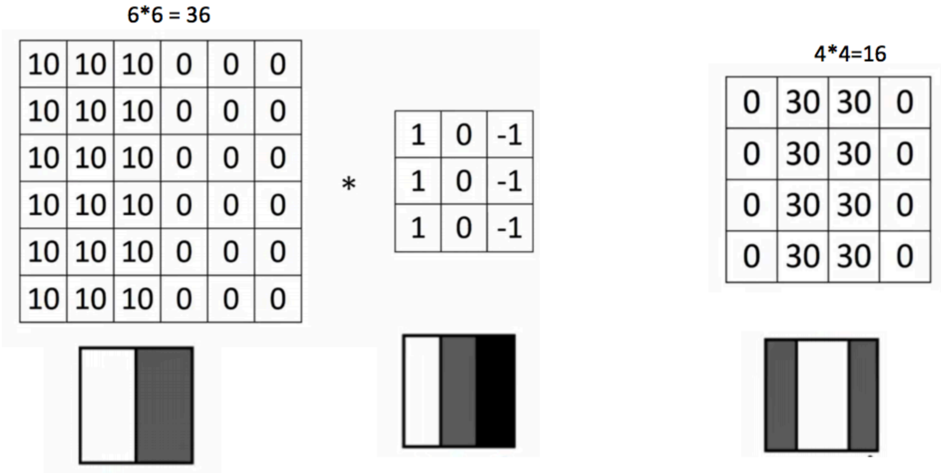
Image Classification using CNN

CNN stands for Convolutionary Neural Network. Convolution stands for a pointwise multiplication of 2 functions to produce a third function. One of the functions here happens to be the image pixel matrix and another is the filter which shall be imposed upon the image. The dot product of the 2 function matrices causes the creation of an ‘Activation Map’ or a ‘Feature Map’.



CNN has the ability to learn hierarchical features like edges, textures and shapes enabling accurate object detection. It is also capable of extracting meaningful spatial features from images. CNN consists of various layers:

Input Layer:

This layer takes raw images as input and are represented in the form of matrices of pixel values. Dimensions of input layers correspond to size of input images, i.e., height, width and color channels.

Convolution Layer:

This layer is responsible for feature extraction. It consists of features, also known as kernels, which are convolved with the raw image inputs to capture relevant patterns and features. These layers learn to detect important edges, textures, shapes, layers and features. It provides with an output in the form of feature maps.

Pooling Layers:

This layer is responsible for reducing the spatial dimensions of the feature maps created by the convolution layer. It is responsible for performing down-sampling operations to retain the most salient features and information while discarding the unnecessary details. This helps in achieving translational invariance and reduced computational complexity.

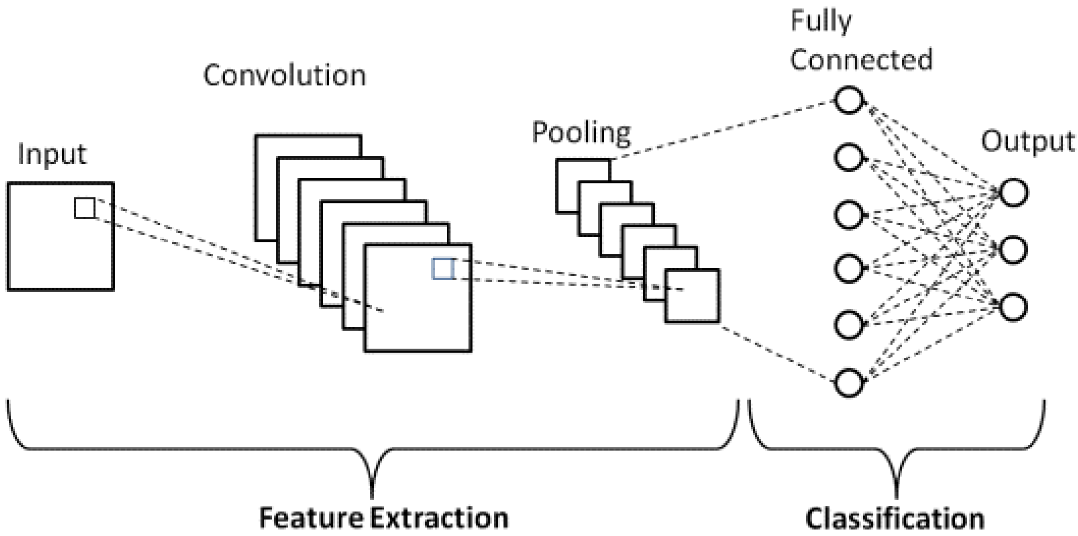
- **Translational Invariance:** Property of a system or function that remains unchanged when it undergoes a translation (shift) in some specific direction. Eg.: • A translation-invariant system can recognize an object (e.g., a face) regardless of its position within the image.

