NEURAL NETWORK AND DEEP LEARNING

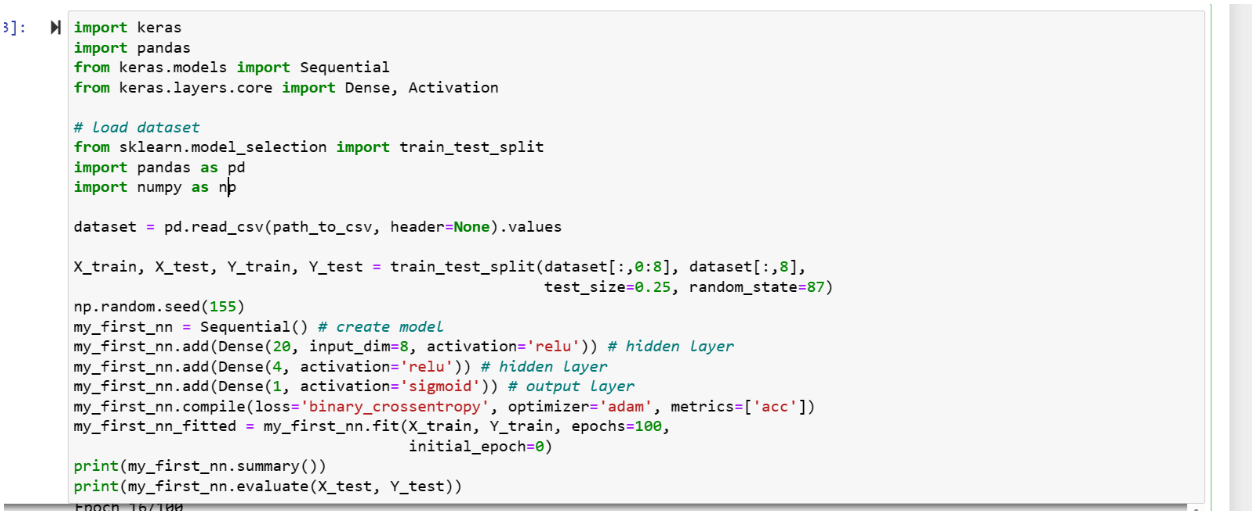
Assignment 2

37130432

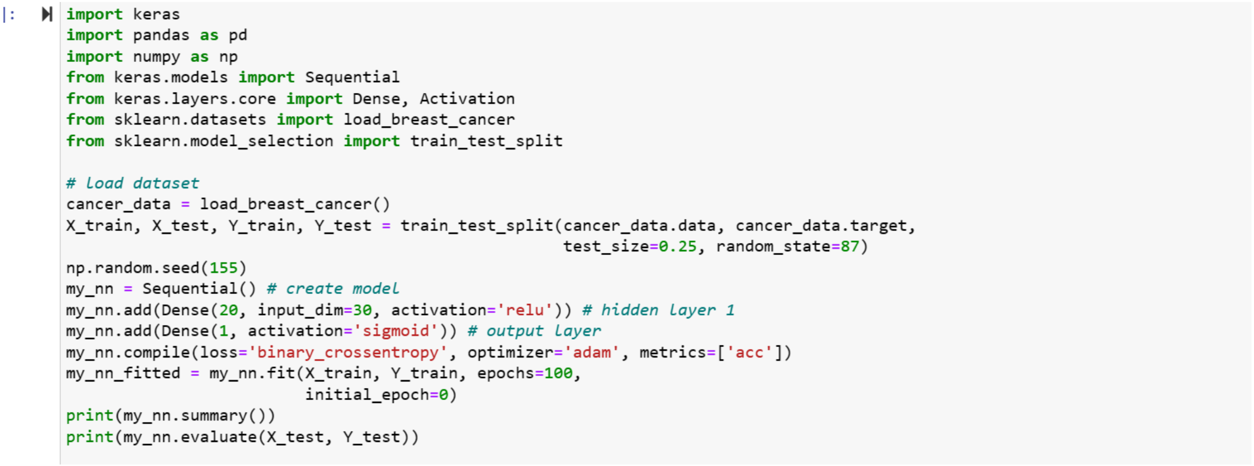
Github :https://github.com/vemparalahemasri/NNDL_Assignment2

