PERFORMANCE STUDY:

Let us try to change some nature of our network and observe the performance.

1. Number of hidden layers
2. Number of epochs
3. Number of convolutions.

CHANGING NUMBER OF HIDDEN LAYERS:

Adding extra 3 hidden layers:

We just add 3 extra hidden layers with same configuration,

```
In [52]: cnn = Sequential()

         #Convolution
         cnn.add(Conv2D(32, (3, 3), activation="relu", input_shape=(64, 64, 3)))

         #Pooling
         cnn.add(MaxPooling2D(pool_size = (2, 2)))

         # 2nd Convolution
         cnn.add(Conv2D(32, (3, 3), activation="relu"))

         # 2nd Pooling layer
         cnn.add(MaxPooling2D(pool_size = (2, 2)))

         # Flatten the layer
         cnn.add(Flatten())

         # Fully Connected Layers
         cnn.add(Dense(activation = 'relu', units = 128))
         cnn.add(Dense(activation = 'relu', units = 128))
         cnn.add(Dense(activation = 'relu', units = 128))
         cnn.add(Dense(activation = 'relu', units = 128))
         cnn.add(Dense(activation = 'sigmoid', units = 1))

         # Compile the Neural network
         cnn.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])
```

and the summary would be,

```
cnn.summary()
```

```
Model: "sequential_7"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_25 (Conv2D)           (None, 62, 62, 32)        896
_________________________________________________________________
max_pooling2d_23 (MaxPooling (None, 31, 31, 32)        0
_________________________________________________________________
conv2d_26 (Conv2D)           (None, 29, 29, 32)        9248
_________________________________________________________________
max_pooling2d_24 (MaxPooling (None, 14, 14, 32)        0
_________________________________________________________________
flatten_5 (Flatten)          (None, 6272)              0
_________________________________________________________________
dense_10 (Dense)             (None, 128)               802944
_________________________________________________________________
dense_11 (Dense)             (None, 128)               16512
_________________________________________________________________
dense_12 (Dense)             (None, 128)               16512
_________________________________________________________________
dense_13 (Dense)             (None, 128)               16512
_________________________________________________________________
dense_14 (Dense)             (None, 1)                 129
=================================================================
Total params: 862,753
Trainable params: 862,753
Non-trainable params: 0
_________________________________________________________________
```

We now run our new model for same test and train data for same no of epochs and same activation function to observe training accuracy,