Fully Connected Layers:

The output of the last pooling layers is flattened and connected to one or more fully connected layers. These fully connected layers act as **Traditional Neural Network Layers** and help in classification of the features extracted from the image. The role of the fully connected layers is to learn complex relationships between features and provide output in the form of class probabilities and predictions.

Output Layer:

This is the final layer of CNN and consists of neurons equal to the number of distinct classes in the classification task. The output layer provides each class's classification probabilities or predictions, indicating the likelihood of the input image belonging to a



In [1]:

```
import numpy as np
import keras
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense,Flatten
from keras.layers import Dropout
from keras.layers import Flatten
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
from keras import backend as K
import matplotlib.pyplot as plt
```

```
WARNING:tensorflow:From C:\Users\Admin\anaconda3\Lib\site-packages\keras\s
rc\losses.py:2976: The name tf.losses.sparse_softmax_cross_entropy is depr
ecated. Please use tf.compat.v1.losses.sparse_softmax_cross_entropy instea
d.
```

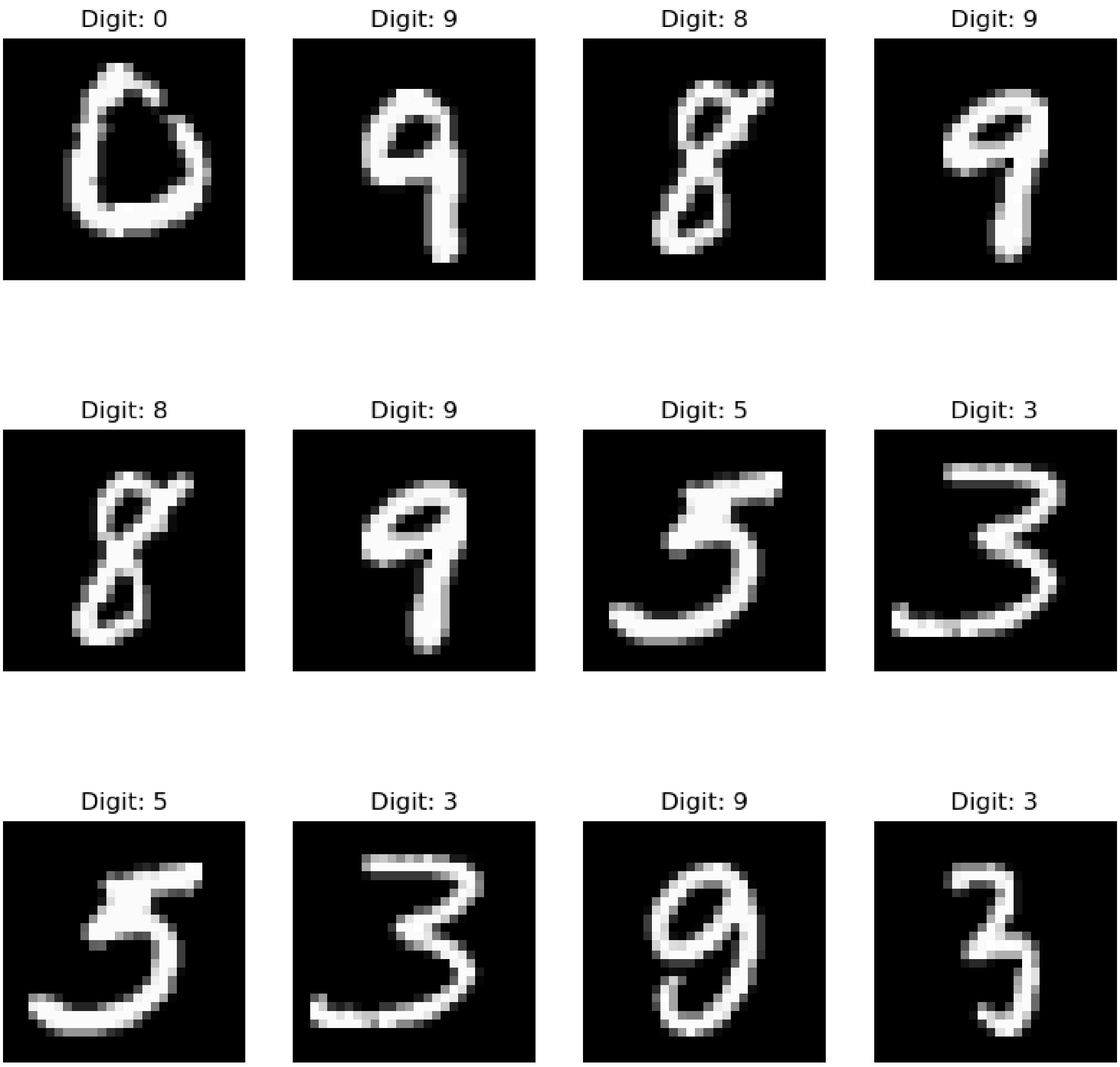
In [26]:

```
(x_train,y_train), (x_test,y_test) = mnist.load_data()
```