ICP IN KERAS

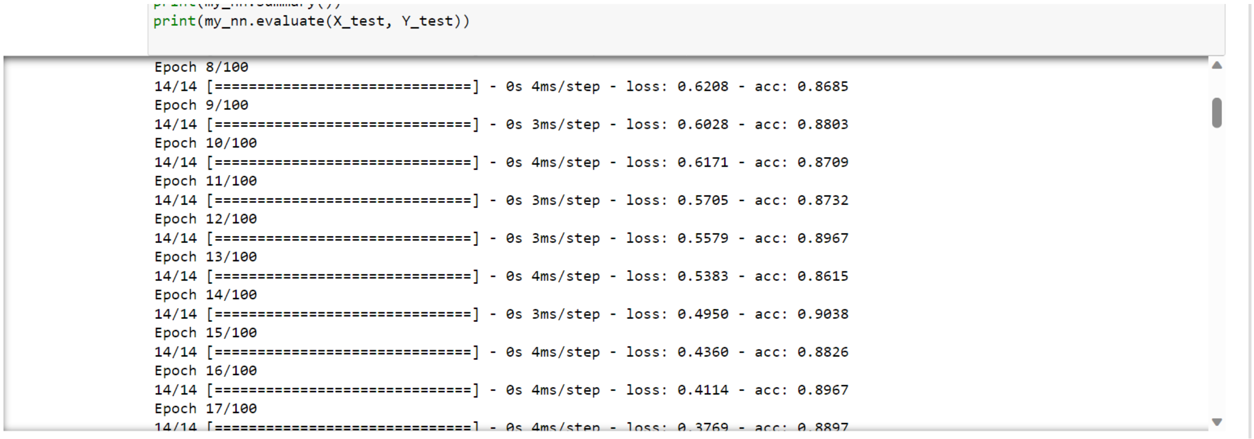
Applications of machine learning (ML) to invasively obtained ICP data are rare. The Scalzo et al. invasively obtained ICP data was subjected to ML algorithms. In comparison to numerous linear regression models and the adaptive boosting technique, they recommended utilising heavily randomised decision trees to predict intracranial hypertension using parameters linked to ICP wave morphology. A sequenced reliance among the ordered data becomes more complex due to time components inherently. Therefore, time series forecasting issues may cause conventional regression models to underperform.



```
3]:  import keras
     import pandas
     from keras.models import Sequential
     from keras.layers.core import Dense, Activation

     # load dataset
     from sklearn.model_selection import train_test_split
     import pandas as pd
     import numpy as np

     dataset = pd.read_csv(path_to_csv, header=None).values

     X_train, X_test, Y_train, Y_test = train_test_split(dataset[:,0:8], dataset[:,8],
                                                         test_size=0.25, random_state=87)
     np.random.seed(155)
     my_first_nn = Sequential() # create model
     my_first_nn.add(Dense(20, input_dim=8, activation='relu')) # hidden layer
     my_first_nn.add(Dense(4, activation='relu')) # hidden layer
     my_first_nn.add(Dense(1, activation='sigmoid')) # output layer
     my_first_nn.compile(loss='binary_crossentropy', optimizer='adam', metrics=['acc'])
     my_first_nn_fitted = my_first_nn.fit(X_train, Y_train, epochs=100,
                                          initial_epoch=0)
     print(my_first_nn.summary())
     print(my_first_nn.evaluate(X_test, Y_test))
     Epoch 16/100
```

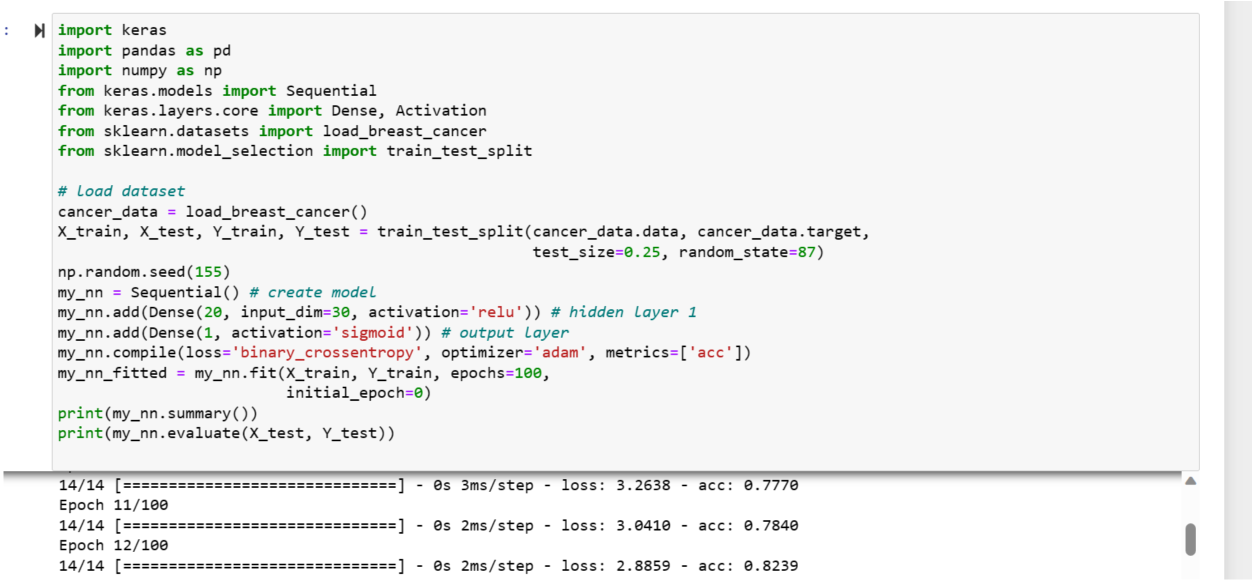


```
[ ]:
import keras
import pandas as pd
import numpy as np
from keras.models import Sequential
from keras.layers.core import Dense, Activation
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split

# load dataset
cancer_data = load_breast_cancer()
X_train, X_test, Y_train, Y_test = train_test_split(cancer_data.data, cancer_data.target,
                                                     test_size=0.25, random_state=87)
np.random.seed(155)
my_nn = Sequential() # create model
my_nn.add(Dense(20, input_dim=30, activation='relu')) # hidden layer 1
my_nn.add(Dense(1, activation='sigmoid')) # output layer
my_nn.compile(loss='binary_crossentropy', optimizer='adam', metrics=['acc'])
my_nn_fitted = my_nn.fit(X_train, Y_train, epochs=100,
                         initial_epoch=0)
print(my_nn.summary())
print(my_nn.evaluate(X_test, Y_test))
```



