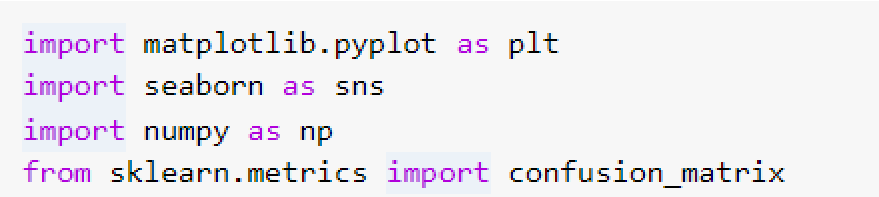
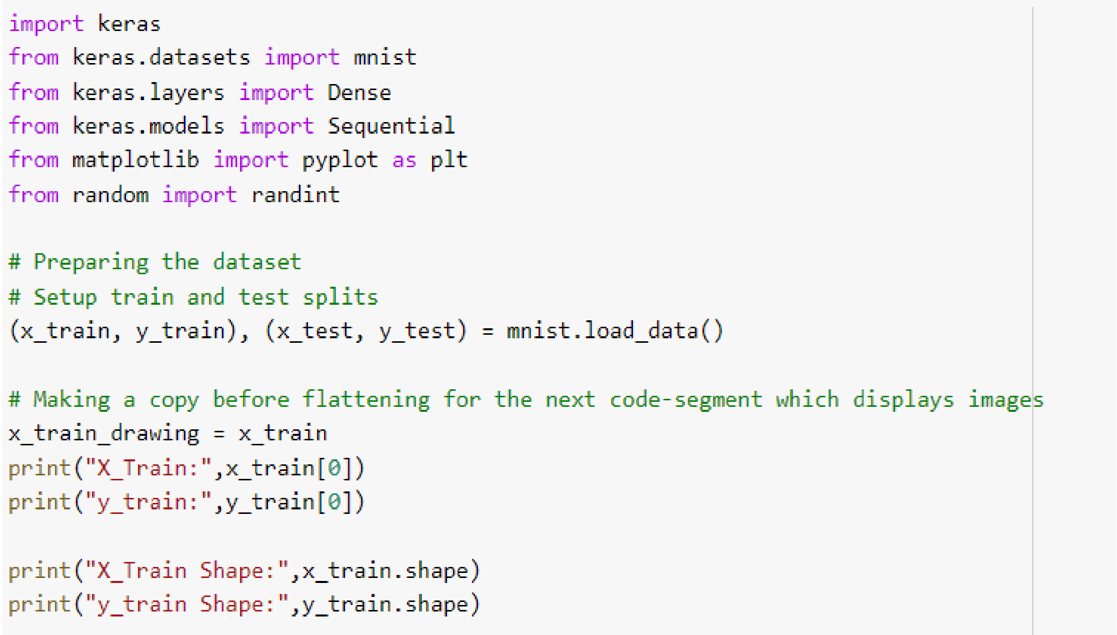
EXPERIMENT-4

AIM: Write a program to implement and verify the performance of shallow Neural Networks with different number of neurons.

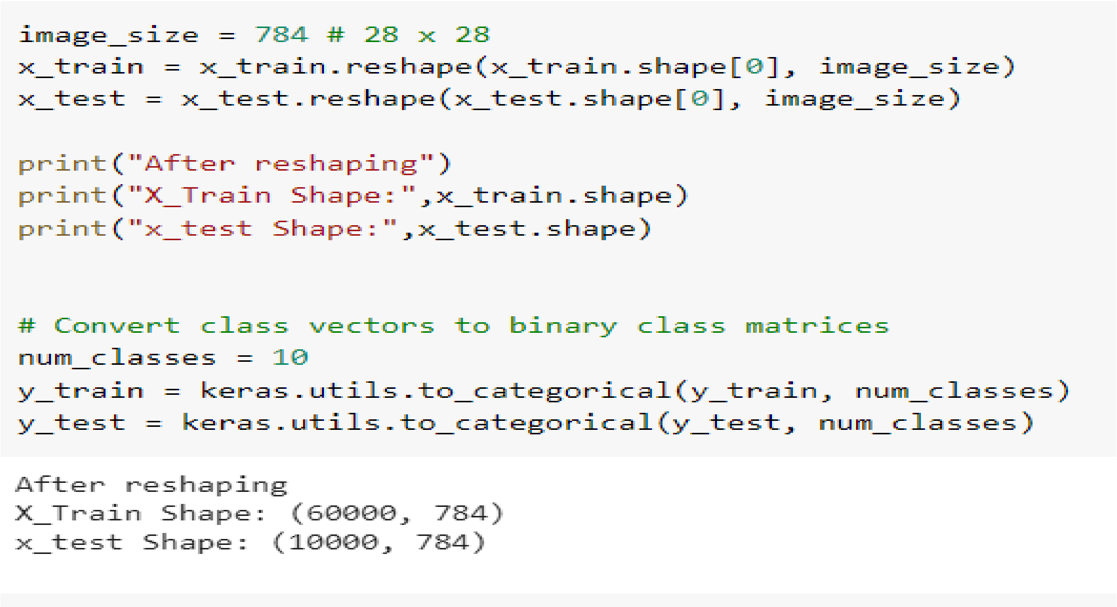
CODE and OUTPUT:



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

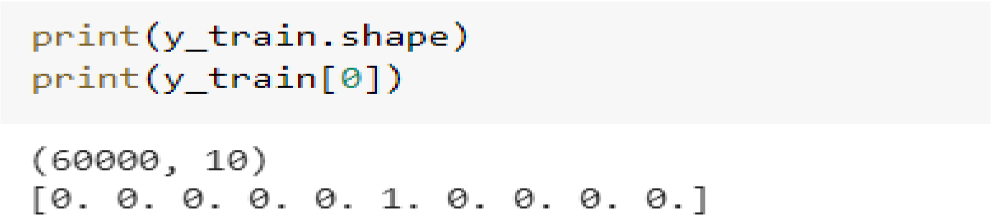


```
import keras
from keras.datasets import mnist
from keras.layers import Dense
from keras.models import Sequential
from matplotlib import pyplot as plt
from random import randint

# 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
print("X_Train:",x_train[0])
print("y_train:",y_train[0])

print("X_Train Shape:",x_train.shape)
print("y_train Shape:",y_train.shape)
```