In [27]:
```
import random
n=7
fig, axs = plt.subplots(nrows=3, ncols=4, figsize=(10,10))
indices = random.sample(range(len(x_train)), 12)
images = x_train[indices]
labels = y_train[indices]
for i in range(3):
    for j in range(4):
        index = i * 2 + j
        image = images[index].reshape((28, 28))  # Reshape the image to
        axs[i, j].imshow(image, cmap='gray')  # Plot the image
        axs[i, j].set_title(f"Digit: {labels[index]}")  # Add title wit
        axs[i, j].axis('off')  # Hide axis ticks and labels

print(y_train[n])
```

```
3
```



In [4]:
```
x_train.shape, y_train.shape
```

Out[4]:
```
((60000, 28, 28), (60000,))
```

In [5]:
```
x_test.shape, y_test.shape
```

Out[5]:
```
((10000, 28, 28), (10000,))
```

Hence, we can see here that the image size is (28x28) and the training dataset consists of 60,000 images to train the model. The testing dataset consists of 10,000 images of Handwritten digits.

To train the neural network with the image dataset, we require a 3 dimensional image and all the images given in the dataset are of 2 dimensions only. Hence, we use the reshape property to change the dimension of all the images to 3 dimensional images, i.e., (28x28x1)

In [29]:
```
x_train = x_train.reshape(x_train.shape[0], 28,28,1)
x_test = x_test.reshape(x_test.shape[0], 28, 28, 1)
```

In [30]:
```
y_train = keras.utils.to_categorical(y_train, 10)
y_test = keras.utils.to_categorical(y_test, 10)
```

In [31]:
```
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
```

In [32]:
```
x_train /= 255
x_test /= 255
```

Creating the Model:

The `model = Sequential()` is used to simply define a private Convolutionary Neural Network where various layers can be added by using the `add()` method. It provides us with basic features like:

1. Accessing individual layers using indexing
2. Compiling entire model for training
3. Evaluating Model's Performance

1. Conv2D(): It stands for Convolutional 2D layer. It extracts the spatial patterns and features from the images and outputs a feature map which consists of extracted features from the image.
2. MaxPooling2D(): It represents the Pooling Layer and reduce the dimensionality of data while retaining the spatial features extracted by the Convolutional layer. In other words, it performs the downsampling operation.
3. Dropout(): It is used to solve the problem of over fitting where the model performs well on training data but under-performs on the testing data leading to lower accuracies. It is a regularization technique.
4. Flatten(): It reshapes the multi-dimensional input into a single-dimensional vector. This allows the data to be fed into fully-connected layers, which require 1D inputs.
5. Dense(): It represents the Fully Connected Layer and connects each neuron in one layer to each neuron in the next layer, performs linear transformations based on weights and biases associated to each neuron and transforms the data's dimensionality to increase its feature complexity and adapt to the desired output.