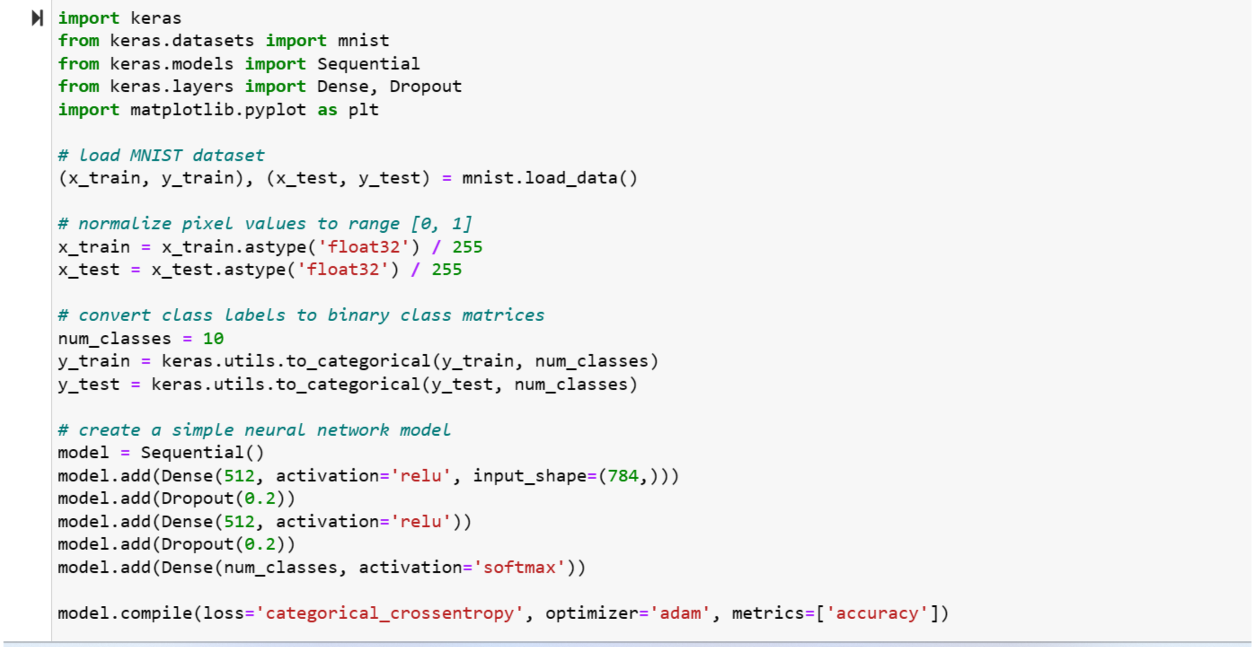
```
print(my_nn.summary())
print(my_nn.evaluate(X_test, Y_test))
```

```
Epoch 8/100
14/14 [==============================] - 0s 4ms/step - loss: 0.6208 - acc: 0.8685
Epoch 9/100
14/14 [==============================] - 0s 3ms/step - loss: 0.6028 - acc: 0.8803
Epoch 10/100
14/14 [==============================] - 0s 4ms/step - loss: 0.6171 - acc: 0.8709
Epoch 11/100
14/14 [==============================] - 0s 3ms/step - loss: 0.5705 - acc: 0.8732
Epoch 12/100
14/14 [==============================] - 0s 3ms/step - loss: 0.5579 - acc: 0.8967
Epoch 13/100
14/14 [==============================] - 0s 4ms/step - loss: 0.5383 - acc: 0.8615
Epoch 14/100
14/14 [==============================] - 0s 3ms/step - loss: 0.4950 - acc: 0.9038
Epoch 15/100
14/14 [==============================] - 0s 4ms/step - loss: 0.4360 - acc: 0.8826
Epoch 16/100
14/14 [==============================] - 0s 4ms/step - loss: 0.4114 - acc: 0.8967
Epoch 17/100
14/14 [==============================] - 0s 4ms/step - loss: 0.3769 - acc: 0.8897
```

