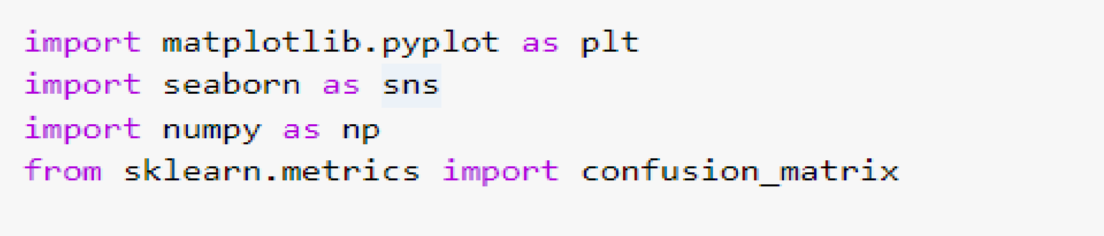
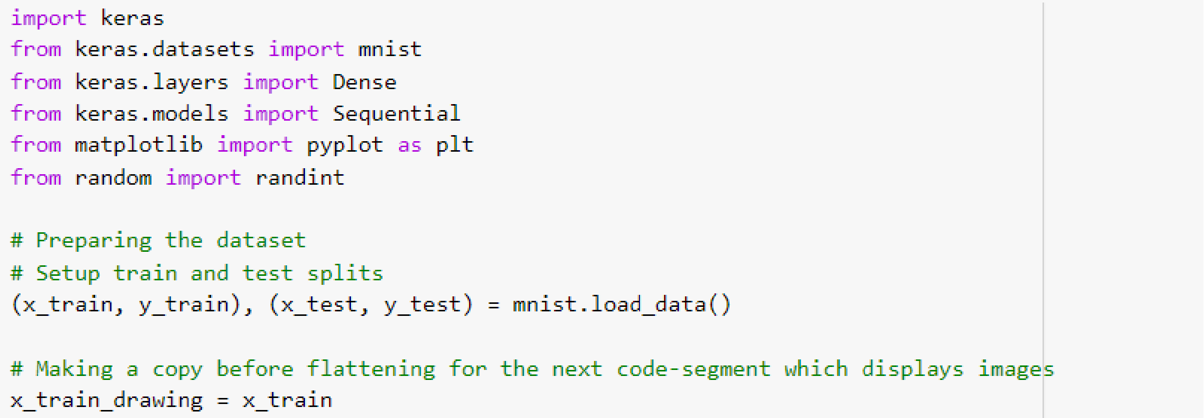
EXPERIMENT-6

AIM: **Write a program to evaluate the performance of different optimizers on MNIST dataset.**

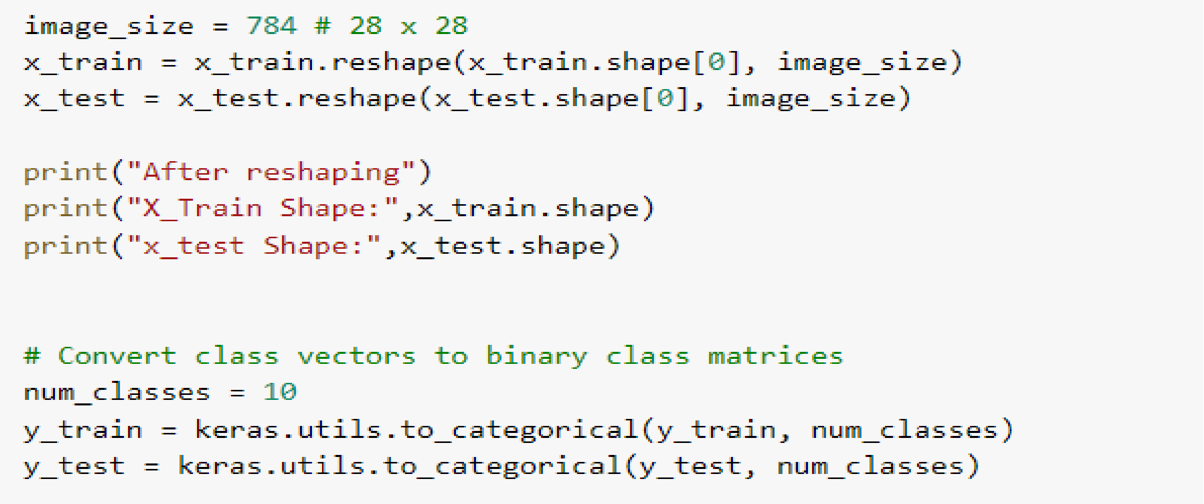
CODE and OUTPUT:



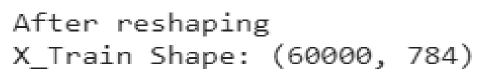
```
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from sklearn.metrics import confusion_matrix
```



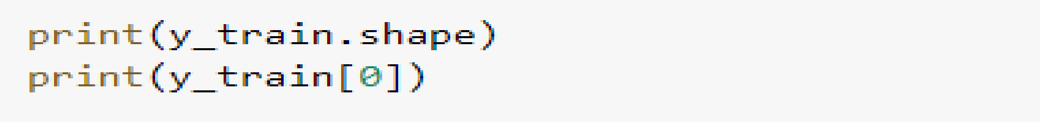
```
import keras
from keras.datasets import mnist
from keras.layers import Dense
from keras.models import Sequential
from matplotlib import pyplot as plt
from random import randint

# Preparing the dataset
# Setup train and test splits
(x_train, y_train), (x_test, y_test) = mnist.load_data()

# Making a copy before flattening for the next code-segment which displays images
x_train_drawing = x_train
```