In [10]:
```
batch_size = 128
num_classes = 10
epochs = 50
input_shape = (28, 28, 1)
model = Sequential()
model.add(Conv2D(32,kernel_size=(3,3),activation='relu',input_shape=inp
model.add(Conv2D(64, (3,3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))
model.compile(loss=keras.losses.categorical_crossentropy,optimizer=kera
```

```
WARNING:tensorflow:From C:\Users\Admin\anaconda3\Lib\site-packages\keras\s
rc\backend.py:873: The name tf.get_default_graph is deprecated. Please use
tf.compat.v1.get_default_graph instead.

WARNING:tensorflow:From C:\Users\Admin\anaconda3\Lib\site-packages\keras\s
rc\layers\pooling\max_pooling2d.py:161: The name tf.nn.max_pool is depreca
ted. Please use tf.nn.max_pool2d instead.
```

Training the Model

The `epochs` value is used in Deep Learning CNN models wherein it denotes the number of times the entire dataset of images, i.e. `mnist dataset` is being passed throughout the entire model. We are also trying to print the training loss(loss) and validation_loss(val_loss), training accuracy(accuracy) and validation_accuracy(val_accuracy) here. The conditions for finding if the model is over-fitting or under-fitting is as follows:

1. If validation_loss >> training loss ; then the model is **Overfitting**.
2. If validation_loss > training_loss ; then the model is **slightly Overfitting**.
3. If validation_loss < training_loss ; then the model is **slightly Underfitting**.
4. If validation_loss << training_loss ; then the model is **Underfitting**.

In [20]:
```
hist = model.fit(x_train, y_train, batch_size=batch_size, epochs=epochs
print("The model has been trained Successfully.")
model.save('mnist.h5')
print("Saving the model as 'mnist.h5'")
```

```
- accuracy: 0.8816 - val_loss: 0.2730 - val_accuracy: 0.9210
Epoch 47/50
469/469 [==============================] - 39s 84ms/step - loss: 0.3858
- accuracy: 0.8840 - val_loss: 0.2703 - val_accuracy: 0.9211
Epoch 48/50
469/469 [==============================] - 39s 84ms/step - loss: 0.3839
- accuracy: 0.8839 - val_loss: 0.2677 - val_accuracy: 0.9212
Epoch 49/50
469/469 [==============================] - 40s 84ms/step - loss: 0.3819
- accuracy: 0.8832 - val_loss: 0.2652 - val_accuracy: 0.9224
Epoch 50/50
469/469 [==============================] - 39s 84ms/step - loss: 0.3787
- accuracy: 0.8850 - val_loss: 0.2629 - val_accuracy: 0.9228
The model has been trained Successfully.
Saving the model as 'mnist.h5'

C:\Users\Admin\anaconda3\Lib\site-packages\keras\src\engine\training.p
y:3103: UserWarning: You are saving your model as an HDF5 file via `mod
el.save()`. This file format is considered legacy. We recommend using i
nstead the native Keras format, e.g. `model.save('my_model.keras')`.
```

Evaluating the Model: mnist.h5

In [21]:
```
score = model.evaluate(x_test, y_test, verbose=0)
print("Test Loss = ", score[0])
print("Test Accuracy = ", score[1])
```

```
Test Loss =  0.26294204592704773
Test Accuracy =  0.9228000044822693
```

Model with epoch=20:

In [12]:
```
batch_size = 128
num_classes = 10
epochs = 50
input_shape = (28, 28, 1)
model1 = Sequential()
model1.add(Conv2D(32,kernel_size=(3,3),activation='relu',input_shape=ir
model1.add(Conv2D(64, (3,3), activation='relu'))
model1.add(MaxPooling2D(pool_size=(2,2)))
model1.add(Dropout(0.25))
model1.add(Flatten())
model1.add(Dense(256, activation='relu'))
model1.add(Dropout(0.5))
model1.add(Dense(num_classes, activation='softmax'))
model1.compile(loss=keras.losses.categorical_crossentropy,optimizer=ker
```