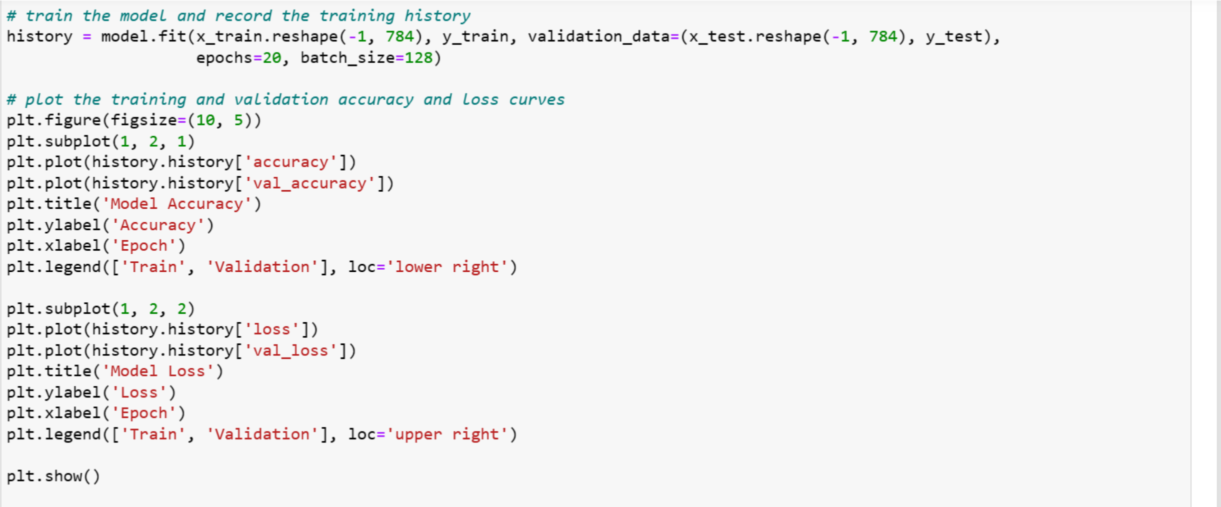


```
[ ]:
import keras
import pandas as pd
import numpy as np
from keras.models import Sequential
from keras.layers.core import Dense, Activation
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split

# load dataset
cancer_data = load_breast_cancer()
X_train, X_test, Y_train, Y_test = train_test_split(cancer_data.data, cancer_data.target,
                                                     test_size=0.25, random_state=87)
np.random.seed(155)
my_nn = Sequential() # create model
my_nn.add(Dense(20, input_dim=30, activation='relu')) # hidden layer 1
my_nn.add(Dense(1, activation='sigmoid')) # output layer
my_nn.compile(loss='binary_crossentropy', optimizer='adam', metrics=['acc'])
my_nn_fitted = my_nn.fit(X_train, Y_train, epochs=100,
                         initial_epoch=0)
print(my_nn.summary())
print(my_nn.evaluate(X_test, Y_test))
```

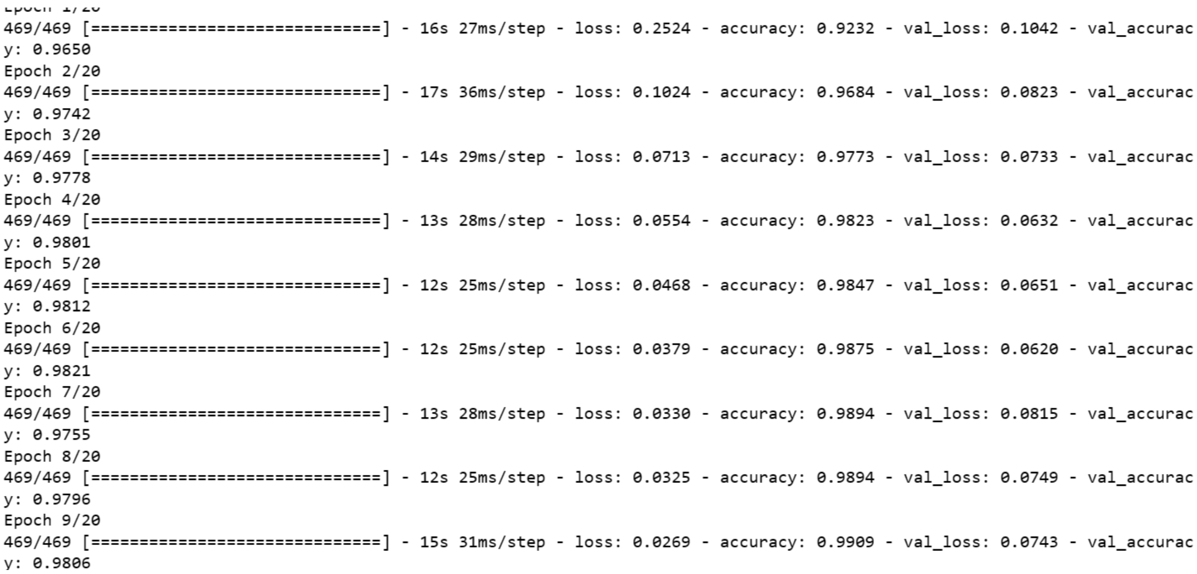
```
14/14 [==============================] - 0s 3ms/step - loss: 3.2638 - acc: 0.7770
Epoch 11/100
14/14 [==============================] - 0s 2ms/step - loss: 3.0410 - acc: 0.7840
Epoch 12/100
14/14 [==============================] - 0s 2ms/step - loss: 2.8859 - acc: 0.8239
```



