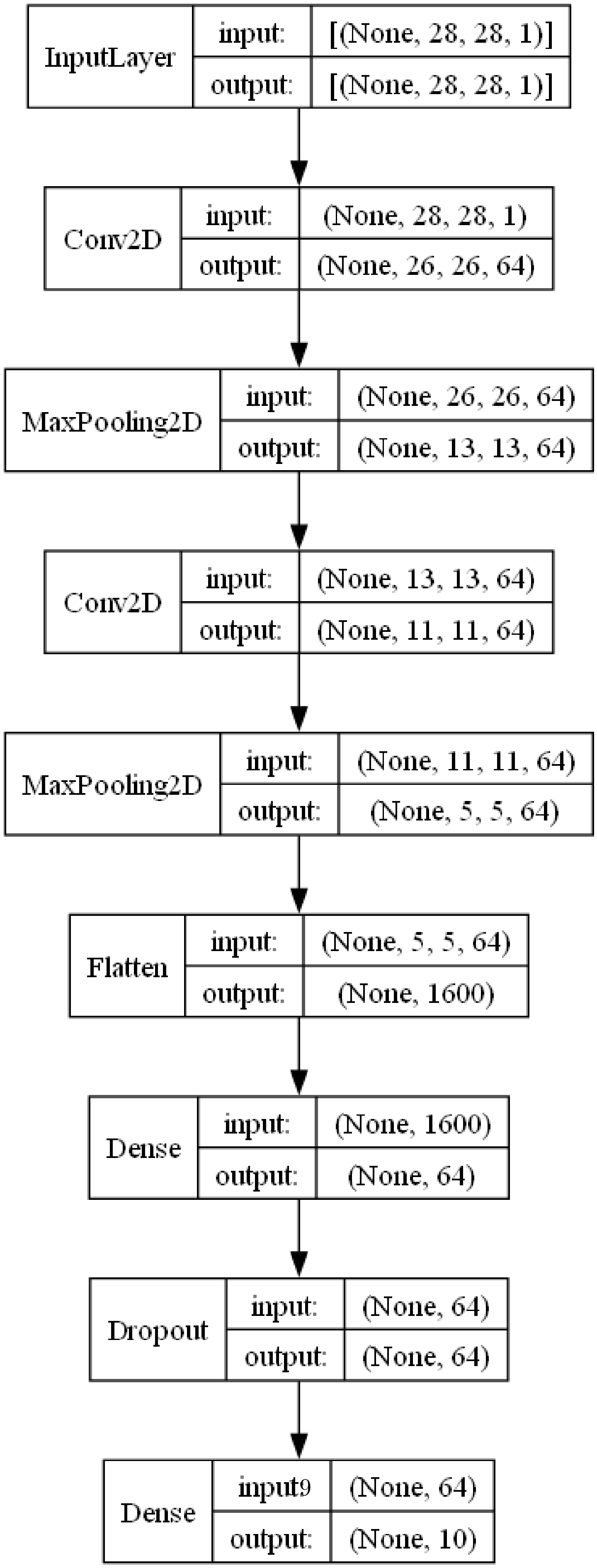
```
# Dropout layer, 0.2 rate
model_3.add(Dropout(rate=0.2))
# Final output dense Layer
model_3.add(Dense(10, activation='softmax'))

#Compile the model with sparse_categorical_crossentropy loss
model_3.
↪compile(loss='sparse_categorical_crossentropy',optimizer='adam',metrics=['accuracy'])
```