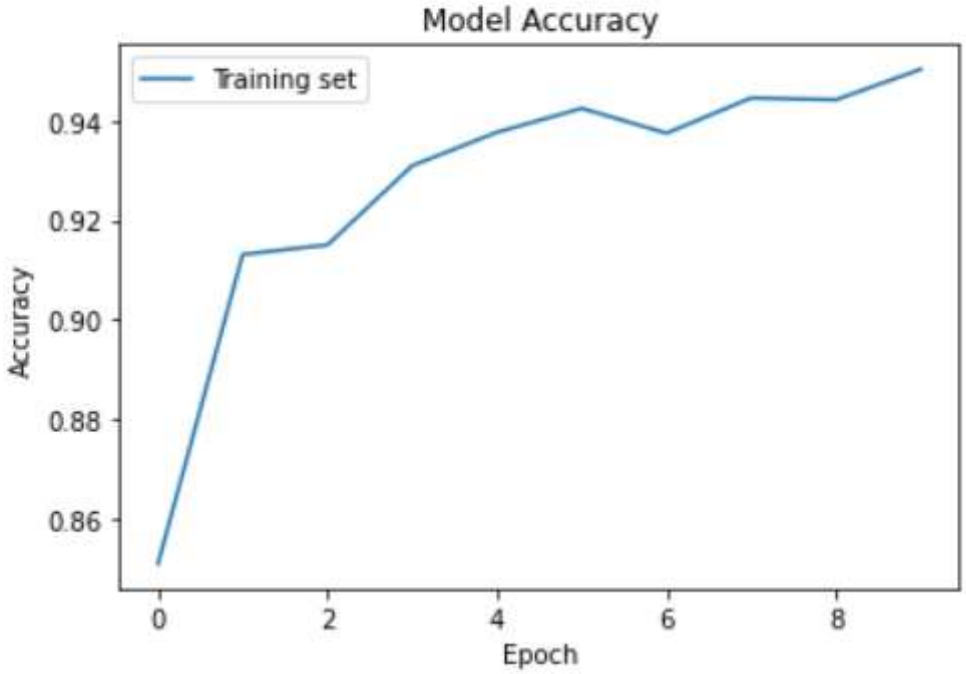
```
In [56]: cnn_model = cnn.fit_generator(training_set,
                                       steps_per_epoch = 163,
                                       epochs = 10,
                                       validation_data = validation_generator,
                                       validation_steps = 624)
```

```
Epoch 1/10
163/163 [==============================] - ETA: 0s - loss: 0.3512 - accuracy: 0.8510WARNING:tensorflow:Your
ta; interrupting training. Make sure that your dataset or generator can generate at least `steps_per_epoch
n this case, 624 batches). You may need to use the repeat() function when building your dataset.
163/163 [==============================] - 49s 296ms/step - loss: 0.3512 - accuracy: 0.8510 - val_loss: 0.4
0.8125
Epoch 2/10
163/163 [==============================] - 55s 335ms/step - loss: 0.2149 - accuracy: 0.9132
Epoch 3/10
163/163 [==============================] - 54s 329ms/step - loss: 0.2073 - accuracy: 0.9151
Epoch 4/10
163/163 [==============================] - 53s 325ms/step - loss: 0.1786 - accuracy: 0.9310
Epoch 5/10
163/163 [==============================] - 69s 424ms/step - loss: 0.1645 - accuracy: 0.9377
Epoch 6/10
163/163 [==============================] - 76s 469ms/step - loss: 0.1544 - accuracy: 0.9425
Epoch 7/10
163/163 [==============================] - 72s 443ms/step - loss: 0.1582 - accuracy: 0.9375
Epoch 8/10
163/163 [==============================] - 72s 442ms/step - loss: 0.1478 - accuracy: 0.9446
Epoch 9/10
163/163 [==============================] - 72s 443ms/step - loss: 0.1422 - accuracy: 0.9442
Epoch 10/10
163/163 [==============================] - 71s 438ms/step - loss: 0.1324 - accuracy: 0.9503
```

The training accuracy goes to 95.03%

Lets observe how the accuracy changes as 3 new hidden layers are added,

In [61]:
```
plt.plot(cnn_model.history['accuracy'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Training set'], loc='upper left')
plt.show()
```



It varies same as before. Now lets test our data to see whether there is a performance change,

```
test_accu = cnn.evaluate_generator(test_set,steps=624)
```

```
WARNING:tensorflow:Your input ran out of data; interrupting
least `steps_per_epoch * epochs` batches (in this case, 624
our dataset.
```

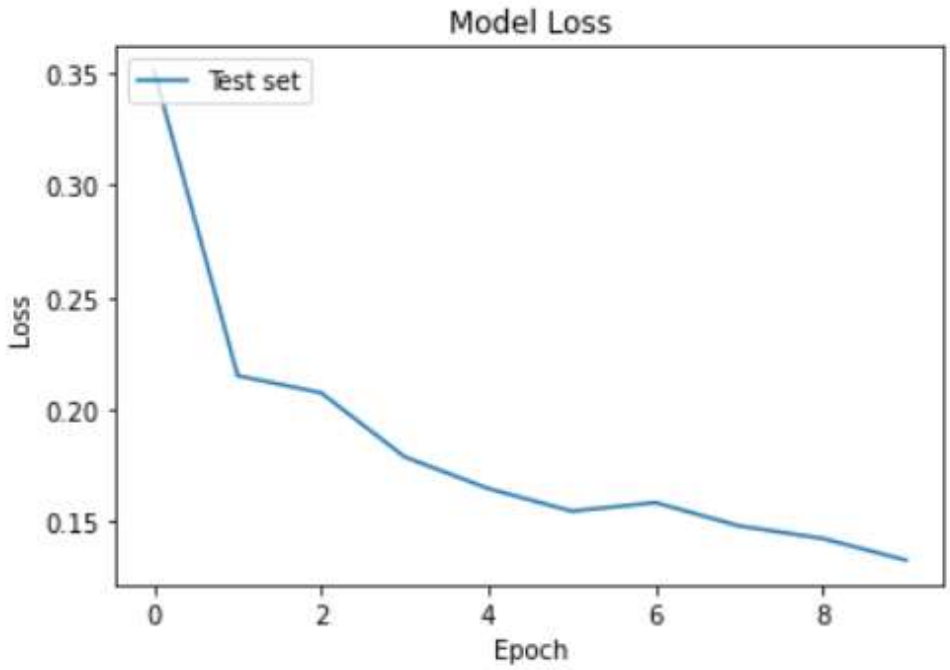
```
print('The testing accuracy is :',test_accu[1]*100, '%')
```

```
The testing accuracy is : 87.9807710647583 %
```