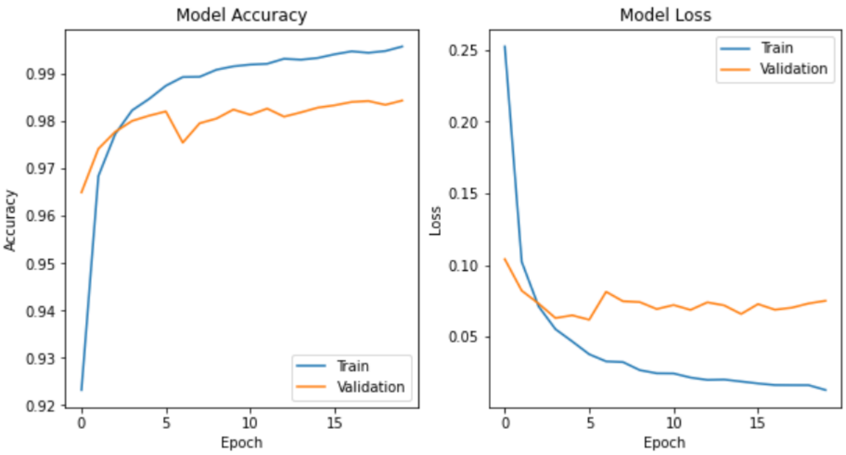
```
import keras
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Dropout
import matplotlib.pyplot as plt

# Load MNIST dataset
(x_train, y_train), (x_test, y_test) = mnist.load_data()

# normalize pixel values to range [0, 1]
x_train = x_train.astype('float32') / 255
x_test = x_test.astype('float32') / 255

# convert class labels 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)

# create a simple neural network model
model = Sequential()
model.add(Dense(512, activation='relu', input_shape=(784,)))
model.add(Dropout(0.2))
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(num_classes, activation='softmax'))

model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

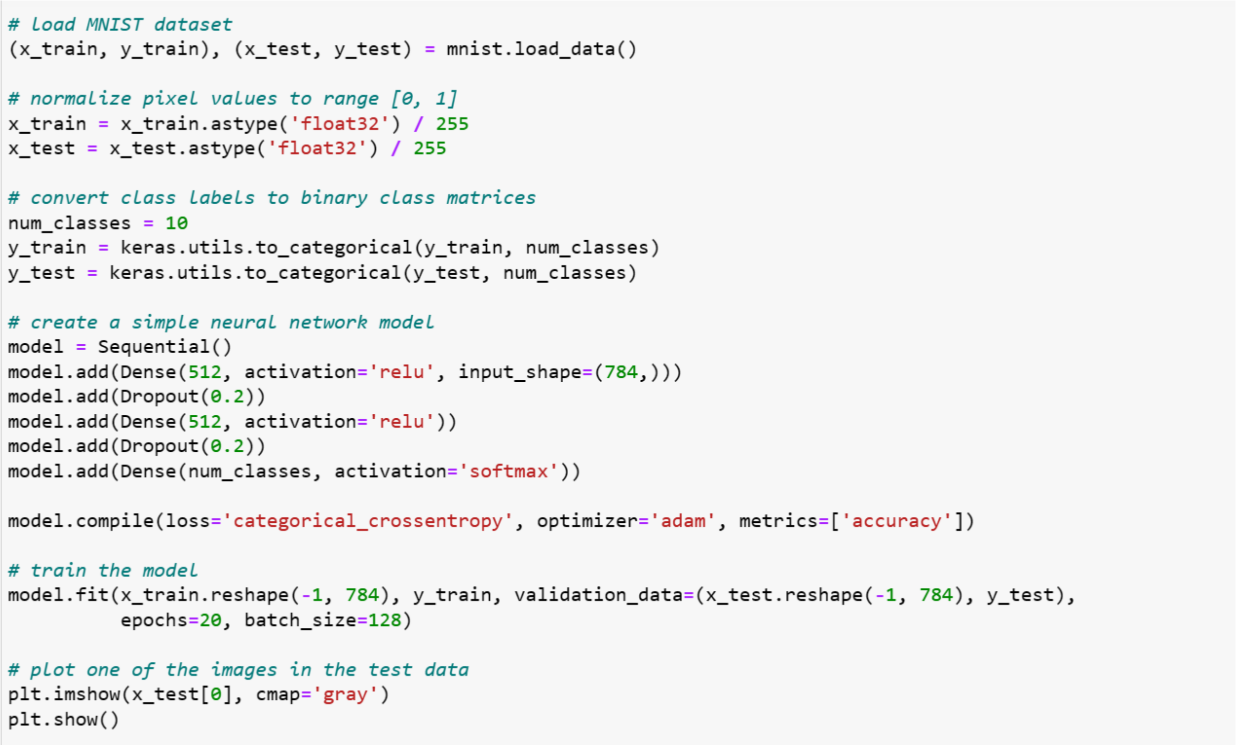


```
# train the model and record the training history
history = model.fit(x_train.reshape(-1, 784), y_train, validation_data=(x_test.reshape(-1, 784), y_test),
                    epochs=20, batch_size=128)

# plot the training and validation accuracy and loss curves
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='lower right')

plt.subplot(1, 2, 2)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper right')