[39]:
```
model_3.build()
model_3.summary()
```

```
Model: "sequential_3"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_6 (Conv2D)           (None, 26, 26, 64)        640

 max_pooling2d_6 (MaxPoolin  (None, 13, 13, 64)        0
 g2D)

 conv2d_7 (Conv2D)           (None, 11, 11, 64)        36928

 max_pooling2d_7 (MaxPoolin  (None, 5, 5, 64)          0
 g2D)

 flatten_3 (Flatten)         (None, 1600)              0

 dense_6 (Dense)             (None, 64)                102464

 dropout_2 (Dropout)         (None, 64)                0

 dense_7 (Dense)             (None, 10)                650

=================================================================
Total params: 140682 (549.54 KB)
Trainable params: 140682 (549.54 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

[40]:
```
plot_model(model_3, show_shapes=True, show_layer_names=False)
```

[40]:


InputLayer
input:
[(None, 28, 28, 1)]
output:
[(None, 28, 28, 1)]
Conv2D
input:
(None, 28, 28, 1)
output:
(None, 26, 26, 64)
MaxPooling2D
input:
(None, 26, 26, 64)
output:
(None, 13, 13, 64)
Conv2D
input:
(None, 13, 13, 64)
output:
(None, 11, 11, 64)
MaxPooling2D
input:
(None, 11, 11, 64)
output:
(None, 5, 5, 64)
Flatten
input:
(None, 5, 5, 64)
output:
(None, 1600)
Dense
input:
(None, 1600)
output:
(None, 64)
Dropout
input:
(None, 64)
output:
(None, 64)
Dense
input9
(None, 64)
output:
(None, 10)

[41]:
```
batch_size=128
epochs=10
hist_3 = model_3.fit(X_train,␣
↪y_train,epochs=epochs,batch_size=batch_size,verbose=1,validation_split=0.2)
```

```
Epoch 1/10
375/375 [==============================] - 13s 31ms/step - loss: 0.2943 -
accuracy: 0.9091 - val_loss: 0.0794 - val_accuracy: 0.9766
Epoch 2/10
375/375 [==============================] - 11s 29ms/step - loss: 0.0883 -
accuracy: 0.9733 - val_loss: 0.0597 - val_accuracy: 0.9829
Epoch 3/10
375/375 [==============================] - 11s 28ms/step - loss: 0.0636 -
accuracy: 0.9806 - val_loss: 0.0492 - val_accuracy: 0.9852
Epoch 4/10
375/375 [==============================] - 11s 28ms/step - loss: 0.0505 -
accuracy: 0.9842 - val_loss: 0.0441 - val_accuracy: 0.9870
Epoch 5/10
375/375 [==============================] - 11s 28ms/step - loss: 0.0423 -
accuracy: 0.9864 - val_loss: 0.0385 - val_accuracy: 0.9888
Epoch 6/10
375/375 [==============================] - 11s 28ms/step - loss: 0.0350 -
accuracy: 0.9891 - val_loss: 0.0368 - val_accuracy: 0.9892
Epoch 7/10
375/375 [==============================] - 11s 29ms/step - loss: 0.0287 -
accuracy: 0.9908 - val_loss: 0.0444 - val_accuracy: 0.9883
Epoch 8/10
375/375 [==============================] - 11s 28ms/step - loss: 0.0266 -
accuracy: 0.9913 - val_loss: 0.0383 - val_accuracy: 0.9894
Epoch 9/10
375/375 [==============================] - 10s 27ms/step - loss: 0.0215 -
accuracy: 0.9931 - val_loss: 0.0373 - val_accuracy: 0.9896
Epoch 10/10
375/375 [==============================] - 10s 28ms/step - loss: 0.0205 -
accuracy: 0.9931 - val_loss: 0.0422 - val_accuracy: 0.9900
```

[42]:
```
score_3 = model_3.evaluate(X_test, y_test, verbose = 0)
print('Test loss:', score_3[0])
print('Test accuracy:', score_3[1])
```