In [13]:
```
hist = model1.fit(x_train, y_train, batch_size=batch_size, epochs=20, v
print("The model has been trained Successfully.")
model1.save('mnist_epoch20.h5')
print("Saving the model as 'mnist_epoch20.h5'")
```

```
- accuracy: 0.8128 - val_loss: 0.4415 - val_accuracy: 0.8844
Epoch 17/20
469/469 [==============================] - 35s 75ms/step - loss: 0.5936
- accuracy: 0.8177 - val_loss: 0.4251 - val_accuracy: 0.8868
Epoch 18/20
469/469 [==============================] - 36s 78ms/step - loss: 0.5735
- accuracy: 0.8236 - val_loss: 0.4104 - val_accuracy: 0.8911
Epoch 19/20
469/469 [==============================] - 35s 75ms/step - loss: 0.5604
- accuracy: 0.8302 - val_loss: 0.3976 - val_accuracy: 0.8948
Epoch 20/20
469/469 [==============================] - 35s 75ms/step - loss: 0.5404
- accuracy: 0.8345 - val_loss: 0.3861 - val_accuracy: 0.8970
The model has been trained Successfully.
Saving the model as 'mnist_epoch20.h5'

C:\Users\Admin\anaconda3\Lib\site-packages\keras\src\engine\training.p
y:3103: UserWarning: You are saving your model as an HDF5 file via `mod
el.save()`. This file format is considered legacy. We recommend using i
nstead the native Keras format, e.g. `model.save('my_model.keras')`.
```

mnist_epoch20 model evaluation:

In [14]:
```
score = model1.evaluate(x_test, y_test, verbose=0)
print("Test Loss = ", score[0])
print("Test Accuracy = ", score[1])
```

```
Test Loss =  0.38610392808914185
Test Accuracy =  0.8970000147819519
```

Model with epoch=70:

In [17]:
```
batch_size = 128
num_classes = 10
epochs = 50
input_shape = (28, 28, 1)
model2 = Sequential()
model2.add(Conv2D(32,kernel_size=(3,3),activation='relu',input_shape=ir
model2.add(Conv2D(64, (3,3), activation='relu'))
model2.add(MaxPooling2D(pool_size=(2,2)))
model2.add(Dropout(0.25))
model2.add(Flatten())
model2.add(Dense(256, activation='relu'))
model2.add(Dropout(0.5))
model2.add(Dense(num_classes, activation='softmax'))
model2.compile(loss=keras.losses.categorical_crossentropy,optimizer=ker
```

In [18]:
```
hist = model2.fit(x_train, y_train, batch_size=batch_size, epochs=70, v
print("The model has been trained Successfully.")
model2.save('mnist_epoch70.h5')
print("Saving the model as 'mnist_epoch70.h5'")
```

```
Epoch 65/70
469/469 [==============================] - 40s 86ms/step - loss: 0.3263
- accuracy: 0.9007 - val_loss: 0.2231 - val_accuracy: 0.9346
Epoch 66/70
469/469 [==============================] - 39s 83ms/step - loss: 0.3238
- accuracy: 0.9024 - val_loss: 0.2215 - val_accuracy: 0.9350
Epoch 67/70
469/469 [==============================] - 40s 85ms/step - loss: 0.3212
- accuracy: 0.9014 - val_loss: 0.2200 - val_accuracy: 0.9352
Epoch 68/70
469/469 [==============================] - 39s 83ms/step - loss: 0.3184
- accuracy: 0.9033 - val_loss: 0.2185 - val_accuracy: 0.9358
Epoch 69/70
469/469 [==============================] - 39s 83ms/step - loss: 0.3203
- accuracy: 0.9026 - val_loss: 0.2172 - val_accuracy: 0.9360
Epoch 70/70
469/469 [==============================] - 39s 83ms/step - loss: 0.3156
- accuracy: 0.9037 - val_loss: 0.2156 - val_accuracy: 0.9362
The model has been trained Successfully.
Saving the model as 'mnist_epoch70.h5'
```

mnist_epoch70 model evaluation:

In [19]:
```
score = model2.evaluate(x_test, y_test, verbose=0)
print("Test Loss = ", score[0])
print("Test Accuracy = ", score[1])
```

```
Test Loss =  0.21564091742038727
Test Accuracy =  0.9362000226974487
```

Trying to change the make-up of the layers of CNN in order to improve the accuracy with (4x4) Conv2D layer:

In [17]:
```
model4 = Sequential()
model4.add(Conv2D(32, (3,3), activation='relu', input_shape=(28,28,1)))
model4.add(MaxPooling2D(pool_size=(2,2)))
model4.add(Conv2D(64, (4,4), activation='relu'))
model4.add(MaxPooling2D((2,2)))
model4.add(Dropout(0.25))
model4.add(Flatten())
model4.add(Dense(256, activation='relu'))
model4.add(Dropout(0.5))
model4.add(Dense(10, activation='softmax'))
model4.compile(loss=keras.losses.categorical_crossentropy,optimizer=ker
```

Model on (4x4) kernel selection with 20 epochs:

In [19]:
```
hist = model4.fit(x_train, y_train, batch_size=batch_size, epochs=20,
print("The model has been trained Successfully.")
model4.save('new_mnist(4x4)_epoch20.h5')
print("Saving the model as 'new_mnist(4x4)_epoch20.h5'")
```

```
- accuracy: 0.6005 - val_loss: 1.3085 - val_accuracy: 0.8070
Epoch 17/20
469/469 [==============================] - 26s 56ms/step - loss: 1.4339
- accuracy: 0.6115 - val_loss: 1.2178 - val_accuracy: 0.8143
Epoch 18/20
469/469 [==============================] - 26s 56ms/step - loss: 1.3611
- accuracy: 0.6257 - val_loss: 1.1321 - val_accuracy: 0.8206
Epoch 19/20
469/469 [==============================] - 27s 57ms/step - loss: 1.2916
- accuracy: 0.6406 - val_loss: 1.0536 - val_accuracy: 0.8270
Epoch 20/20
469/469 [==============================] - 26s 56ms/step - loss: 1.2273
- accuracy: 0.6559 - val_loss: 0.9818 - val_accuracy: 0.8318
The model has been trained Successfully.
Saving the model as 'new_mnist(4x4)_epoch20.h5'

C:\Users\Admin\anaconda3\Lib\site-packages\keras\src\engine\training.p
y:3103: UserWarning: You are saving your model as an HDF5 file via `mod
el.save()`. This file format is considered legacy. We recommend using i
nstead the native Keras format, e.g. `model.save('my_model.keras')`.
```

Model on (4x4) kernel selection with 50 epochs:

In [23]:
```
hist = model4.fit(x_train, y_train, batch_size=batch_size, epochs=50,
print("The model has been trained Successfully.")
model4.save('new_mnist(4x4)_epoch50.h5')
print("Saving the model as 'new_mnist(4x4)_epoch50.h5'")
```

```
Epoch 45/50
469/469 [==============================] - 26s 56ms/step - loss: 0.4308
- accuracy: 0.8692 - val_loss: 0.2683 - val_accuracy: 0.9292
Epoch 46/50
469/469 [==============================] - 26s 55ms/step - loss: 0.4243
- accuracy: 0.8714 - val_loss: 0.2652 - val_accuracy: 0.9301
Epoch 47/50
469/469 [==============================] - 26s 56ms/step - loss: 0.4191
- accuracy: 0.8720 - val_loss: 0.2619 - val_accuracy: 0.9306
Epoch 48/50
469/469 [==============================] - 26s 56ms/step - loss: 0.4155
- accuracy: 0.8725 - val_loss: 0.2589 - val_accuracy: 0.9313
Epoch 49/50
469/469 [==============================] - 26s 56ms/step - loss: 0.4132
- accuracy: 0.8759 - val_loss: 0.2560 - val_accuracy: 0.9318
Epoch 50/50
469/469 [==============================] - 26s 56ms/step - loss: 0.4076
- accuracy: 0.8760 - val_loss: 0.2532 - val_accuracy: 0.9319
The model has been trained Successfully.
Saving the model as 'new_mnist(4x4)_epoch50.h5'
```