plt.show()
```

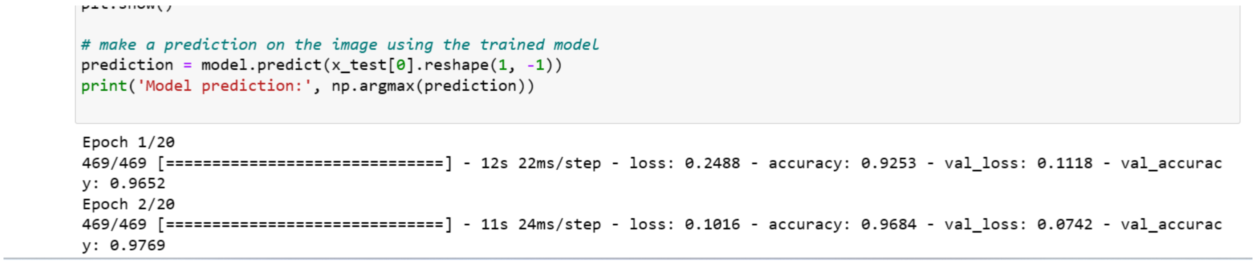


```
Epoch 1/20
469/469 [==============================] - 16s 27ms/step - loss: 0.2524 - accuracy: 0.9232 - val_loss: 0.1042 - val_accurac
y: 0.9650
Epoch 2/20
469/469 [==============================] - 17s 36ms/step - loss: 0.1024 - accuracy: 0.9684 - val_loss: 0.0823 - val_accurac
y: 0.9742
Epoch 3/20
469/469 [==============================] - 14s 29ms/step - loss: 0.0713 - accuracy: 0.9773 - val_loss: 0.0733 - val_accurac
y: 0.9778
Epoch 4/20
469/469 [==============================] - 13s 28ms/step - loss: 0.0554 - accuracy: 0.9823 - val_loss: 0.0632 - val_accurac
y: 0.9801
Epoch 5/20
469/469 [==============================] - 12s 25ms/step - loss: 0.0468 - accuracy: 0.9847 - val_loss: 0.0651 - val_accurac
y: 0.9812
Epoch 6/20
469/469 [==============================] - 12s 25ms/step - loss: 0.0379 - accuracy: 0.9875 - val_loss: 0.0620 - val_accurac
y: 0.9821
Epoch 7/20
469/469 [==============================] - 13s 28ms/step - loss: 0.0330 - accuracy: 0.9894 - val_loss: 0.0815 - val_accurac
y: 0.9755
Epoch 8/20
469/469 [==============================] - 12s 25ms/step - loss: 0.0325 - accuracy: 0.9894 - val_loss: 0.0749 - val_accurac
y: 0.9796
Epoch 9/20
469/469 [==============================] - 15s 31ms/step - loss: 0.0269 - accuracy: 0.9909 - val_loss: 0.0743 - val_accurac
y: 0.9806
```



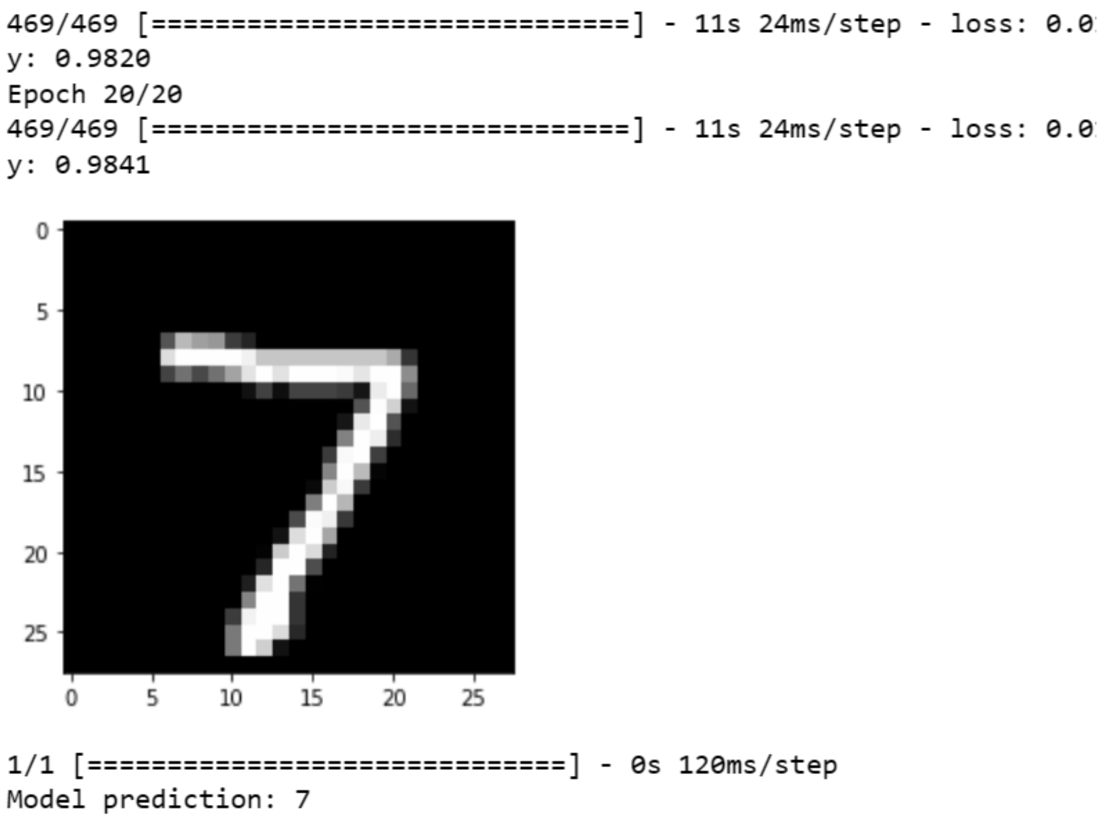


```
# load MNIST dataset
(x_train, y_train), (x_test, y_test) = mnist.load_data()

# normalize pixel values to range [0, 1]
x_train = x_train.astype('float32') / 255
x_test = x_test.astype('float32') / 255

# convert class labels 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)

# create a simple neural network model
model = Sequential()
model.add(Dense(512, activation='relu', input_shape=(784,)))
model.add(Dropout(0.2))
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(num_classes, activation='softmax'))

model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

# train the model
model.fit(x_train.reshape(-1, 784), y_train, validation_data=(x_test.reshape(-1, 784), y_test),
          epochs=20, batch_size=128)