```
Test loss: 0.03154883161187172
Test accuracy: 0.9904999732971191
```

[44]:
```
yhat_test = np.argmax(model_3.predict(X_test),axis=-1)
print(yhat_test[0:10])
```

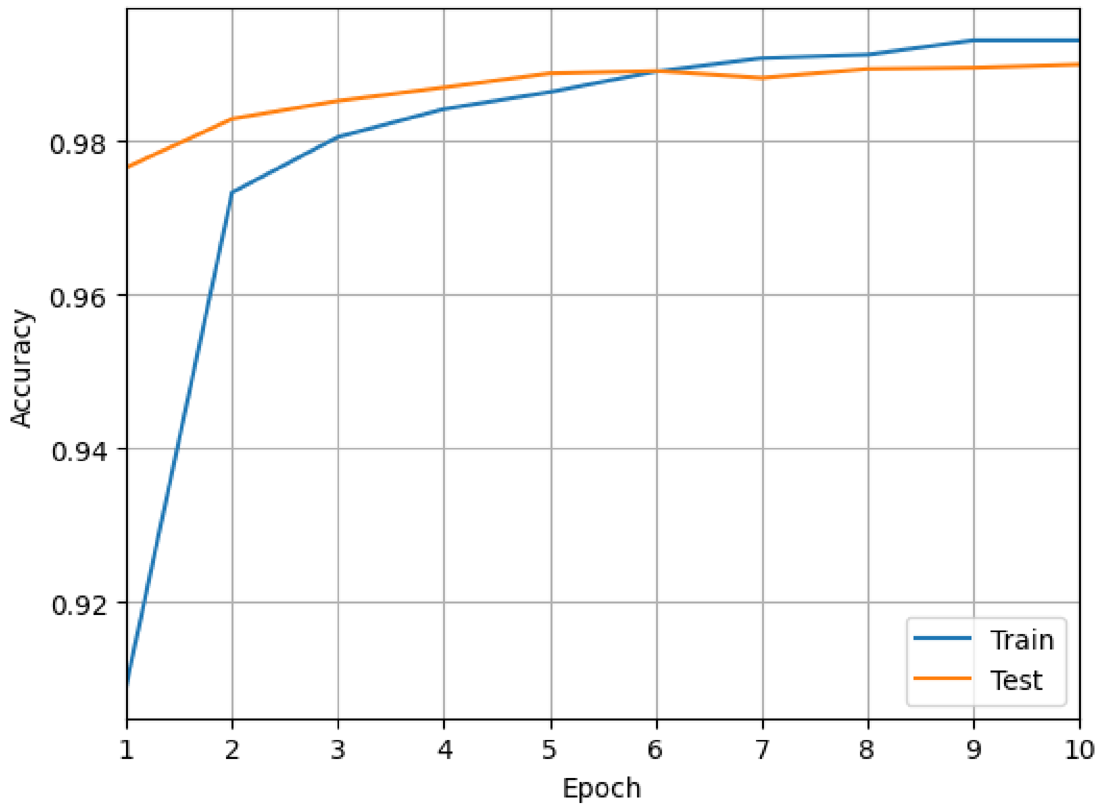
```
print(y_test[0:10])
print('Confusion Matrix:')
print(confusion_matrix(y_test, yhat_test))
```