Our test accuracy is 87.98%

In [62]:

```
plt.plot(cnn_model.history['loss'])
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend([ 'Test set'], loc='upper left')
plt.show()
```



	NO HIDDEN LAYER	3 HIDDEN LAYERS	
TRAINING ACCURACY	95.15	95.03	
TESTING ACCURACY	87.5	87.98	

THUS THE INCREASE IN THE DEPTH OF NETWORK DOES NOT AFFECT THE PERFORMANCE TO LARGER EXTENT. ONLY LITTLE VARIATION IS OBSERVED FOR THIS DATA.

NUMBER OF EPOCHS:

We now increase the no of epochs to 20 to observe whether there is a change in the performance.

```
cnn = Sequential()

#Convolution
cnn.add(Conv2D(32, (3, 3), activation="relu", input_shape=(64, 64, 3)))

#Pooling
cnn.add(MaxPooling2D(pool_size = (2, 2)))

# 2nd Convolution
cnn.add(Conv2D(32, (3, 3), activation="relu"))

# 2nd Pooling layer
cnn.add(MaxPooling2D(pool_size = (2, 2)))

# Flatten the layer
cnn.add(Flatten())

# Fully Connected Layers
cnn.add(Dense(activation = 'relu', units = 128))
cnn.add(Dense(activation = 'sigmoid', units = 1))

# Compile the Neural network
cnn.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])
```

We work with the same network as before,

```
cnn.summary()
```

```
Model: "sequential_6"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_23 (Conv2D)           (None, 62, 62, 32)        896
_________________________________________________________________
max_pooling2d_21 (MaxPooling (None, 31, 31, 32)        0
_________________________________________________________________
conv2d_24 (Conv2D)           (None, 29, 29, 32)        9248
_________________________________________________________________
max_pooling2d_22 (MaxPooling (None, 14, 14, 32)        0
_________________________________________________________________
flatten_4 (Flatten)          (None, 6272)              0
_________________________________________________________________
dense_8 (Dense)              (None, 128)               802944
_________________________________________________________________
dense_9 (Dense)              (None, 1)                 129
=================================================================
Total params: 813,217
Trainable params: 813,217
Non-trainable params: 0
_________________________________________________________________
```

Same model as the no hidden layer network. Now we change the no of epochs to observe the result.