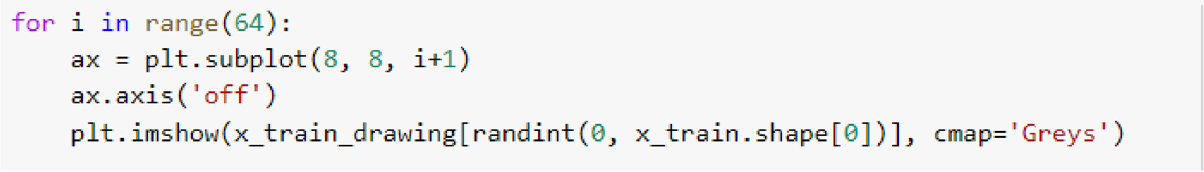
```
image_size = 784 # 28 x 28
x_train = x_train.reshape(x_train.shape[0], image_size)
x_test = x_test.reshape(x_test.shape[0], image_size)

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)
x_test Shape: (10000, 784)
```

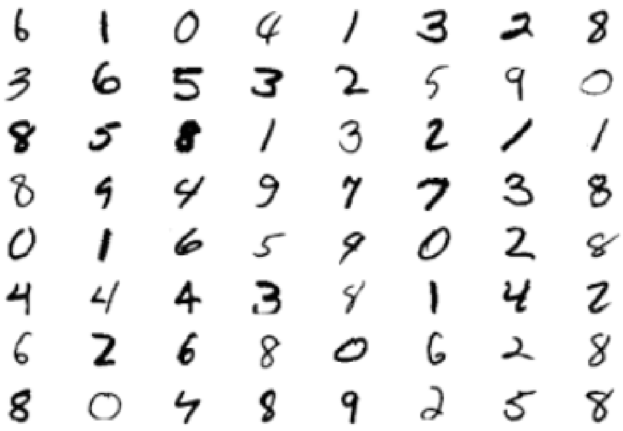


```
print(y_train.shape)
print(y_train[0])
```

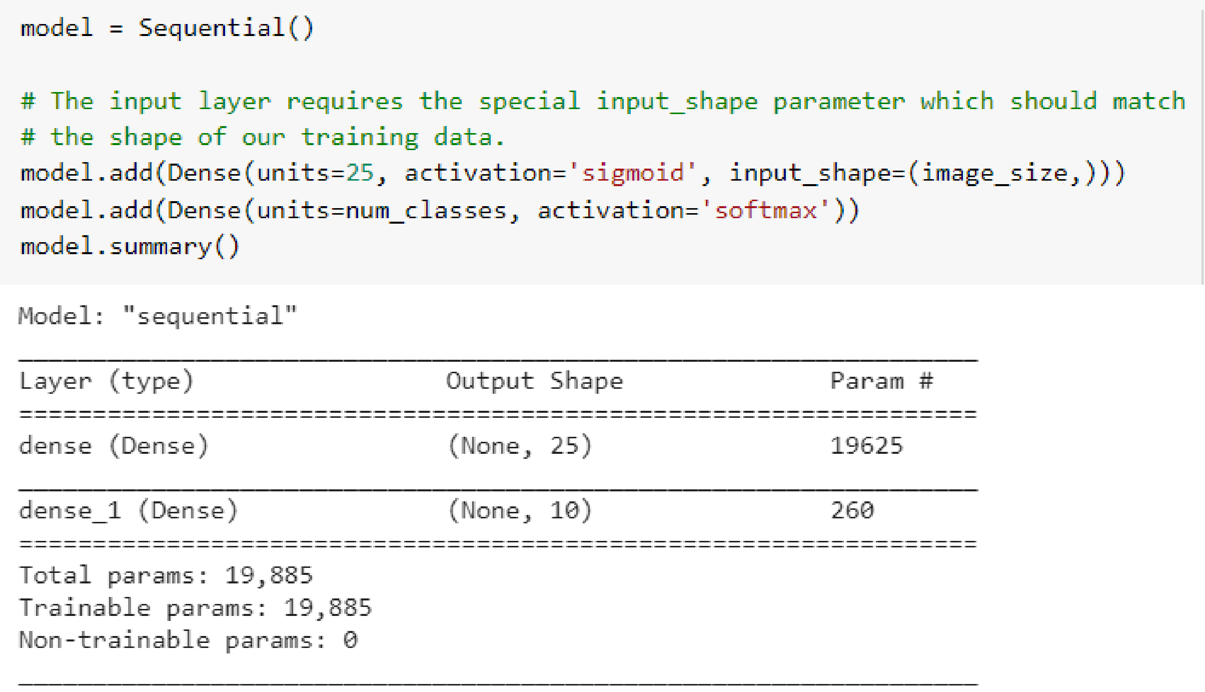
```
(60000, 10)
[0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
```



```
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')
```