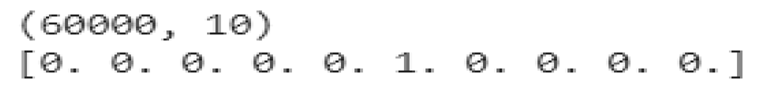
```
image_size = 784 # 28 x 28
x_train = x_train.reshape(x_train.shape[0], image_size)
x_test = x_test.reshape(x_test.shape[0], image_size)

print("After reshaping")
print("X_Train Shape:",x_train.shape)
print("x_test Shape:",x_test.shape)


# Convert class vectors to binary class matrices
num_classes = 10
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)
```



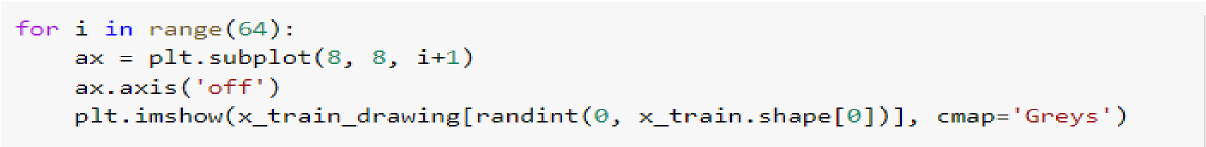
```
After reshaping
X_Train Shape: (60000, 784)
```



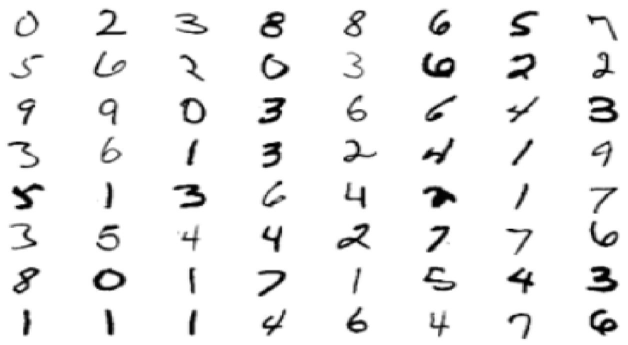
```
print(y_train.shape)
print(y_train[0])
```

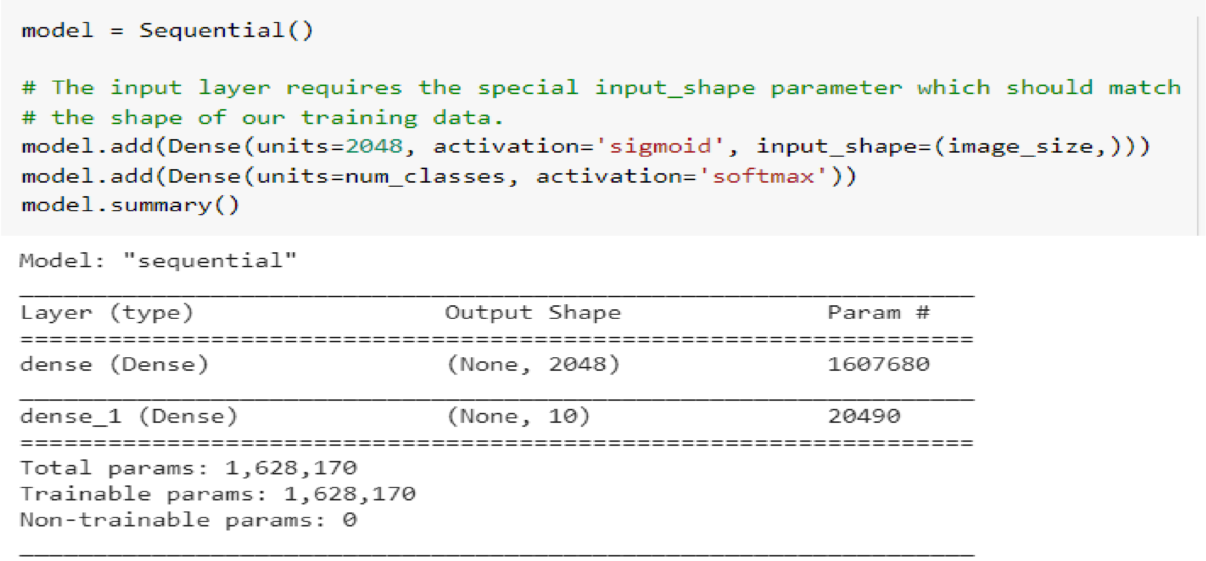


```
(60000, 10)
[0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
```



```
for i in range(64):
    ax = plt.subplot(8, 8, i+1)
    ax.axis('off')
    plt.imshow(x_train_drawing[randint(0, x_train.shape[0])], cmap='Greys')
```