```
313/313 [==============================] - 1s 4ms/step
[7 2 1 0 4 1 4 9 5 9]
[7 2 1 0 4 1 4 9 5 9]
Confusion Matrix:
[[ 977    0    1    0    0    0    0    1    1    0]
 [   0 1131    0    2    0    0    0    0    2    0]
 [   2    0 1021    3    0    0    0    5    1    0]
 [   0    0    1 1006    0    1    0    0    2    0]
 [   1    1    0    0  968    0    5    1    1    5]
 [   2    0    0   10    0  871    1    1    6    1]
 [   5    3    0    1    1    1  941    0    6    0]
 [   0    1    3    0    0    0    0 1022    0    2]
 [   1    1    0    2    0    0    0    2  968    0]
 [   2    2    0    0    1    1    1    1    1 1000]]
```

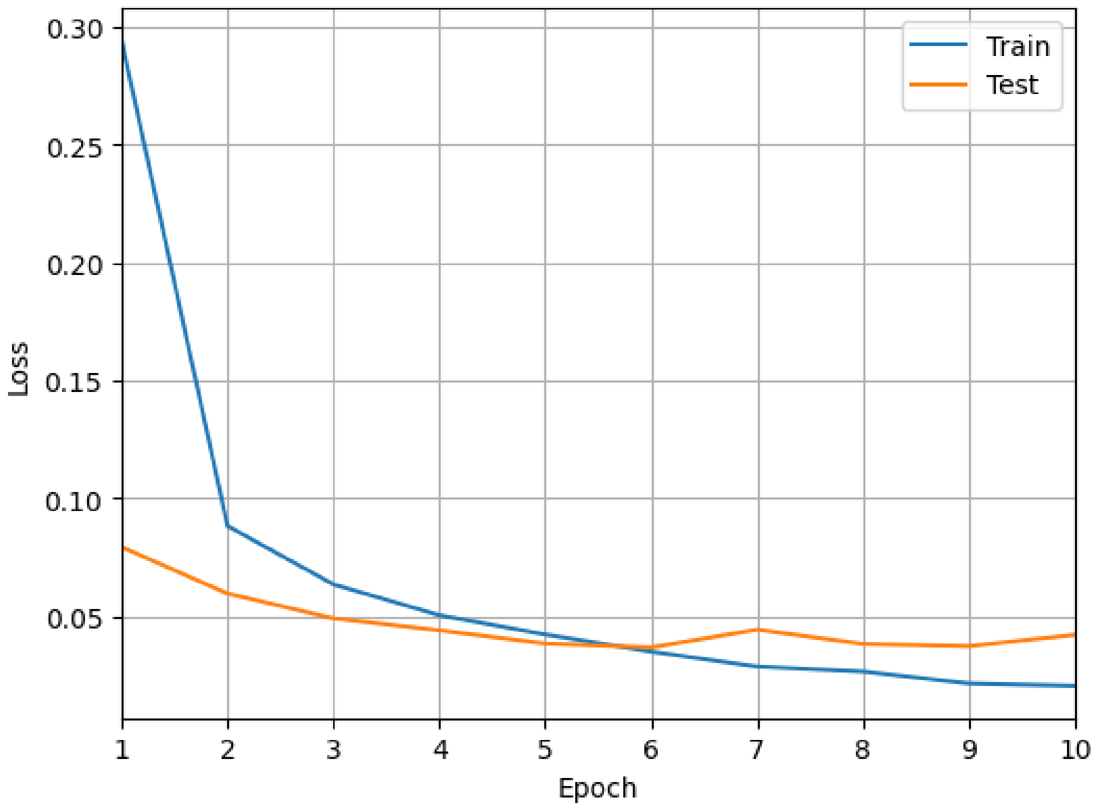
[47]:
```
print('Accuracy:')
print(float(accuracy_score(y_test, yhat_test))*100,'%')
```

```
Accuracy:
99.05000000000001 %
```

[45]:
```
plt.plot(epochRange,hist_3.history['accuracy'])
plt.plot(epochRange,hist_3.history['val_accuracy'])
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.grid()
plt.xlim((1,epochs))
plt.legend(['Train','Test'])
plt.show()
```



[46]:
```
plt.plot(epochRange,hist_3.history['loss'])
plt.plot(epochRange,hist_3.history['val_loss'])
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.grid()
plt.xlim((1,epochs))
plt.legend(['Train','Test'])
plt.show()
```



3.4 Changing kernel-size, stride and padding in Convolutional layer

[49]:
```
model_4 = Sequential() # Linear stacking of layers