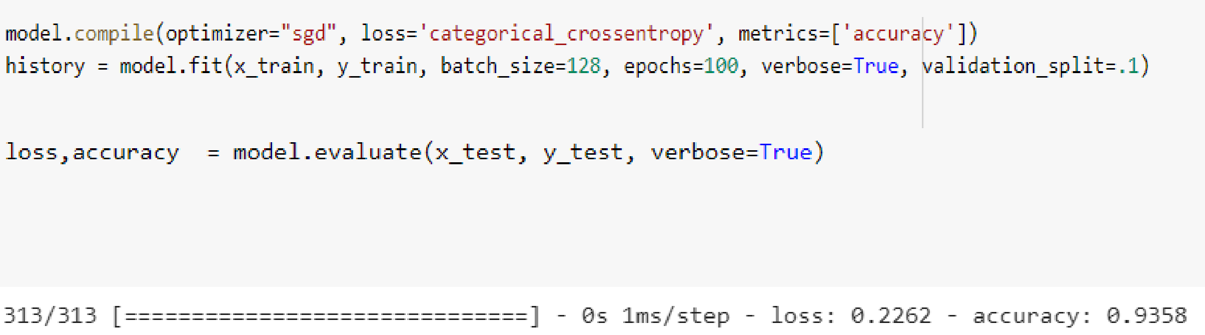


Layer with 25 neurons



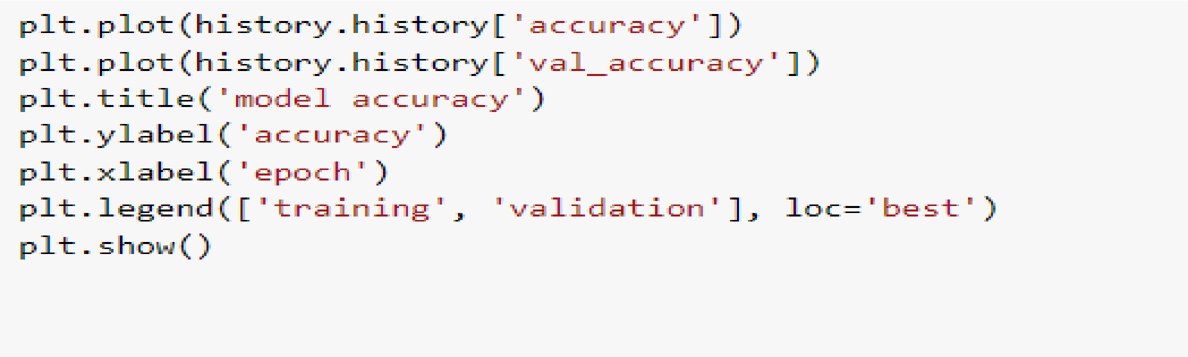
```
model = Sequential()

# The input layer requires the special input_shape parameter which should match
# the shape of our training data.
model.add(Dense(units=25, 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, 25)                19625     
_________________________________________________________________
dense_1 (Dense)              (None, 10)                260       
=================================================================
Total params: 19,885
Trainable params: 19,885
Non-trainable params: 0
_________________________________________________________________
```

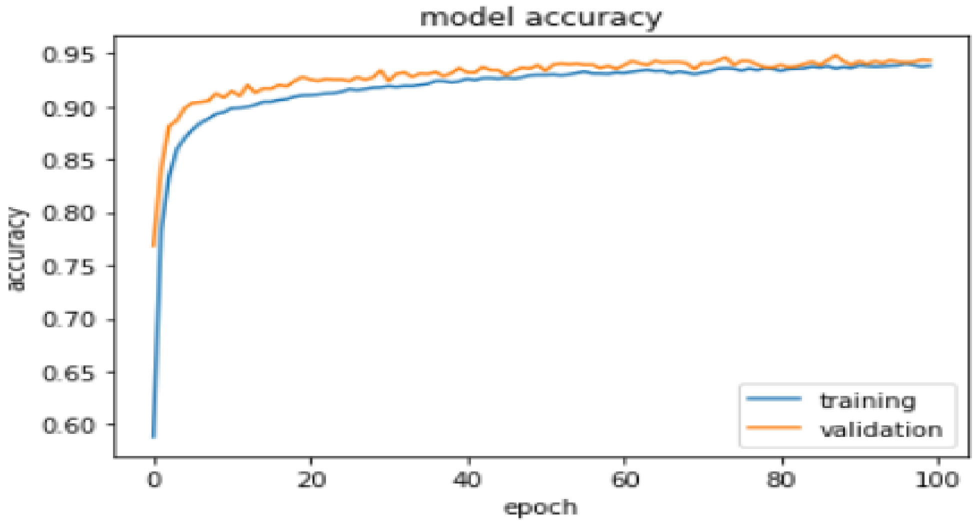


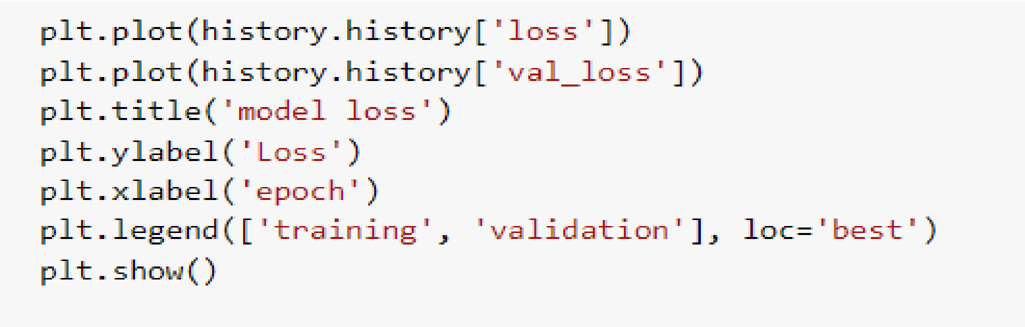
```
model.compile(optimizer="sgd", loss='categorical_crossentropy', metrics=['accuracy'])
history = model.fit(x_train, y_train, batch_size=128, epochs=100, verbose=True, validation_split=.1)


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