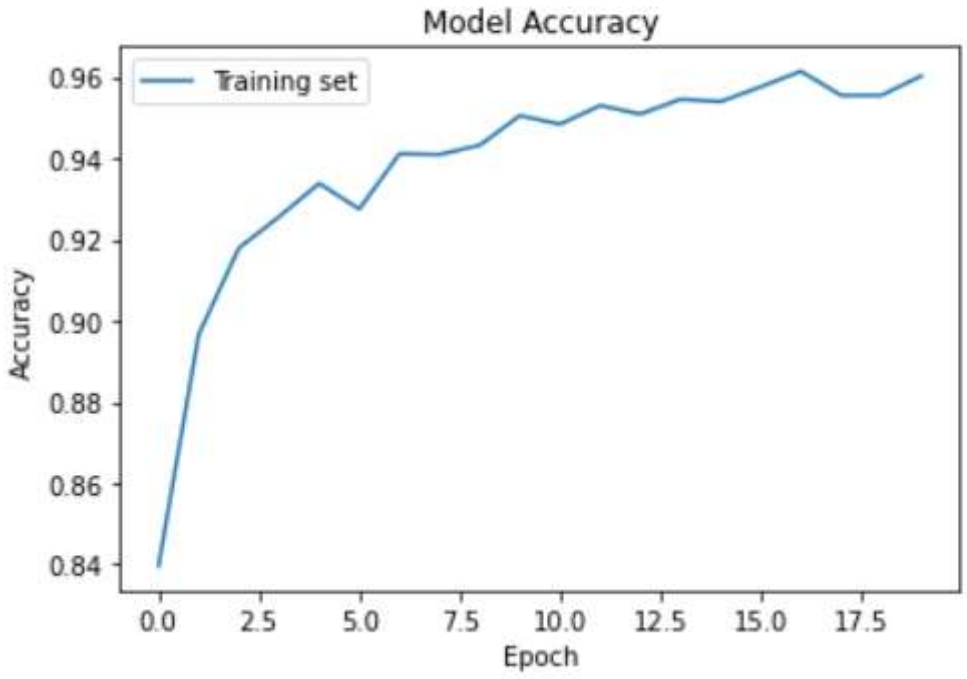
```
cnn_model = cnn.fit_generator(training_set,
                         steps_per_epoch = 163,
                         epochs = 20,
                         validation_data = validation_generator,
                         validation_steps = 624)
```

```
Epoch 1/20
163/163 [==============================] - ETA: 0s - loss: 0.3730 - accuracy: 0.8397WARNING:tensorflow:Your input ran out of da
ta; interrupting training. Make sure that your dataset or generator can generate at least `steps_per_epoch * epochs` batches (i
n this case, 624 batches). You may need to use the repeat() function when building your dataset.
163/163 [==============================] - 55s 338ms/step - loss: 0.3730 - accuracy: 0.8397 - val_loss: 0.2776 - val_accuracy:
0.8750
Epoch 2/20
163/163 [==============================] - 72s 442ms/step - loss: 0.2417 - accuracy: 0.8967
Epoch 3/20
163/163 [==============================] - 72s 442ms/step - loss: 0.2066 - accuracy: 0.9179
Epoch 4/20
163/163 [==============================] - 72s 440ms/step - loss: 0.1848 - accuracy: 0.9256
Epoch 5/20
163/163 [==============================] - 71s 436ms/step - loss: 0.1645 - accuracy: 0.9339
Epoch 6/20
163/163 [==============================] - 818s 5s/step - loss: 0.1788 - accuracy: 0.9275
Epoch 7/20
163/163 [==============================] - 50s 305ms/step - loss: 0.1496 - accuracy: 0.9411
Epoch 8/20
163/163 [==============================] - 65s 401ms/step - loss: 0.1528 - accuracy: 0.9410
Epoch 9/20
163/163 [==============================] - 71s 437ms/step - loss: 0.1439 - accuracy: 0.9433
Epoch 10/20
163/163 [==============================] - 72s 442ms/step - loss: 0.1294 - accuracy: 0.9505
Epoch 11/20
163/163 [==============================] - 71s 436ms/step - loss: 0.1350 - accuracy: 0.9484
Epoch 12/20
Epoch 12/20
163/163 [==============================] - 71s 437ms/step - loss: 0.1199 - accuracy: 0.9530
Epoch 13/20
163/163 [==============================] - 71s 437ms/step - loss: 0.1293 - accuracy: 0.9509
Epoch 14/20
163/163 [==============================] - 73s 446ms/step - loss: 0.1169 - accuracy: 0.9546
Epoch 15/20
163/163 [==============================] - 72s 438ms/step - loss: 0.1224 - accuracy: 0.9540
Epoch 16/20
163/163 [==============================] - 72s 441ms/step - loss: 0.1079 - accuracy: 0.9576
Epoch 17/20
163/163 [==============================] - 71s 438ms/step - loss: 0.1037 - accuracy: 0.9615
Epoch 18/20
163/163 [==============================] - 72s 440ms/step - loss: 0.1148 - accuracy: 0.9555
Epoch 19/20
163/163 [==============================] - 71s 435ms/step - loss: 0.1182 - accuracy: 0.9555
Epoch 20/20
163/163 [==============================] - 72s 440ms/step - loss: 0.1018 - accuracy: 0.9603
```

Our accuracy got increased to 96.03%

Lets observe how the accuracy is varying with increase in the no of epochs.

```
plt.plot(cnn_model.history['accuracy'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Training set'], loc='upper left')
plt.show()
```



Varies as same as before. Now lets evaluate our testing set.

```
test_accu = cnn.evaluate_generator(test_set,steps=624)
```