# Convolution Layer 1: 8 filters, kernel size 2x2, relu activation, same␣
 ↪padding, stride 2
model_4.add(Conv2D(8,kernel_size=(2,2),activation='relu',strides=2,␣
 ↪padding='same',input_shape=(28, 28, 1)))
# MaxPooling: pool size 2, stride 2
model_4.add(MaxPooling2D(pool_size=2,strides=2))
# Convolution Layer 2: 16 filters, kernel size 2x2, relu activation, same␣
 ↪padding, stride 2
model_4.add(Conv2D(16,kernel_size=(2,2),activation='relu',strides=2,␣
 ↪padding='same'))
# MaxPooling: pool size 2, stride 2
model_4.add(MaxPooling2D(pool_size=2,strides=2))
# Flatten final feature matrix into a 1d array
model_4.add(Flatten(input_shape=(28,28)))
# Fully Connected Layer: 64 units and relu activation
model_4.add(Dense(64,activation='relu'))
```

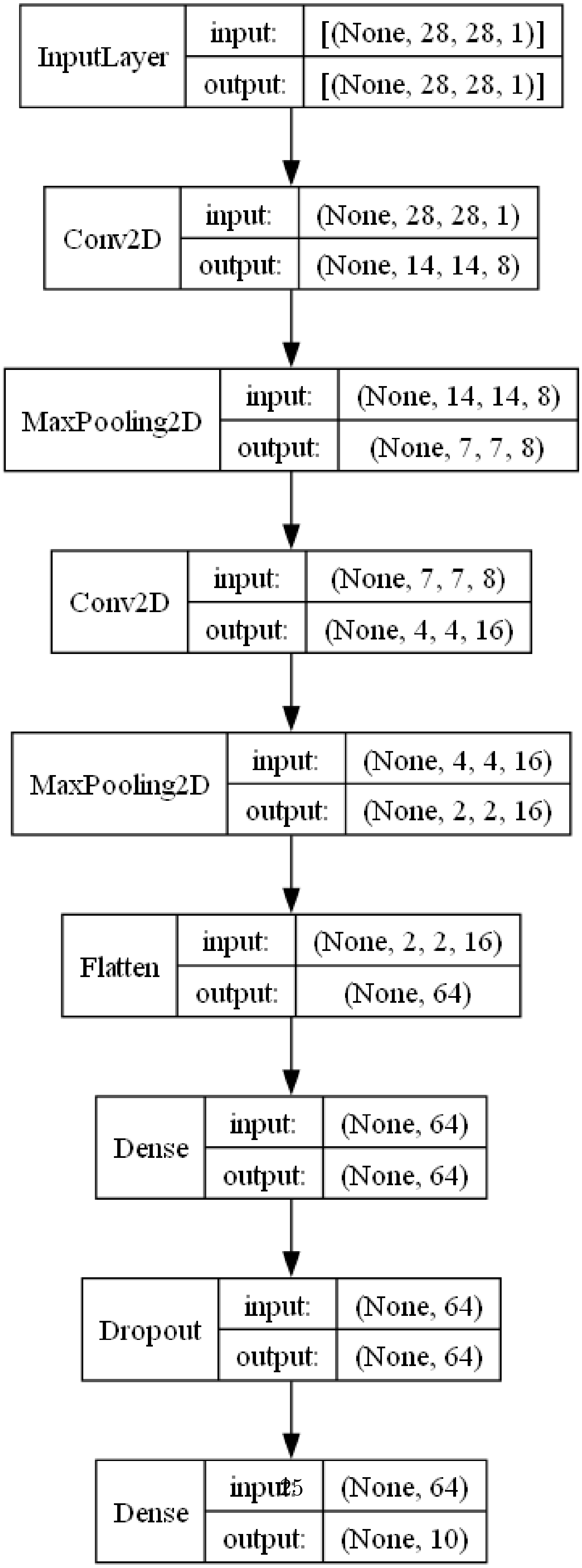
```
# Dropout layer, 0.2 rate
model_4.add(Dropout(rate=0.2))
# Final output dense Layer
model_4.add(Dense(10, activation='softmax'))

#Compile the model with sparse_categorical_crossentropy loss
model_4.
↪compile(loss='sparse_categorical_crossentropy',optimizer='adam',metrics=['accuracy'])
```

[50]:
```
model_4.build()
model_4.summary()
```