# plot one of the images in the test data
plt.imshow(x_test[0], cmap='gray')
plt.show()
```



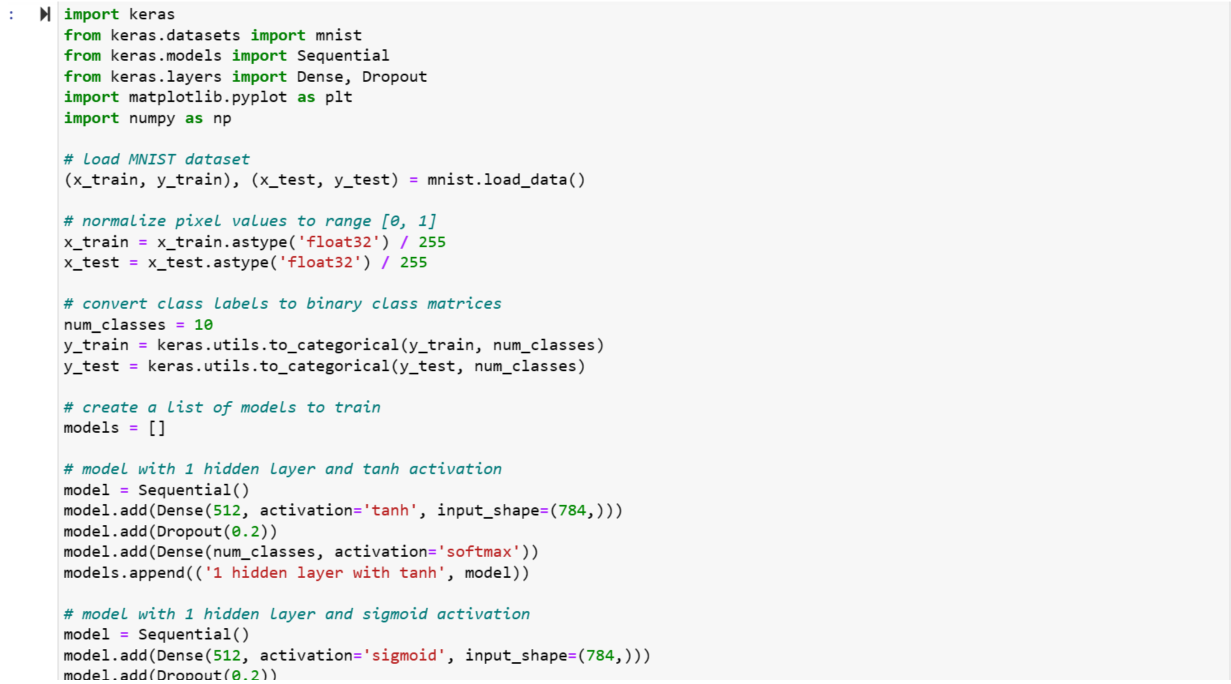
```
plt.show()

# make a prediction on the image using the trained model
prediction = model.predict(x_test[0].reshape(1, -1))
print('Model prediction:', np.argmax(prediction))
```

```
Epoch 1/20
469/469 [==============================] - 12s 22ms/step - loss: 0.2488 - accuracy: 0.9253 - val_loss: 0.1118 - val_accurac
y: 0.9652
Epoch 2/20
469/469 [==============================] - 11s 24ms/step - loss: 0.1016 - accuracy: 0.9684 - val_loss: 0.0742 - val_accurac
y: 0.9769
```

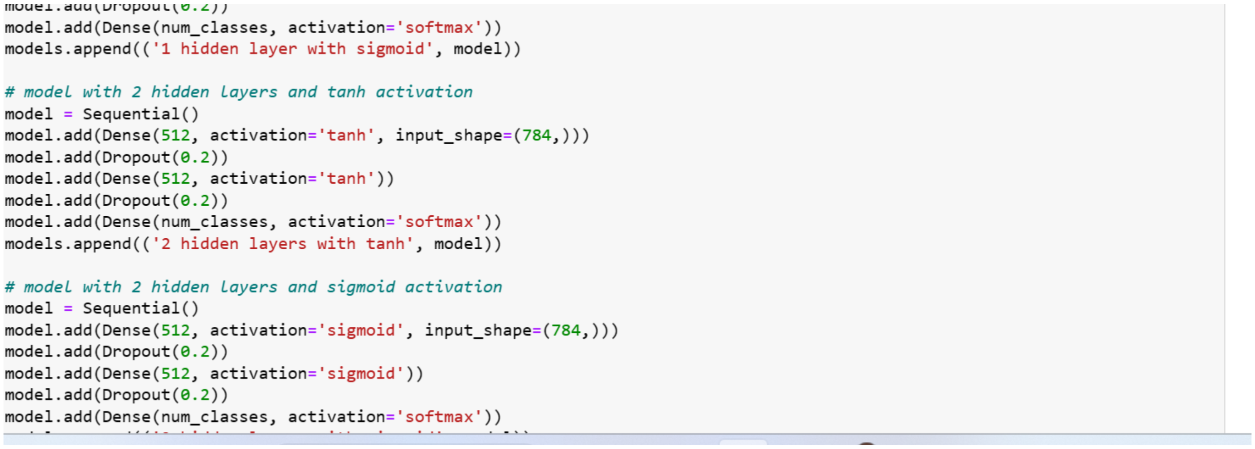