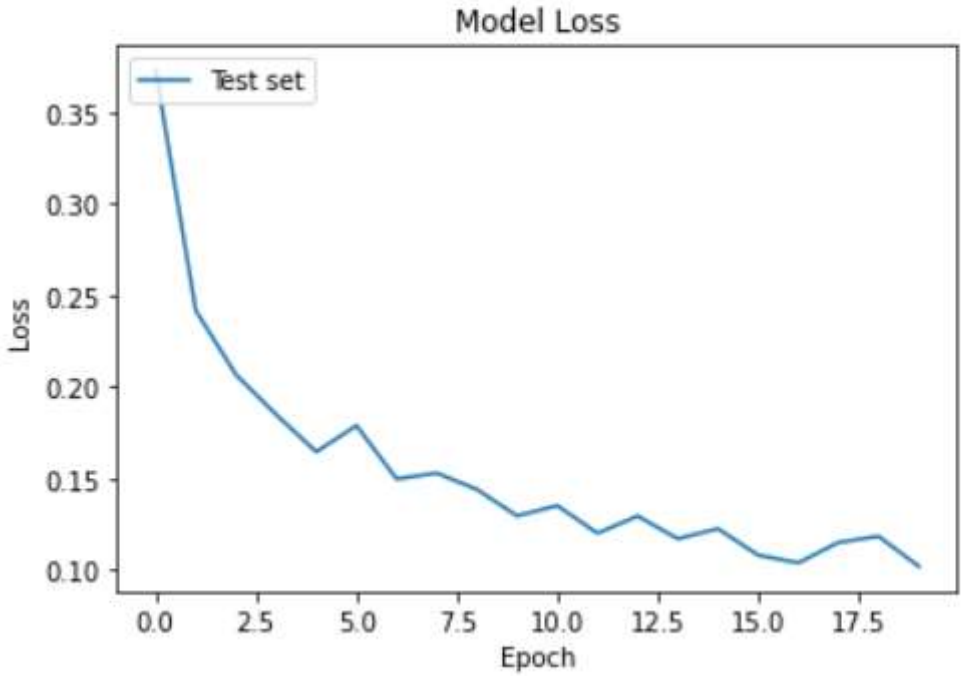
```
WARNING:tensorflow:Your input ran out of data; interrupting
least `steps_per_epoch * epochs` batches (in this case, 624
our dataset.
```

```
print('The testing accuracy is :',test_accu[1]*100, '%')
```

```
The testing accuracy is : 89.26281929016113 %
```

Our accuracy is 89.26%

```
plt.plot(cnn_model.history['loss'])
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend([ 'Test set'], loc='upper left')
plt.show()
```



	100 EPOCHS	**300 EPOCHS**
TRAINING ACCURACY	95.15	96.03
TESTING ACCURACY	87.5	89.29

WITH INCREASE IN NUMBER OF EPOCHS THE TRAINING ACCURACY AND THE TESTING INCREASES SLIGHTLY FOR THIS DATA.

NUMBER OF CONVOLUTIONS:

Previously we had 2 convolution layers each of which is followed by a pooling layer, but now we add 2 more convolution and pooling layers.

```
cnn = Sequential()

#Convolution
cnn.add(Conv2D(32, (3, 3), activation="relu", input_shape=(64, 64, 3)))

#Pooling
cnn.add(MaxPooling2D(pool_size = (2, 2)))

# 2nd Convolution
cnn.add(Conv2D(32, (3, 3), activation="relu"))

# 2nd Pooling layer
cnn.add(MaxPooling2D(pool_size = (2, 2)))

# 3rd Convolution
cnn.add(Conv2D(32, (3, 3), activation="relu"))

# 3rd Pooling layer
cnn.add(MaxPooling2D(pool_size = (2, 2)))

# 4th Convolution
cnn.add(Conv2D(32, (3, 3), activation="relu"))

# 4th Pooling layer
cnn.add(MaxPooling2D(pool_size = (2, 2)))

# Flatten the layer
cnn.add(Flatten())

# Fully Connected Layers
cnn.add(Dense(activation = 'relu', units = 128))
cnn.add(Dense(activation = 'sigmoid', units = 1))
```

The fully connected layer remains the same as it is a binary classification.