```
Model: "sequential_4"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_8 (Conv2D)           (None, 14, 14, 8)         40

 max_pooling2d_8 (MaxPoolin  (None, 7, 7, 8)           0
 g2D)

 conv2d_9 (Conv2D)           (None, 4, 4, 16)          528

 max_pooling2d_9 (MaxPoolin  (None, 2, 2, 16)          0
 g2D)

 flatten_4 (Flatten)         (None, 64)                0

 dense_8 (Dense)             (None, 64)                4160

 dropout_3 (Dropout)         (None, 64)                0

 dense_9 (Dense)             (None, 10)                650

=================================================================
Total params: 5378 (21.01 KB)
Trainable params: 5378 (21.01 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

[51]:
```
plot_model(model_4, show_shapes=True, show_layer_names=False)
```

[51]:


InputLayer
input:
[(None, 28, 28, 1)]
output:
[(None, 28, 28, 1)]
Conv2D
input:
(None, 28, 28, 1)
output:
(None, 14, 14, 8)
MaxPooling2D
input:
(None, 14, 14, 8)
output:
(None, 7, 7, 8)
Conv2D
input:
(None, 7, 7, 8)
output:
(None, 4, 4, 16)
MaxPooling2D
input:
(None, 4, 4, 16)
output:
(None, 2, 2, 16)
Flatten
input:
(None, 2, 2, 16)
output:
(None, 64)
Dense
input:
(None, 64)
output:
(None, 64)
Dropout
input:
(None, 64)
output:
(None, 64)
Dense
input:
(None, 64)
output:
(None, 10)

[52]:
```
batch_size=128
epochs=10
hist_4 = model_4.fit(X_train,␣
↪y_train,epochs=epochs,batch_size=batch_size,verbose=1,validation_split=0.2)
```

```
Epoch 1/10

375/375 [==============================] - 4s 6ms/step - loss: 1.1088 -
accuracy: 0.6406 - val_loss: 0.5164 - val_accuracy: 0.8395
Epoch 2/10
375/375 [==============================] - 2s 5ms/step - loss: 0.5424 -
accuracy: 0.8238 - val_loss: 0.3939 - val_accuracy: 0.8745
Epoch 3/10
375/375 [==============================] - 2s 6ms/step - loss: 0.4458 -
accuracy: 0.8585 - val_loss: 0.3352 - val_accuracy: 0.8943
Epoch 4/10
375/375 [==============================] - 2s 6ms/step - loss: 0.3961 -
accuracy: 0.8744 - val_loss: 0.2988 - val_accuracy: 0.9057
Epoch 5/10
375/375 [==============================] - 2s 6ms/step - loss: 0.3610 -
accuracy: 0.8871 - val_loss: 0.2859 - val_accuracy: 0.9097
Epoch 6/10
375/375 [==============================] - 2s 5ms/step - loss: 0.3336 -
accuracy: 0.8934 - val_loss: 0.2621 - val_accuracy: 0.9184
Epoch 7/10
375/375 [==============================] - 2s 5ms/step - loss: 0.3118 -
accuracy: 0.9010 - val_loss: 0.2439 - val_accuracy: 0.9230
Epoch 8/10
375/375 [==============================] - 2s 4ms/step - loss: 0.2958 -
accuracy: 0.9053 - val_loss: 0.2286 - val_accuracy: 0.9288
Epoch 9/10
375/375 [==============================] - 2s 4ms/step - loss: 0.2810 -
accuracy: 0.9103 - val_loss: 0.2243 - val_accuracy: 0.9302
Epoch 10/10
375/375 [==============================] - 2s 5ms/step - loss: 0.2652 -
accuracy: 0.9161 - val_loss: 0.2110 - val_accuracy: 0.9352
```

[53]:
```
score_4 = model_4.evaluate(X_test, y_test, verbose = 0)
print('Test loss:', score_4[0])
print('Test accuracy:', score_4[1])
```

```
Test loss: 0.20662404596805573
Test accuracy: 0.9318000078201294
```

[54]:
```
yhat_test = np.argmax(model_4.predict(X_test),axis=-1)
print(yhat_test[0:10])
```