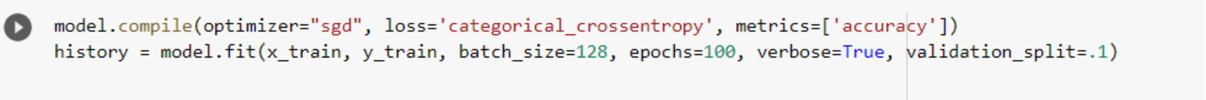




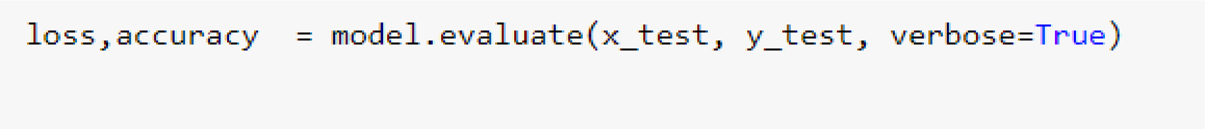
```
model = Sequential()

# The input layer requires the special input_shape parameter which should match
# the shape of our training data.
model.add(Dense(units=2048, activation='sigmoid', input_shape=(image_size,)))
model.add(Dense(units=num_classes, activation='softmax'))
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 2048)              1607680
_________________________________________________________________
dense_1 (Dense)              (None, 10)                20490
=================================================================
Total params: 1,628,170
Trainable params: 1,628,170
Non-trainable params: 0
_________________________________________________________________
```

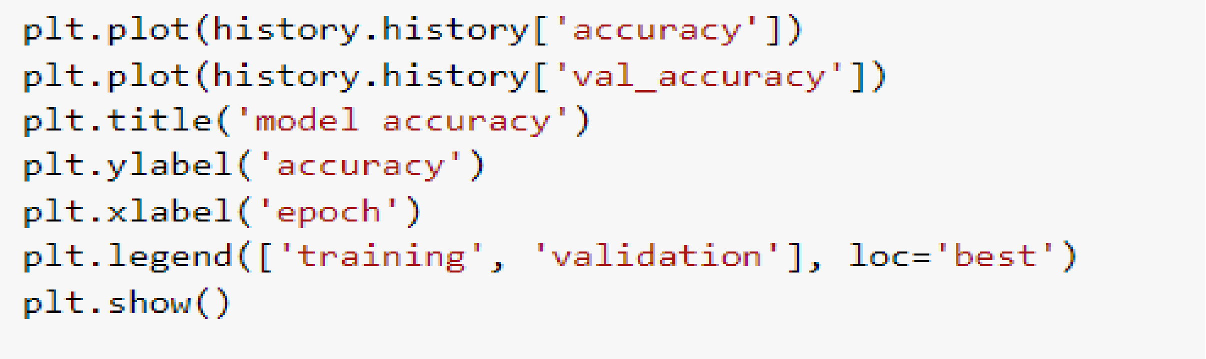
STOCHASTIC GRADIENT DESCENT



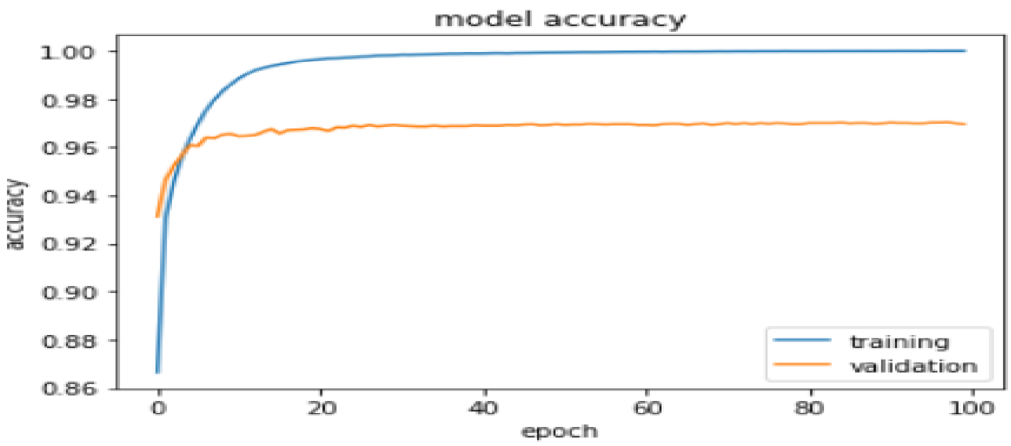
```
model.compile(optimizer="sgd", loss='categorical_crossentropy', metrics=['accuracy'])
history = model.fit(x_train, y_train, batch_size=128, epochs=100, verbose=True, validation_split=.1)
```

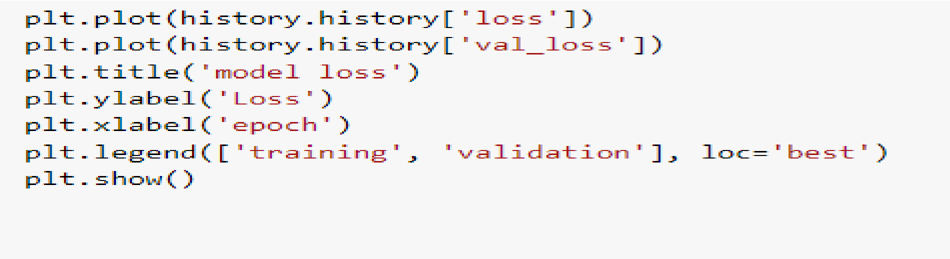


```
loss,accuracy  = model.evaluate(x_test, y_test, verbose=True)
```