In [33]:
```
cnn.summary()
```

```
Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_19 (Conv2D)           (None, 62, 62, 32)        896
_________________________________________________________________
max_pooling2d_17 (MaxPooling (None, 31, 31, 32)        0
_________________________________________________________________
conv2d_20 (Conv2D)           (None, 29, 29, 32)        9248
_________________________________________________________________
max_pooling2d_18 (MaxPooling (None, 14, 14, 32)        0
_________________________________________________________________
conv2d_21 (Conv2D)           (None, 12, 12, 32)        9248
_________________________________________________________________
max_pooling2d_19 (MaxPooling (None, 6, 6, 32)          0
_________________________________________________________________
conv2d_22 (Conv2D)           (None, 4, 4, 32)          9248
_________________________________________________________________
max_pooling2d_20 (MaxPooling (None, 2, 2, 32)          0
_________________________________________________________________
flatten_3 (Flatten)          (None, 128)               0
_________________________________________________________________
dense_6 (Dense)              (None, 128)               16512
_________________________________________________________________
dense_7 (Dense)              (None, 1)                 129
=================================================================
Total params: 45,281
Trainable params: 45,281
Non-trainable params: 0
_________________________________________________________________
```

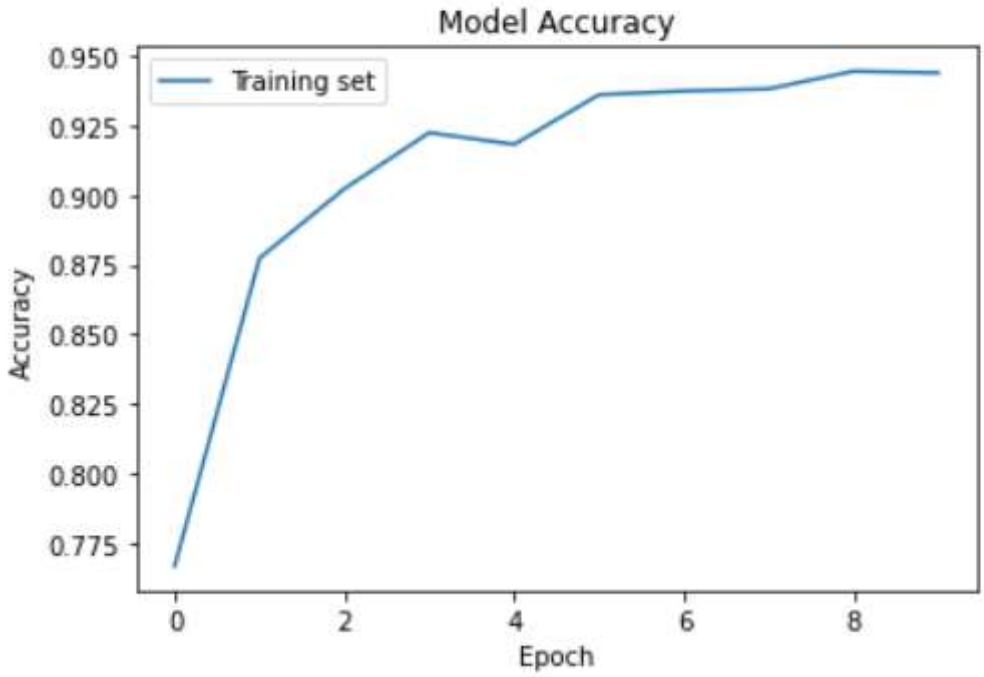
Now we train our data to observe the training accuracy,

```
cnn_model = cnn.fit_generator(training_set,
                         steps_per_epoch = 163,
                         epochs = 10,
                         validation_data = validation_generator,
                         validation_steps = 624)
```

```
Epoch 1/10
163/163 [==============================] - ETA: 0s - loss: 0.5174 - accuracy: 0.7669WARNING:tensorflow:Your input ran out of d
ta; interrupting training. Make sure that your dataset or generator can generate at least `steps_per_epoch * epochs` batches (
n this case, 624 batches). You may need to use the repeat() function when building your dataset.
163/163 [==============================] - 84s 509ms/step - loss: 0.5174 - accuracy: 0.7669 - val_loss: 0.6709 - val_accuracy:
0.6875
Epoch 2/10
163/163 [==============================] - 79s 481ms/step - loss: 0.2794 - accuracy: 0.8773
Epoch 3/10
163/163 [==============================] - 75s 460ms/step - loss: 0.2378 - accuracy: 0.9022
Epoch 4/10
163/163 [==============================] - 72s 440ms/step - loss: 0.1926 - accuracy: 0.9224
Epoch 5/10
163/163 [==============================] - 73s 447ms/step - loss: 0.1993 - accuracy: 0.9181
Epoch 6/10
163/163 [==============================] - 72s 441ms/step - loss: 0.1653 - accuracy: 0.9360
Epoch 7/10
163/163 [==============================] - 66s 403ms/step - loss: 0.1674 - accuracy: 0.9373
Epoch 8/10
163/163 [==============================] - 60s 368ms/step - loss: 0.1667 - accuracy: 0.9381
Epoch 9/10
163/163 [==============================] - 55s 339ms/step - loss: 0.1464 - accuracy: 0.9444
Epoch 10/10
163/163 [==============================] - 53s 325ms/step - loss: 0.1458 - accuracy: 0.9438
```