```
313/313 [==============================] - 0s 1ms/step - loss: 0.2262 - accuracy: 0.9358
```

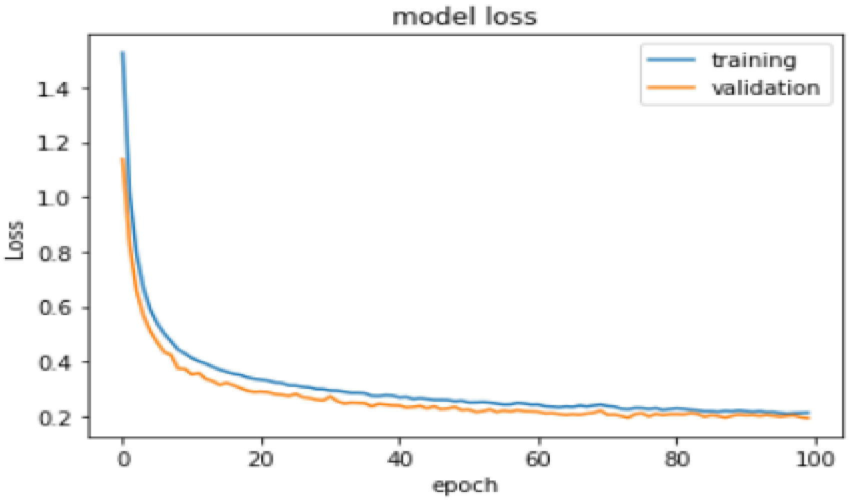


```
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['training', 'validation'], loc='best')
plt.show()
```

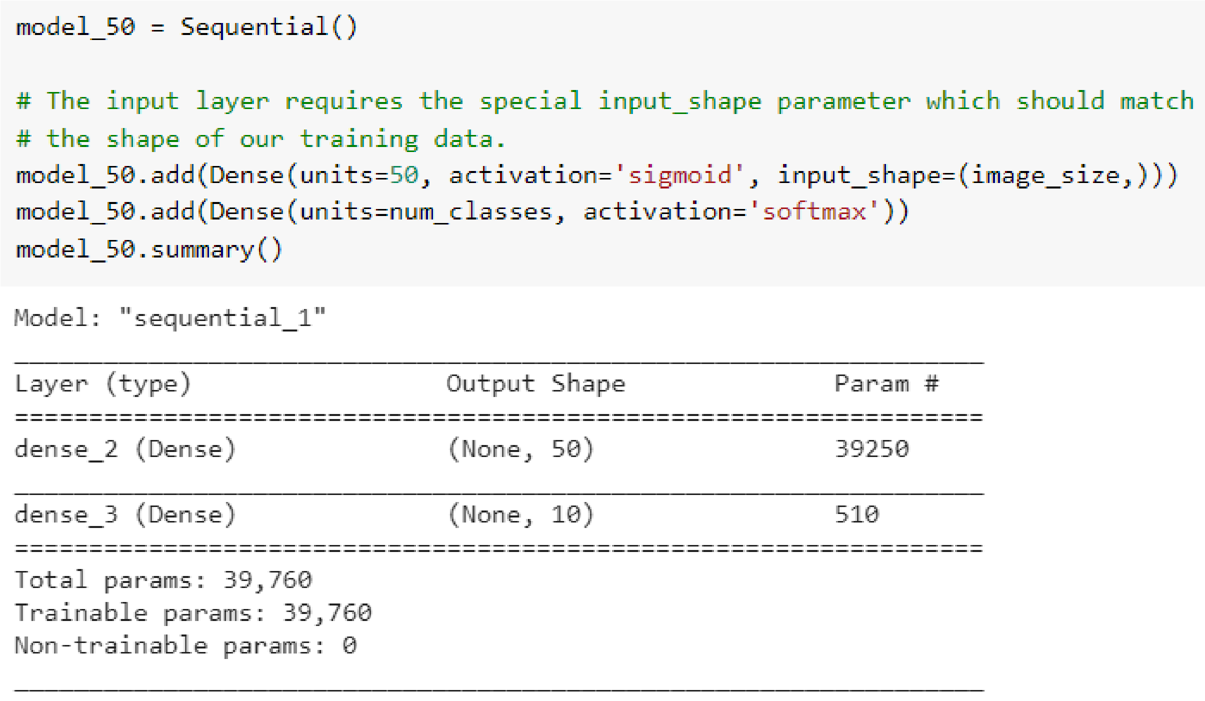




```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('Loss')
plt.xlabel('epoch')
plt.legend(['training', 'validation'], loc='best')
plt.show()
```

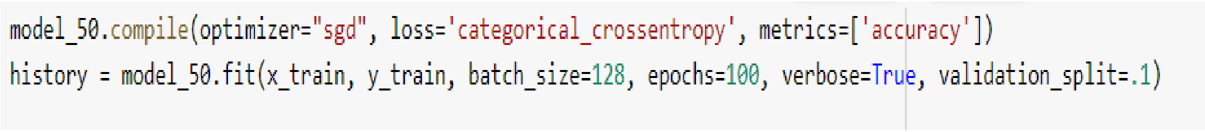


Layer with 50 neurons

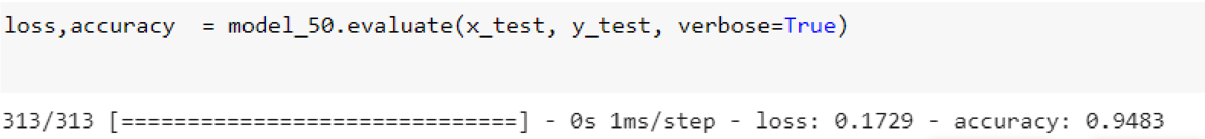


```
model_50 = Sequential()