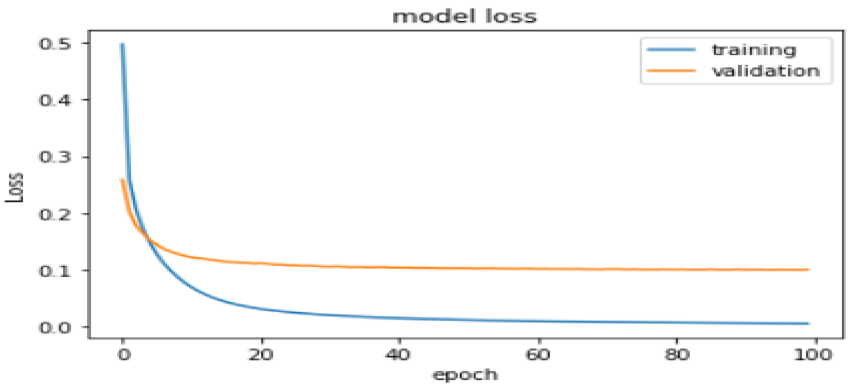


```
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['training', 'validation'], loc='best')
plt.show()
```

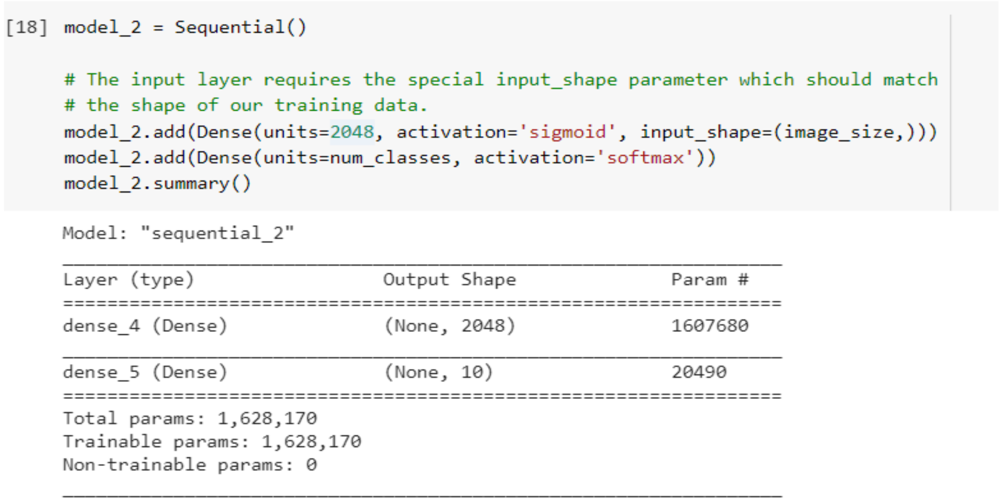




```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('Loss')
plt.xlabel('epoch')
plt.legend(['training', 'validation'], loc='best')
plt.show()
```

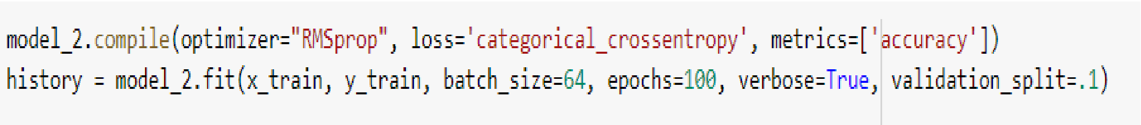


RMSprop

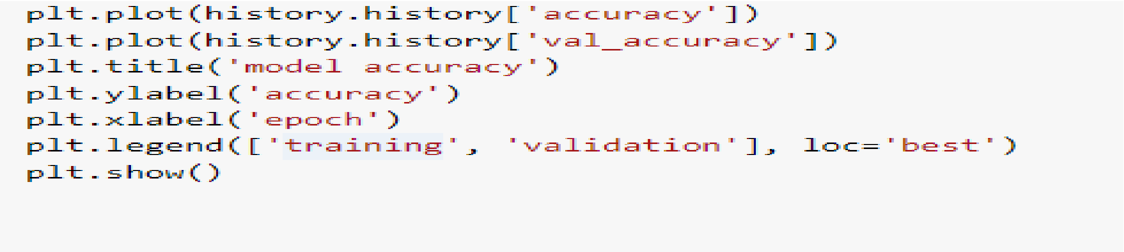


```
[18] model_2 = Sequential()

     # The input layer requires the special input_shape parameter which should match
     # the shape of our training data.
     model_2.add(Dense(units=2048, activation='sigmoid', input_shape=(image_size,)))
     model_2.add(Dense(units=num_classes, activation='softmax'))
     model_2.summary()
```