The training accuracy is 94.38%. Lets observe how the accuracy increases with epochs,

```
plt.plot(cnn_model.history['accuracy'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Training set'], loc='upper left')
plt.show()
```



It varies the same as before. Now lets test the model with our testing set,

```
test_accu = cnn.evaluate_generator(test_set,steps=624)
```

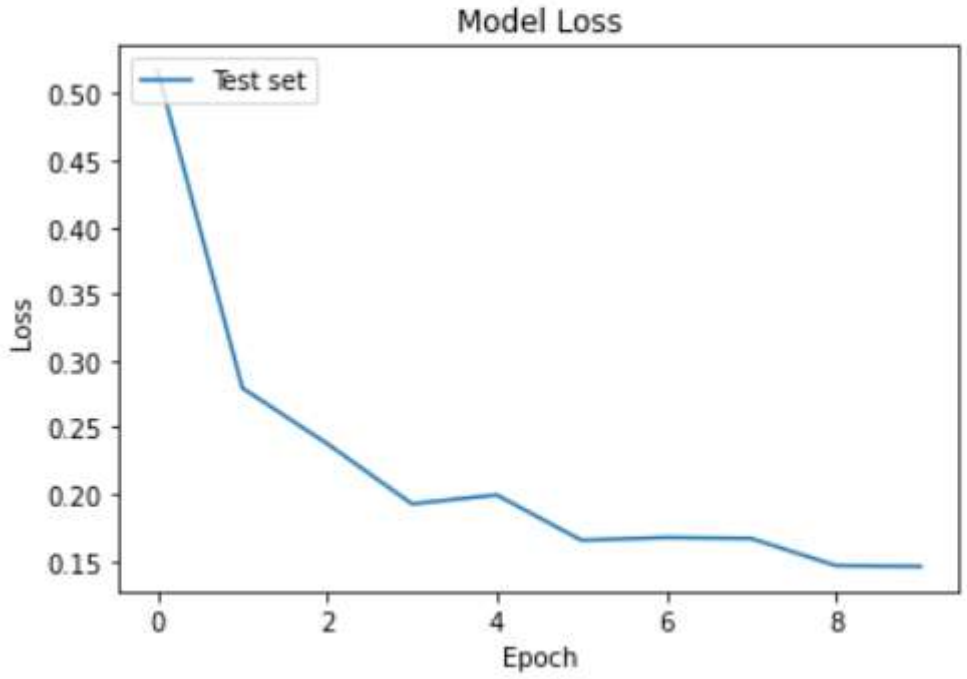
```
WARNING:tensorflow:Your input ran out of data; interrupting tra
least `steps_per_epoch * epochs` batches (in this case, 624 bat
our dataset.
```

```
print('The testing accuracy is :',test_accu[1]*100, '%')
```

```
The testing accuracy is : 83.65384340286255 %
```

The accuracy is 83.65%

```
plt.plot(cnn_model.history['loss'])
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend([ 'Test set'], loc='upper left')
plt.show()
```



	2 CONVOLUTION LAYERS	4 CONVOLUTION LAYERS
TRAINING ACCURACY	95.15	94.38
TESTING ACCURACY	87.5	83.65

WITH VARYING THE NUMBER OF CONVOLUTIONS IT PERFORMS SLIGHTLY WORSER THAN BEFORE.

THUS THE FACTORS AFFECTING PERFORMANCE ARE VIEWED EXPRIMENTALLY.