Model on (4x4) kernel selection with 70 epochs:

In [33]:
```
hist = model4.fit(x_train, y_train, batch_size=batch_size, epochs=70,
print("The model has been trained Successfully.")
model4.save('new_mnist(4x4)_epoch70.h5')
print("Saving the model as 'new_mnist(4x4)_epoch70.h5'")
```

```
Epoch 65/70
469/469 [==============================] - 18s 39ms/step - loss: 0.2528
- accuracy: 0.9238 - val_loss: 0.1542 - val_accuracy: 0.9549
Epoch 66/70
469/469 [==============================] - 18s 39ms/step - loss: 0.2511
- accuracy: 0.9248 - val_loss: 0.1534 - val_accuracy: 0.9556
Epoch 67/70
469/469 [==============================] - 18s 39ms/step - loss: 0.2512
- accuracy: 0.9242 - val_loss: 0.1526 - val_accuracy: 0.9556
Epoch 68/70
469/469 [==============================] - 18s 39ms/step - loss: 0.2498
- accuracy: 0.9251 - val_loss: 0.1518 - val_accuracy: 0.9562
Epoch 69/70
469/469 [==============================] - 18s 39ms/step - loss: 0.2478
- accuracy: 0.9259 - val_loss: 0.1508 - val_accuracy: 0.9564
Epoch 70/70
469/469 [==============================] - 18s 39ms/step - loss: 0.2462
- accuracy: 0.9259 - val_loss: 0.1502 - val_accuracy: 0.9564
The model has been trained Successfully.
Saving the model as 'new_mnist(4x4)_epoch70.h5'
```

Trying to change the make-up of the layers of CNN in order to improve the accuracy with (2x2) Conv2D layer:

In [20]:
```
model5 = Sequential()
model5.add(Conv2D(32, (3,3), activation='relu', input_shape=(28,28,1)))
model5.add(MaxPooling2D(pool_size=(2,2)))
model5.add(Conv2D(64, (2,2), activation='relu'))
model5.add(MaxPooling2D((2,2)))
model5.add(Dropout(0.25))
model5.add(Flatten())
model5.add(Dense(256, activation='relu'))
model5.add(Dropout(0.5))
model5.add(Dense(10, activation='softmax'))
model5.compile(loss=keras.losses.categorical_crossentropy,optimizer=ker
```

Model on (2x2) kernel selection with 20 epochs:

In [21]:
```
hist = model5.fit(x_train, y_train, batch_size=batch_size, epochs=20, \
print("The model has been trained Successfully.")
model5.save('new_mnist(2x2)_epoch20.h5')
print("Saving the model as 'new_mnist(2x2)_epoch20.h5'")
```

```
Epoch 15/20
469/469 [==============================] - 22s 47ms/step - loss: 1.6960
- accuracy: 0.5578 - val_loss: 1.5587 - val_accuracy: 0.7473
Epoch 16/20
469/469 [==============================] - 22s 46ms/step - loss: 1.6257
- accuracy: 0.5732 - val_loss: 1.4762 - val_accuracy: 0.7569
Epoch 17/20
469/469 [==============================] - 22s 47ms/step - loss: 1.5596
- accuracy: 0.5843 - val_loss: 1.3948 - val_accuracy: 0.7627
Epoch 18/20
469/469 [==============================] - 22s 47ms/step - loss: 1.4903
- accuracy: 0.5960 - val_loss: 1.3156 - val_accuracy: 0.7700
Epoch 19/20
469/469 [==============================] - 22s 47ms/step - loss: 1.4304
- accuracy: 0.6065 - val_loss: 1.2409 - val_accuracy: 0.7747
Epoch 20/20
469/469 [==============================] - 22s 47ms/step - loss: 1.3693
- accuracy: 0.6165 - val_loss: 1.1711 - val_accuracy: 0.7787
The model has been trained Successfully.
Saving the model as 'new_mnist(2x2)_epoch20.h5'
```