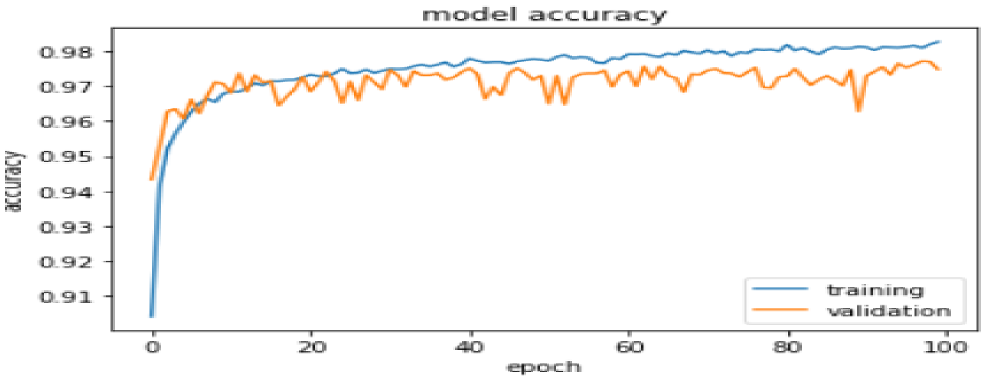
```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_4 (Dense)              (None, 2048)              1607680   
_________________________________________________________________
dense_5 (Dense)              (None, 10)                20490     
=================================================================
Total params: 1,628,170
Trainable params: 1,628,170
Non-trainable params: 0
_________________________________________________________________
```

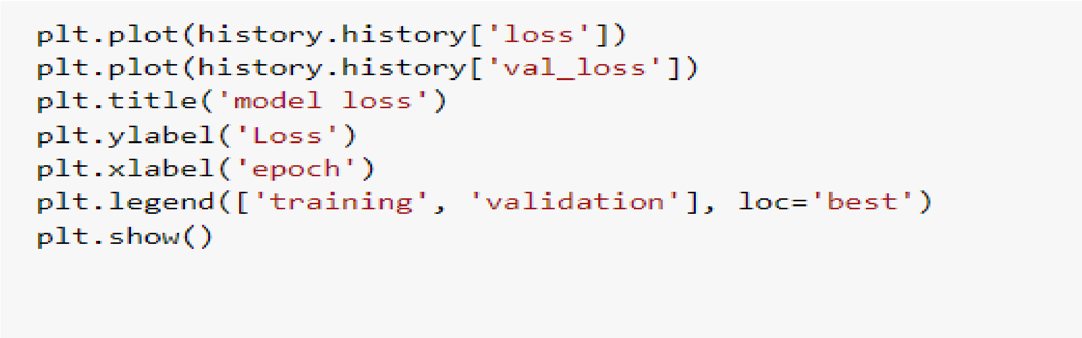


```
model_2.compile(optimizer="RMSprop", loss='categorical_crossentropy', metrics=['accuracy'])
history = model_2.fit(x_train, y_train, batch_size=64, epochs=100, verbose=True, validation_split=.1)
```

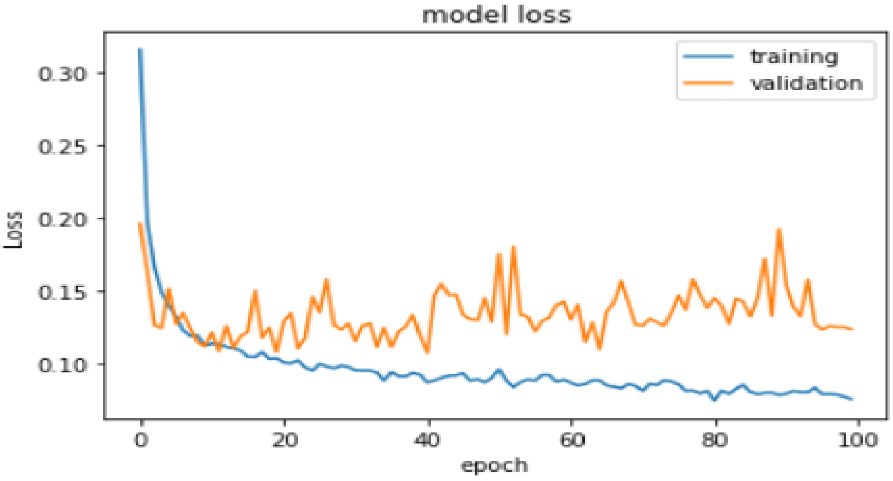


```
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['training', 'validation'], loc='best')
plt.show()
```

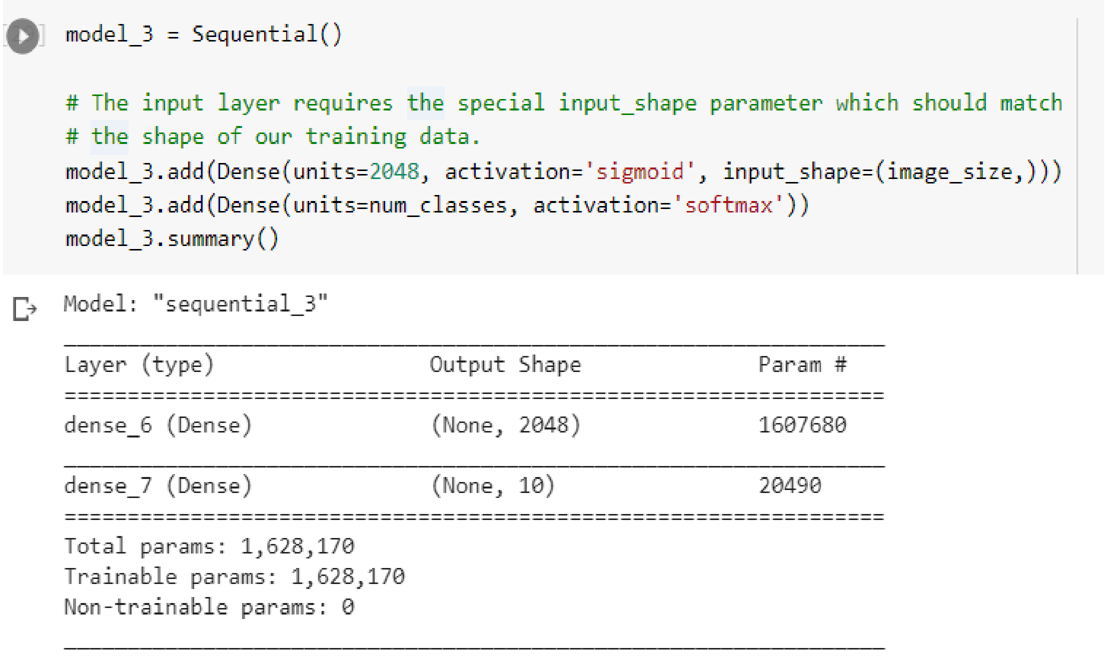




```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('Loss')
plt.xlabel('epoch')
plt.legend(['training', 'validation'], loc='best')
plt.show()
```