# The input layer requires the special input_shape parameter which should match
# the shape of our training data.
model_50.add(Dense(units=50, activation='sigmoid', input_shape=(image_size,)))
model_50.add(Dense(units=num_classes, activation='softmax'))
model_50.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_2 (Dense)              (None, 50)                39250     
_________________________________________________________________
dense_3 (Dense)              (None, 10)                510       
=================================================================
Total params: 39,760
Trainable params: 39,760
Non-trainable params: 0
_________________________________________________________________
```

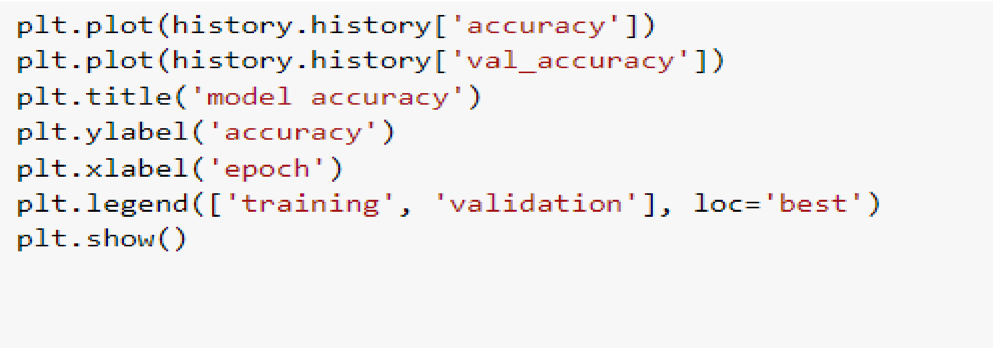


```
model_50.compile(optimizer="sgd", loss='categorical_crossentropy', metrics=['accuracy'])
history = model_50.fit(x_train, y_train, batch_size=128, epochs=100, verbose=True, validation_split=.1)
```

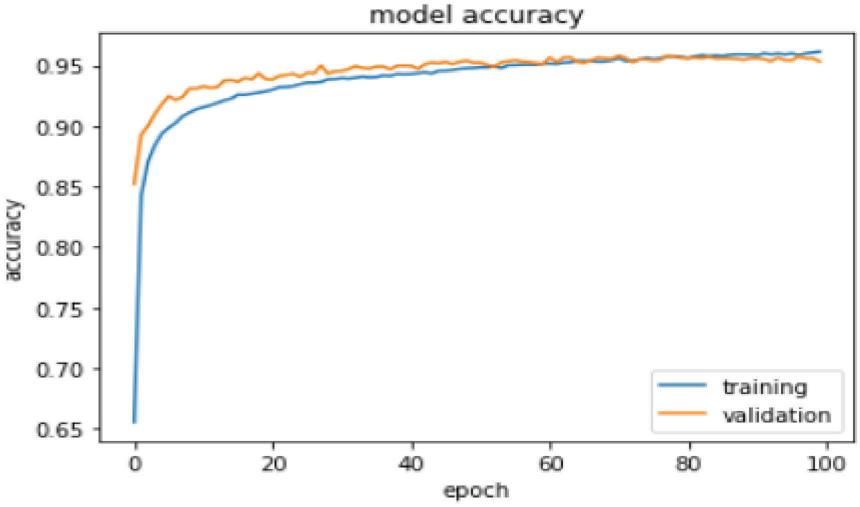


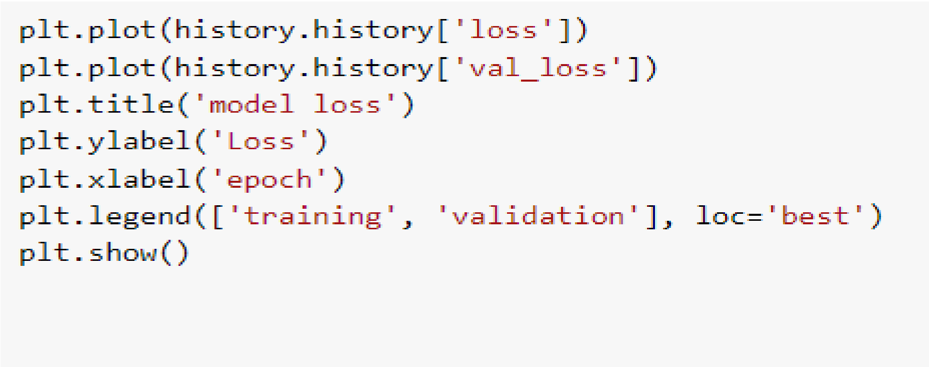
```
loss,accuracy  = model_50.evaluate(x_test, y_test, verbose=True)
```

```
313/313 [==============================] - 0s 1ms/step - loss: 0.1729 - accuracy: 0.9483
```

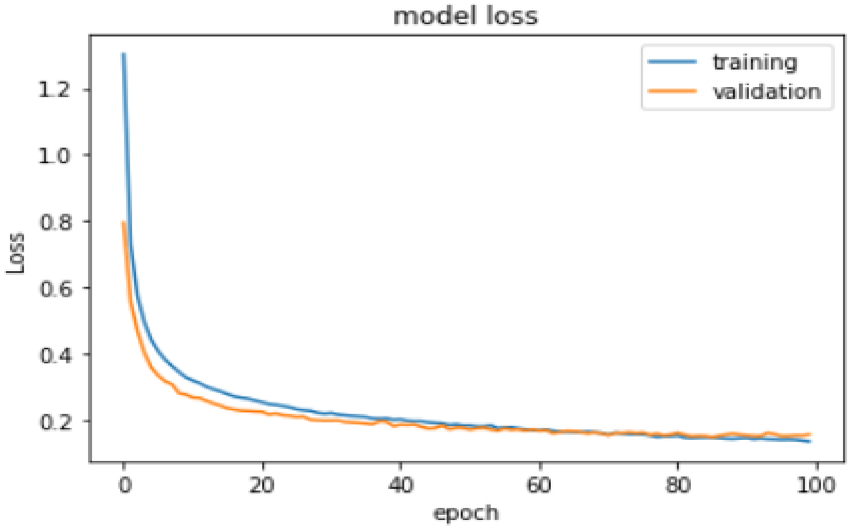