Model on (2x2) kernel selection with 50 epochs:

In [22]:
```
hist = model5.fit(x_train, y_train, batch_size=batch_size, epochs=50, \
print("The model has been trained Successfully.")
model5.save('new_mnist(2x2)_epoch50.h5')
print("Saving the model as 'new_mnist(2x2)_epoch50.h5'")
```

```
Epoch 45/50
469/469 [==============================] - 22s 47ms/step - loss: 0.5910
- accuracy: 0.8137 - val_loss: 0.3988 - val_accuracy: 0.8919
Epoch 46/50
469/469 [==============================] - 22s 46ms/step - loss: 0.5861
- accuracy: 0.8167 - val_loss: 0.3947 - val_accuracy: 0.8933
Epoch 47/50
469/469 [==============================] - 22s 47ms/step - loss: 0.5767
- accuracy: 0.8185 - val_loss: 0.3905 - val_accuracy: 0.8946
Epoch 48/50
469/469 [==============================] - 22s 47ms/step - loss: 0.5721
- accuracy: 0.8214 - val_loss: 0.3862 - val_accuracy: 0.8966
Epoch 49/50
469/469 [==============================] - 22s 47ms/step - loss: 0.5708
- accuracy: 0.8207 - val_loss: 0.3826 - val_accuracy: 0.8967
Epoch 50/50
469/469 [==============================] - 22s 47ms/step - loss: 0.5652
- accuracy: 0.8228 - val_loss: 0.3788 - val_accuracy: 0.8981
The model has been trained Successfully.
Saving the model as 'new_mnist(2x2)_epoch50.h5'
```

Model on (2x2) kernel selection with 70 epochs:

In [34]:
```python
hist = model5.fit(x_train, y_train, batch_size=batch_size, epochs=70, v
print("The model has been trained Successfully.")
model5.save('new_mnist(2x2)_epoch70.h5')
print("Saving the model as 'new_mnist(2x2)_epoch70.h5'")
```

```
Epoch 65/70
469/469 [==============================] - 10s 21ms/step - loss: 0.3679
- accuracy: 0.8877 - val_loss: 0.2405 - val_accuracy: 0.9293
Epoch 66/70
469/469 [==============================] - 10s 21ms/step - loss: 0.3693
- accuracy: 0.8875 - val_loss: 0.2393 - val_accuracy: 0.9302
Epoch 67/70
469/469 [==============================] - 10s 21ms/step - loss: 0.3687
- accuracy: 0.8873 - val_loss: 0.2381 - val_accuracy: 0.9304
Epoch 68/70
469/469 [==============================] - 10s 22ms/step - loss: 0.3629
- accuracy: 0.8885 - val_loss: 0.2367 - val_accuracy: 0.9310
Epoch 69/70
469/469 [==============================] - 10s 21ms/step - loss: 0.3624
- accuracy: 0.8905 - val_loss: 0.2354 - val_accuracy: 0.9308
Epoch 70/70
469/469 [==============================] - 10s 22ms/step - loss: 0.3616
- accuracy: 0.8891 - val_loss: 0.2343 - val_accuracy: 0.9314
The model has been trained Successfully.
Saving the model as 'new_mnist(2x2)_epoch70.h5'
```

Hence we may come to a conclusion that we have trained various models with various `epoch` values wherein the 'new_mnist(4x4)_epoch70' model was the best model performing with an accuracy of 95.64% accuracy. Here the (4x4) represents the kernel size for the second Convolutional Layer as one of the options whereas the (2x2) happens to be the size of filter/kernel size as the second option.

1. 'new_mnist(4x4)_epoch20' --> 83.18%
2. 'new_mnist(4x4)_epoch50' --> 93.19%
3. 'new_mnist(4x4)_epoch70' --> 95.64% (Highest) --> Feature map size = (5x5x64)
4. 'new_mnist(2x2)_epoch20' --> 77.87%
5. 'new_mnist(2x2)_epoch50' --> 89.81%
6. 'new_mnist(2x2)_epoch70' --> 93.14% --> Feature map size = (6x6x64)