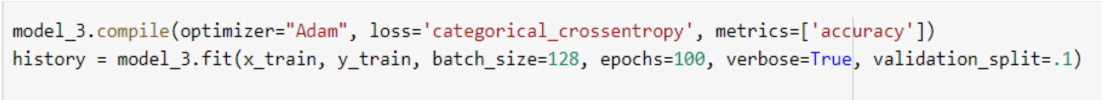


Adam

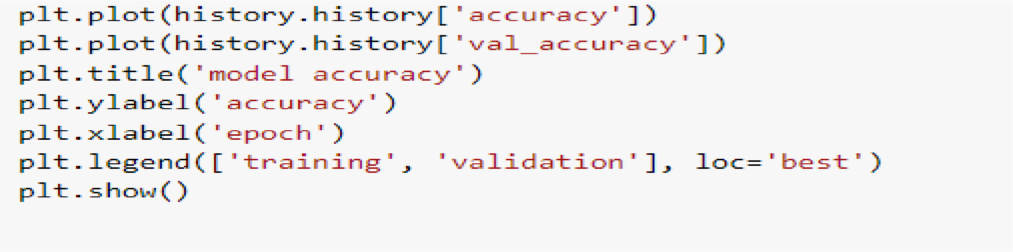


```
model_3 = Sequential()

# The input layer requires the special input_shape parameter which should match
# the shape of our training data.
model_3.add(Dense(units=2048, activation='sigmoid', input_shape=(image_size,)))
model_3.add(Dense(units=num_classes, activation='softmax'))
model_3.summary()
```

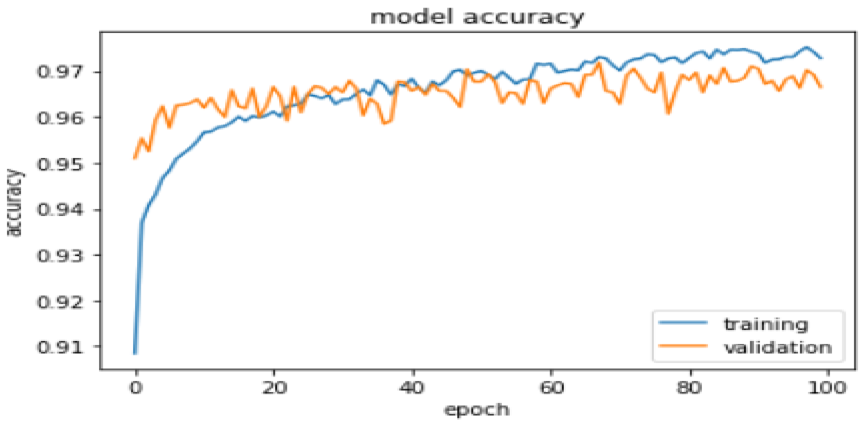
```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_6 (Dense)              (None, 2048)              1607680
_________________________________________________________________
dense_7 (Dense)              (None, 10)                20490
=================================================================
Total params: 1,628,170
Trainable params: 1,628,170
Non-trainable params: 0
_________________________________________________________________
```

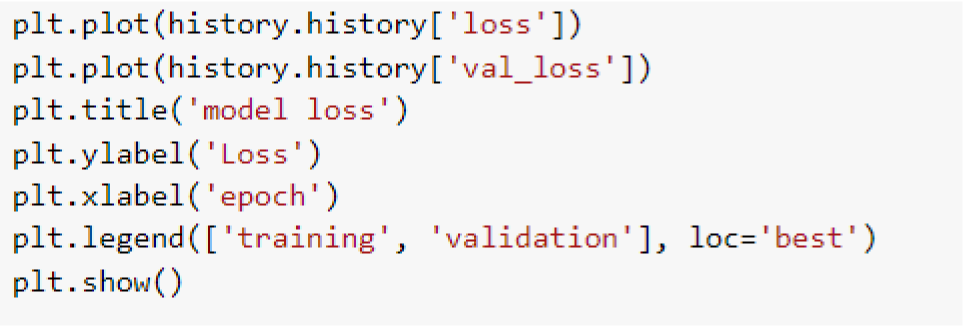


```
model_3.compile(optimizer="Adam", loss='categorical_crossentropy', metrics=['accuracy'])
history = model_3.fit(x_train, y_train, batch_size=128, epochs=100, verbose=True, validation_split=.1)
```