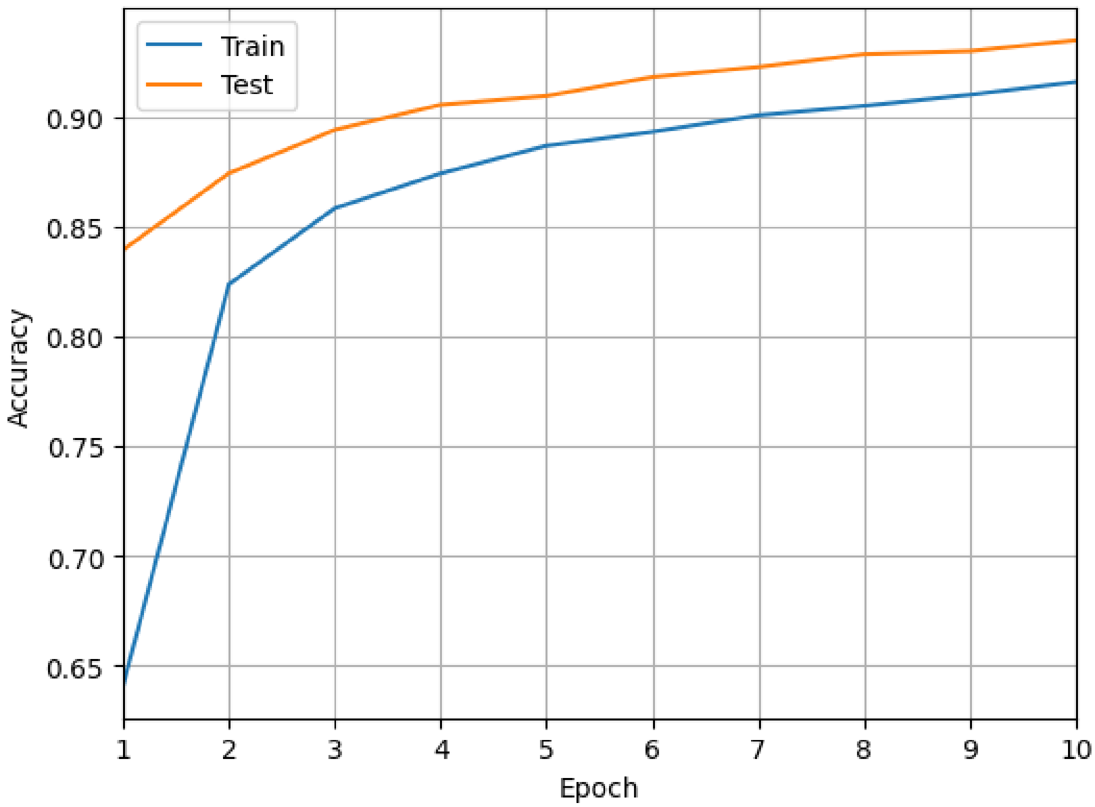
```
print(y_test[0:10])
print('Confusion Matrix:')
print(confusion_matrix(y_test, yhat_test))
```

```
313/313 [==============================] - 1s 3ms/step
[7 2 1 0 4 1 4 9 5 9]
[7 2 1 0 4 1 4 9 5 9]
Confusion Matrix:
[[ 947    1    4    3    0    5   11    5    4    0]
 [    0 1110    7    5    0    1    5    2    5    0]
 [  14    1  973   10    2    0    5   17   10    0]
 [    2    2   21  916    0   36    2    9   20    2]
 [    3    2    2    1  911    2   12    6    4   39]
 [    8    0    1   39    0  816   17    2    4    5]
 [  15    2    5    0    3   13  919    0    1    0]
 [    3    5   19    6    8    0    0  964    6   17]
 [  10    1    8   21    2   27   10    4  879   12]
 [    6    7    4    8   58   17    3   17    6  883]]
```