```
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['training', 'validation'], loc='best')
plt.show()
```

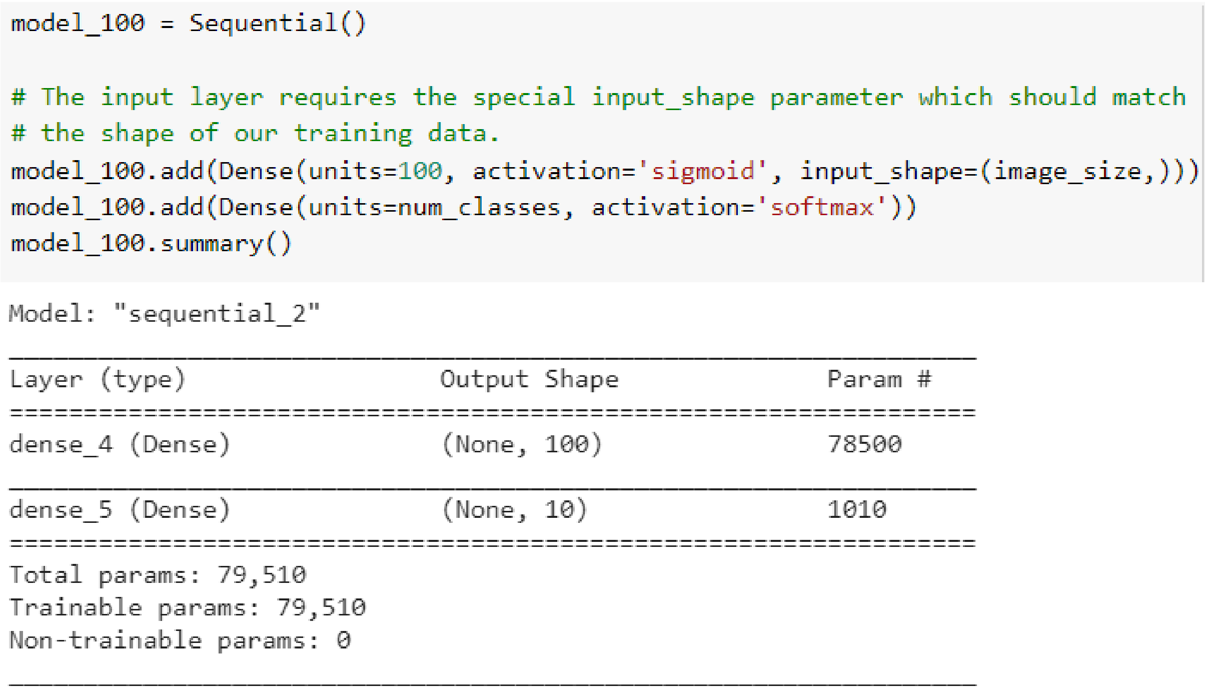




```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('Loss')
plt.xlabel('epoch')
plt.legend(['training', 'validation'], loc='best')
plt.show()
```

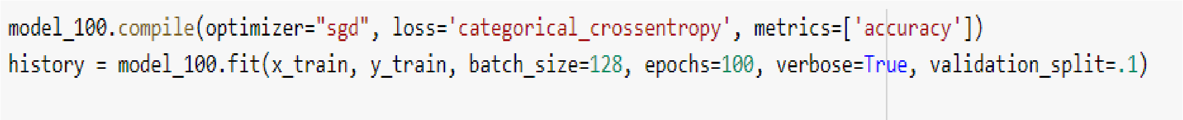


Layer with 100 neurons

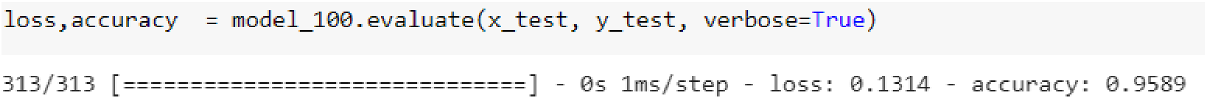


```
model_100 = Sequential()

# The input layer requires the special input_shape parameter which should match
# the shape of our training data.
model_100.add(Dense(units=100, activation='sigmoid', input_shape=(image_size,)))
model_100.add(Dense(units=num_classes, activation='softmax'))
model_100.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_4 (Dense)              (None, 100)               78500     
_________________________________________________________________
dense_5 (Dense)              (None, 10)                1010      
=================================================================
Total params: 79,510
Trainable params: 79,510
Non-trainable params: 0
_________________________________________________________________
```

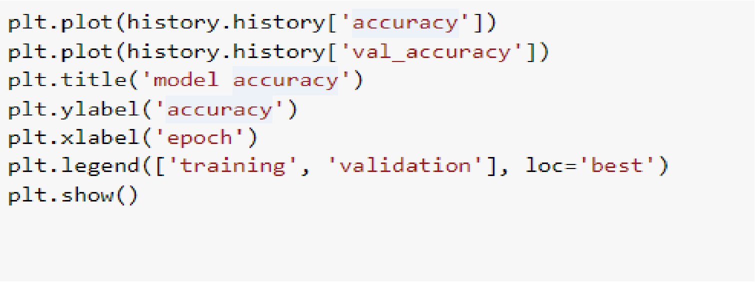


```
model_100.compile(optimizer="sgd", loss='categorical_crossentropy', metrics=['accuracy'])
history = model_100.fit(x_train, y_train, batch_size=128, epochs=100, verbose=True, validation_split=.1)
```

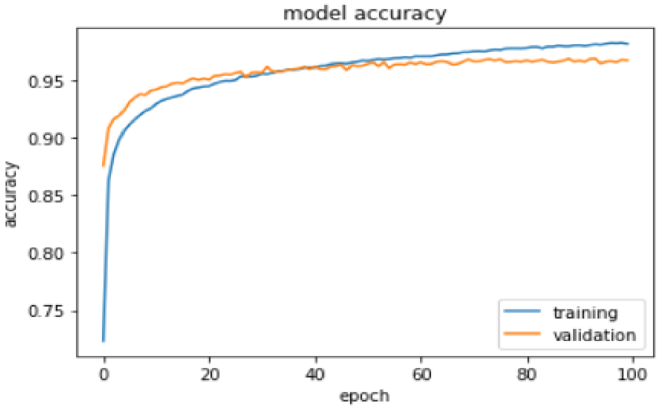


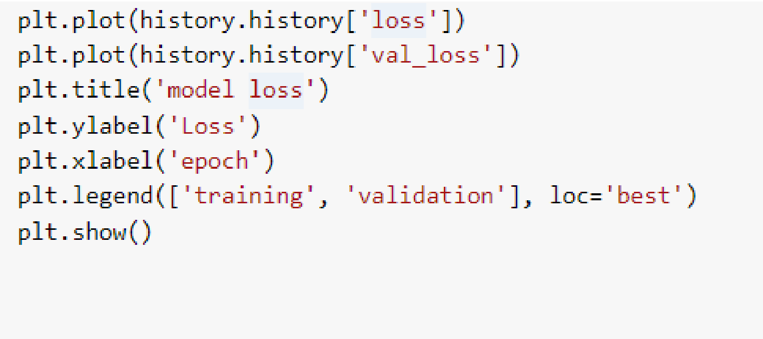
```
loss,accuracy  = model_100.evaluate(x_test, y_test, verbose=True)
```