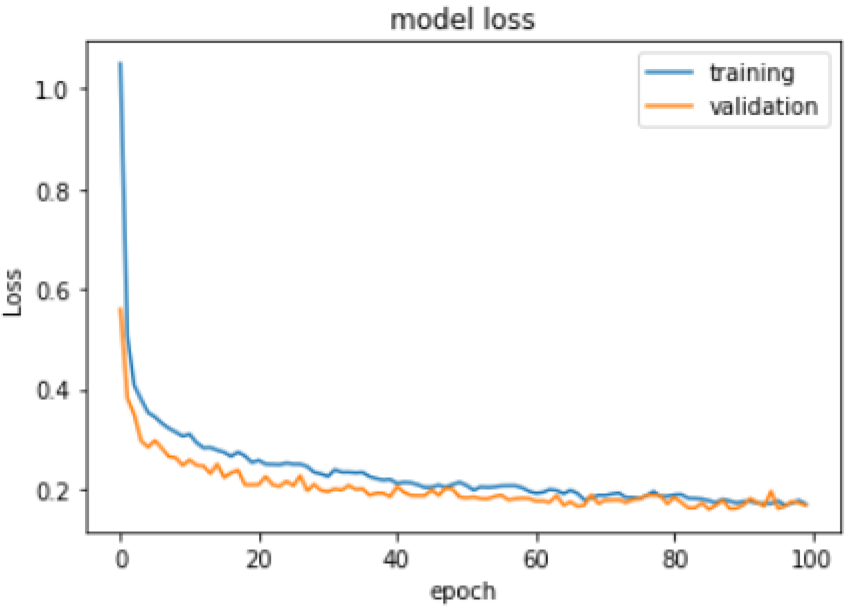


```
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['training', 'validation'], loc='best')
plt.show()
```

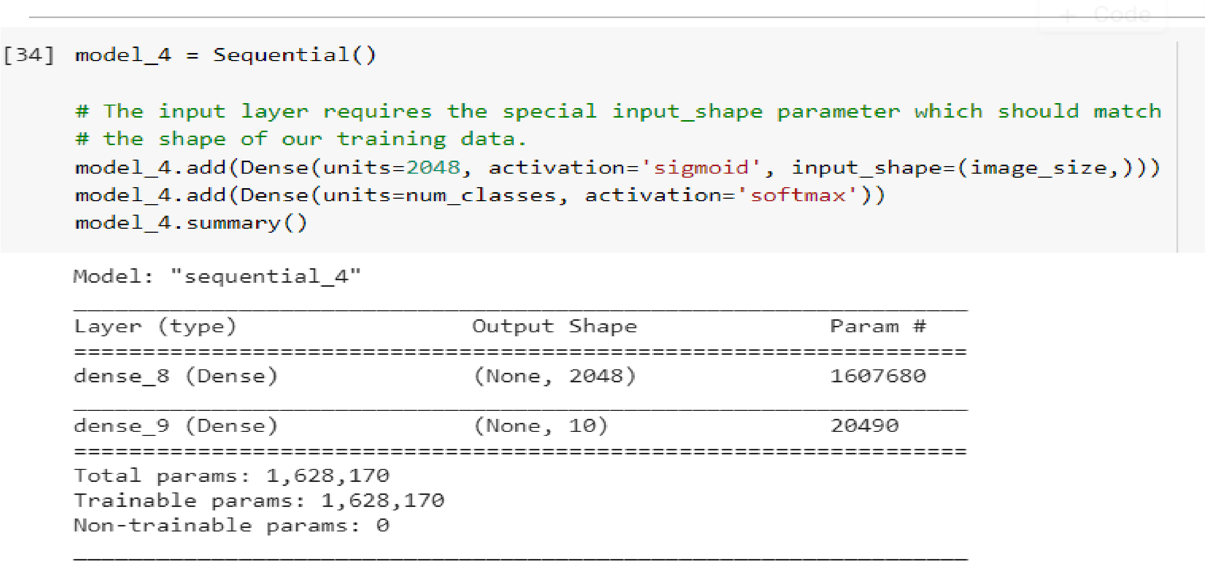




```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('Loss')
plt.xlabel('epoch')
plt.legend(['training', 'validation'], loc='best')
plt.show()
```