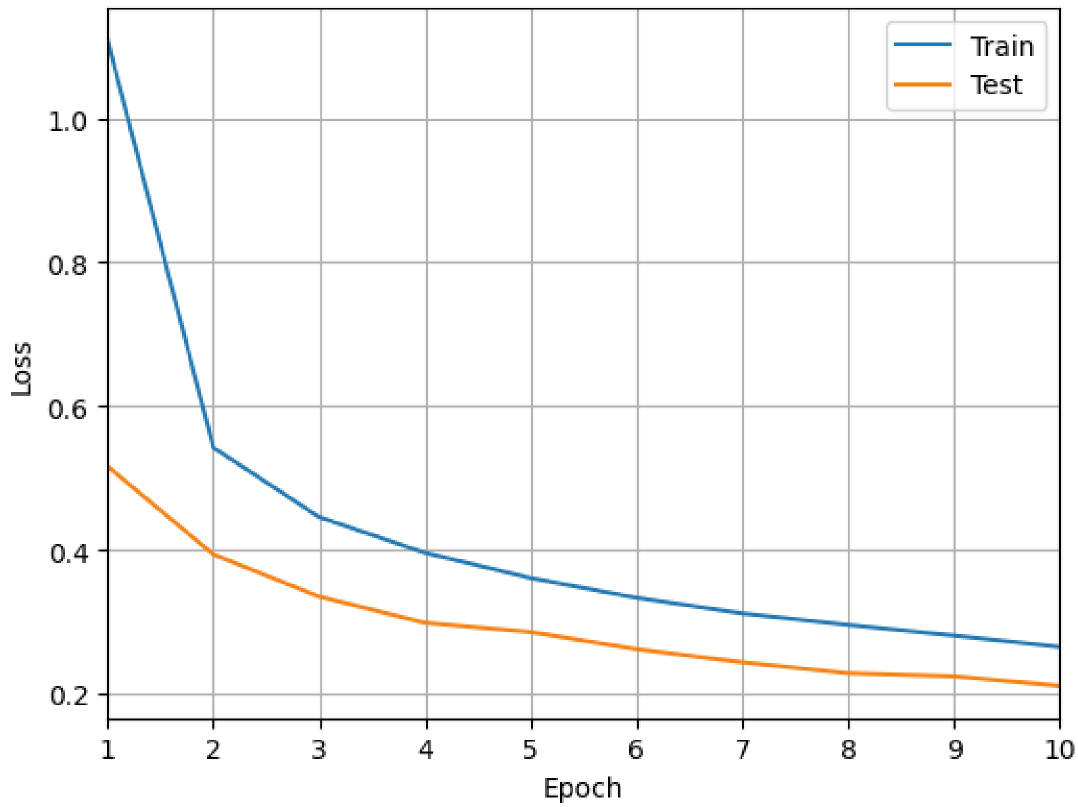
[55]:
```
print('Accuracy:')
print(float(accuracy_score(y_test, yhat_test))*100,'%')
```

```
Accuracy:
93.17999999999999 %
```

[56]:
```
plt.plot(epochRange,hist_4.history['accuracy'])
plt.plot(epochRange,hist_4.history['val_accuracy'])
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.grid()
plt.xlim((1,epochs))
plt.legend(['Train','Test'])
plt.show()
```



[57]:
```
plt.plot(epochRange,hist_4.history['loss'])
plt.plot(epochRange,hist_4.history['val_loss'])
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.grid()
plt.xlim((1,epochs))
plt.legend(['Train','Test'])
plt.show()
```



4 Conclusion

A Convolutional Neural Network architecture was designed to implement the image classification task over the MNIST image dataset. Hyper-parameter tuning was performed and results were recorded.

The model with dropout rate set to 0.2 has the accuracy 98.89%.

The model with no dropout layer has the accuracy 98.79%.

The model with increased filters in Conv2D layers has the accuracy 99.05%.

The model with kernel-size set to (2,2), stride = 2 and same padding has the accuracy 93.18%.