```
313/313 [==============================] - 0s 1ms/step - loss: 0.1314 - accuracy: 0.9589
```

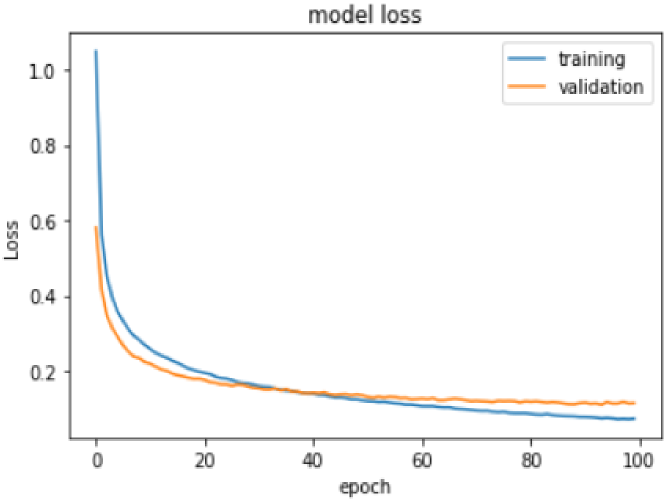


```
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['training', 'validation'], loc='best')
plt.show()
```

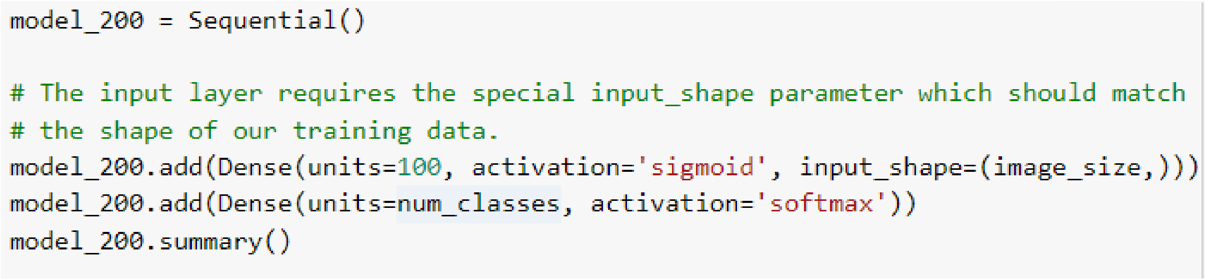




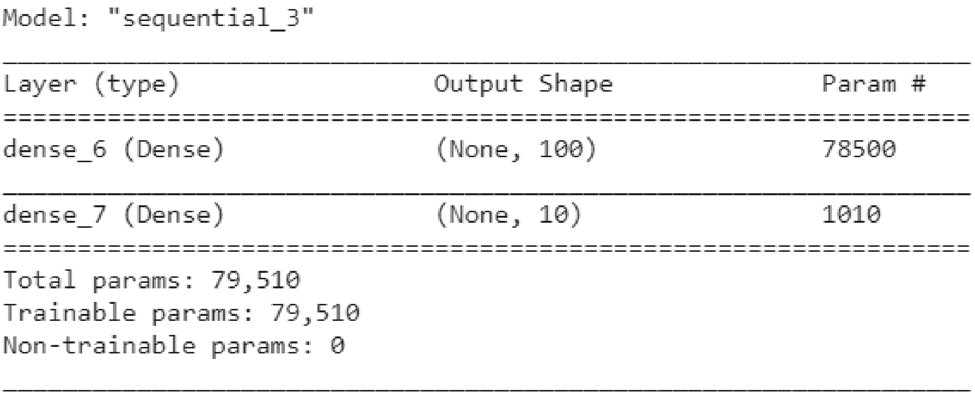
```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('Loss')
plt.xlabel('epoch')
plt.legend(['training', 'validation'], loc='best')
plt.show()
```



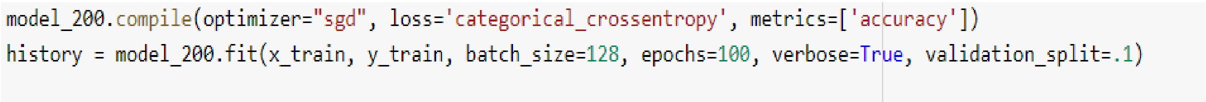
Layer with 200 neurons



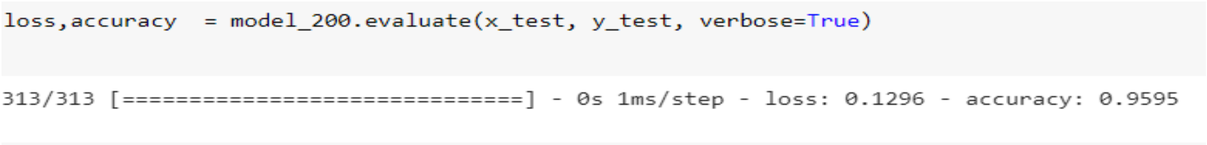
```
model_200 = Sequential()