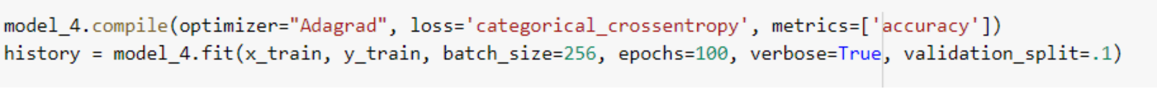


Adagrad

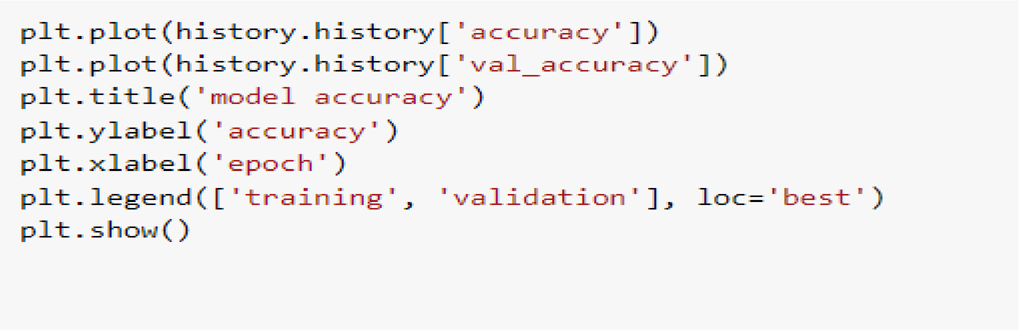


```
[34] model_4 = Sequential()

     # The input layer requires the special input_shape parameter which should match
     # the shape of our training data.
     model_4.add(Dense(units=2048, activation='sigmoid', input_shape=(image_size,)))
     model_4.add(Dense(units=num_classes, activation='softmax'))
     model_4.summary()
```

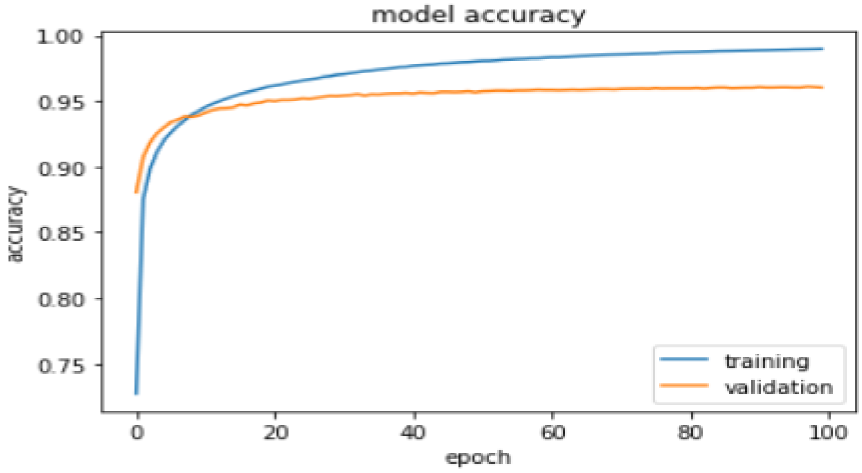
```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_8 (Dense)              (None, 2048)              1607680   
_________________________________________________________________
dense_9 (Dense)              (None, 10)                20490     
=================================================================
Total params: 1,628,170
Trainable params: 1,628,170
Non-trainable params: 0
_________________________________________________________________
```

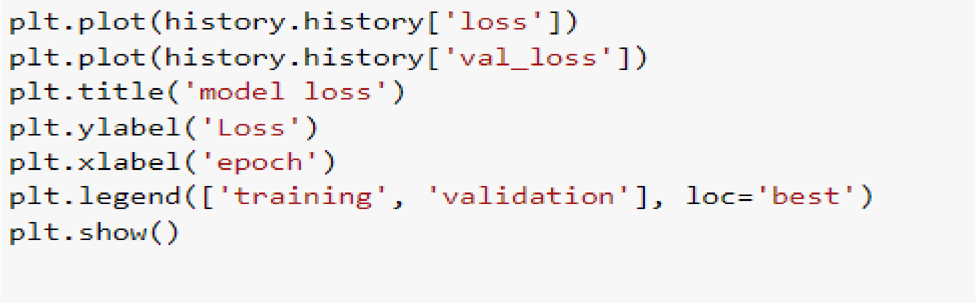


```
model_4.compile(optimizer="Adagrad", loss='categorical_crossentropy', metrics=['accuracy'])
history = model_4.fit(x_train, y_train, batch_size=256, epochs=100, verbose=True, validation_split=.1)
```

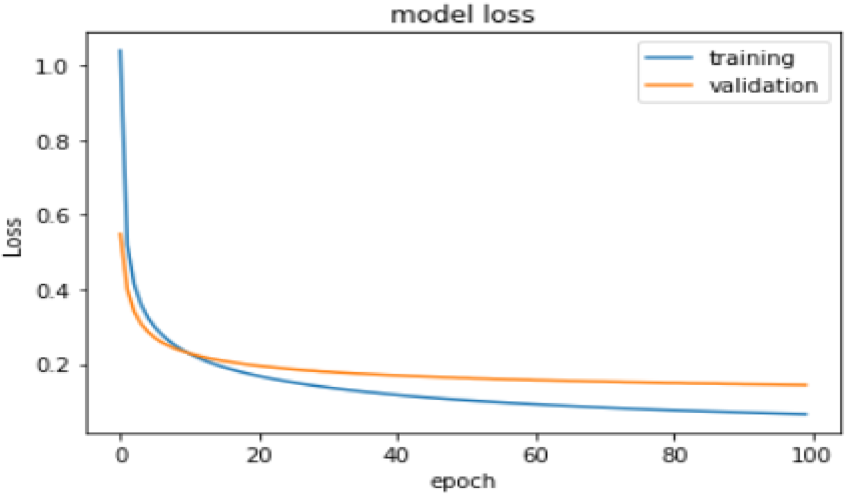


```
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['training', 'validation'], loc='best')
plt.show()
```





```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('Loss')
plt.xlabel('epoch')
plt.legend(['training', 'validation'], loc='best')
plt.show()
```


model loss
Loss
epoch
training
validation