# The input layer requires the special input_shape parameter which should match
# the shape of our training data.
model_200.add(Dense(units=100, activation='sigmoid', input_shape=(image_size,)))
model_200.add(Dense(units=num_classes, activation='softmax'))
model_200.summary()
```



```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_6 (Dense)              (None, 100)               78500     
_________________________________________________________________
dense_7 (Dense)              (None, 10)                1010      
=================================================================
Total params: 79,510
Trainable params: 79,510
Non-trainable params: 0
_________________________________________________________________
```

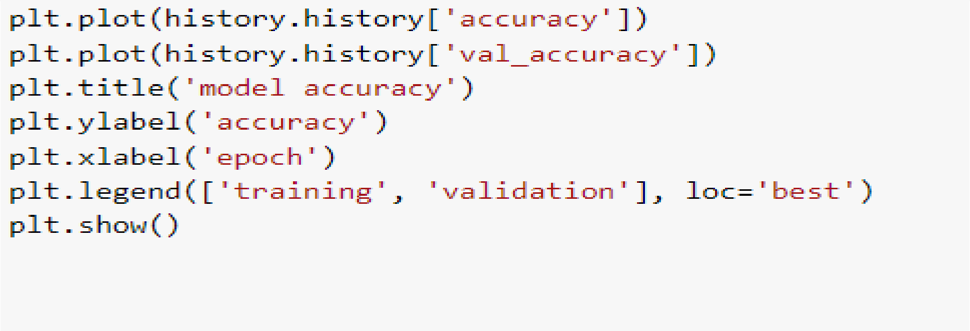


```
model_200.compile(optimizer="sgd", loss='categorical_crossentropy', metrics=['accuracy'])
history = model_200.fit(x_train, y_train, batch_size=128, epochs=100, verbose=True, validation_split=.1)
```

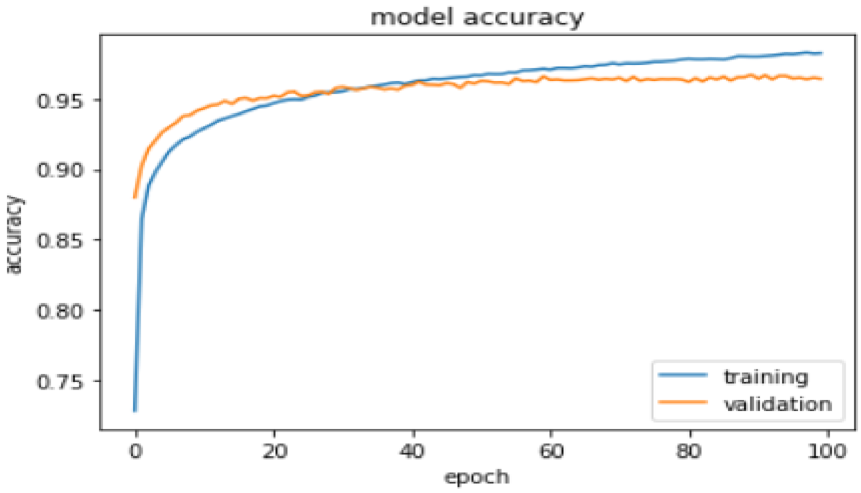


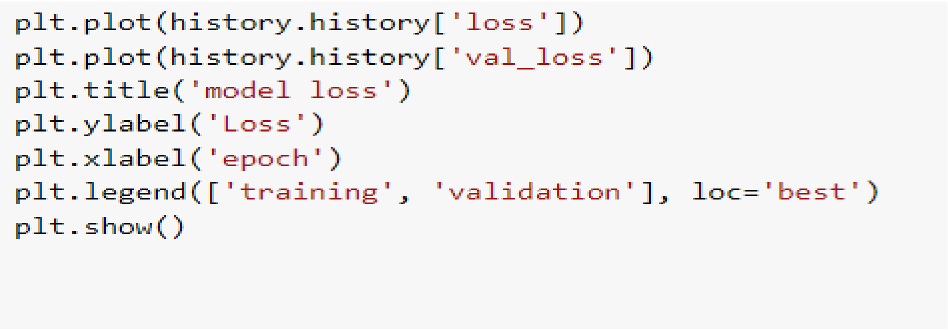
```
loss,accuracy  = model_200.evaluate(x_test, y_test, verbose=True)
```

```
313/313 [==============================] - 0s 1ms/step - loss: 0.1296 - accuracy: 0.9595
```

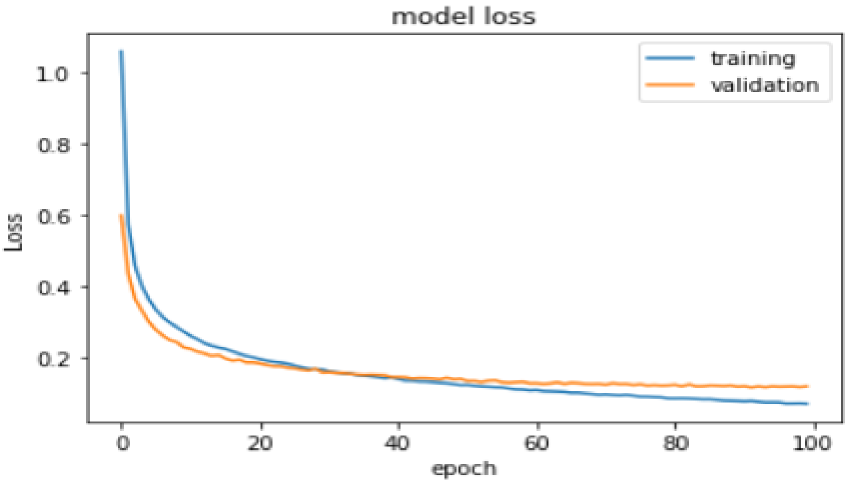


```
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['training', 'validation'], loc='best')
plt.show()
```





```
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