```
469/469 [==============================] - 11s 24ms/step - loss: 0.0
y: 0.9820
Epoch 20/20
469/469 [==============================] - 11s 24ms/step - loss: 0.0
y: 0.9841
```

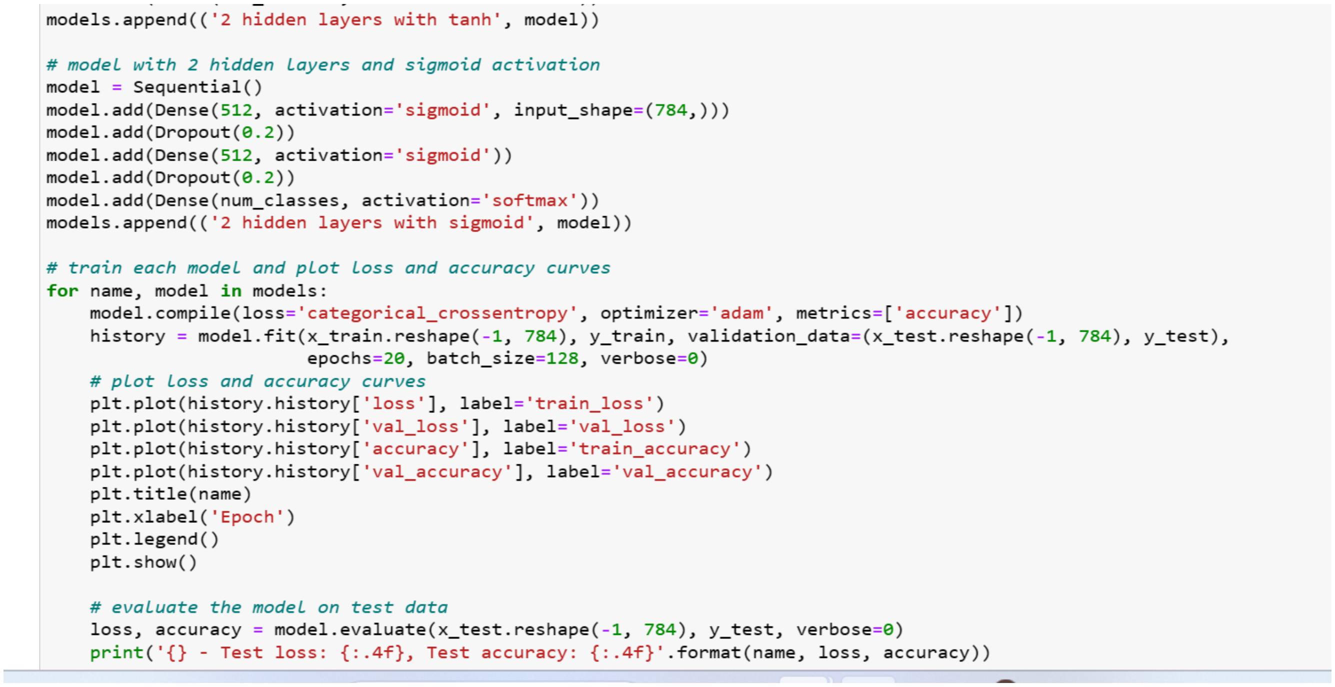
```
1/1 [==============================] - 0s 120ms/step
Model prediction: 7
```



```
import keras
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Dropout
import matplotlib.pyplot as plt
import numpy as np

# Load MNIST dataset
(x_train, y_train), (x_test, y_test) = mnist.load_data()

# normalize pixel values to range [0, 1]
x_train = x_train.astype('float32') / 255
x_test = x_test.astype('float32') / 255

# convert class labels 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)

# create a list of models to train
models = []

# model with 1 hidden layer and tanh activation
model = Sequential()
model.add(Dense(512, activation='tanh', input_shape=(784,)))
model.add(Dropout(0.2))
model.add(Dense(num_classes, activation='softmax'))
models.append(('1 hidden layer with tanh', model))

# model with 1 hidden layer and sigmoid activation
model = Sequential()
model.add(Dense(512, activation='sigmoid', input_shape=(784,)))
model.add(Dropout(0.2))
```

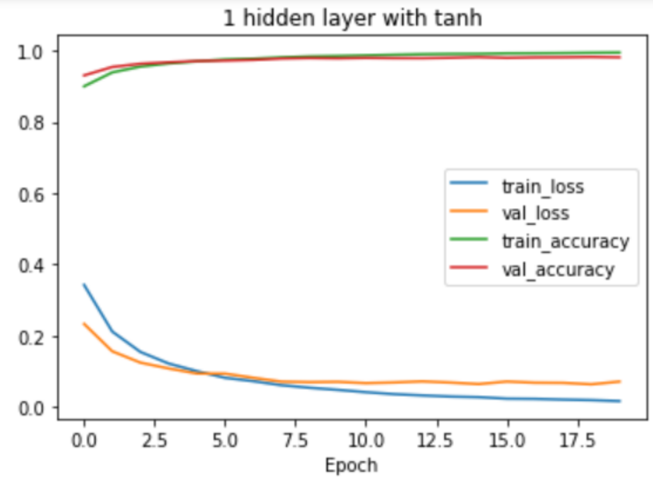


```
model.add(Dropout(0.2))
model.add(Dense(num_classes, activation='softmax'))
models.append(('1 hidden layer with sigmoid', model))

# model with 2 hidden layers and tanh activation
model = Sequential()
model.add(Dense(512, activation='tanh', input_shape=(784,)))
model.add(Dropout(0.2))
model.add(Dense(512, activation='tanh'))
model.add(Dropout(0.2))
model.add(Dense(num_classes, activation='softmax'))
models.append(('2 hidden layers with tanh', model))

# model with 2 hidden layers and sigmoid activation
model = Sequential()
model.add(Dense(512, activation='sigmoid', input_shape=(784,)))
model.add(Dropout(0.2))
model.add(Dense(512, activation='sigmoid'))
model.add(Dropout(0.2))
model.add(Dense(num_classes, activation='softmax'))
```

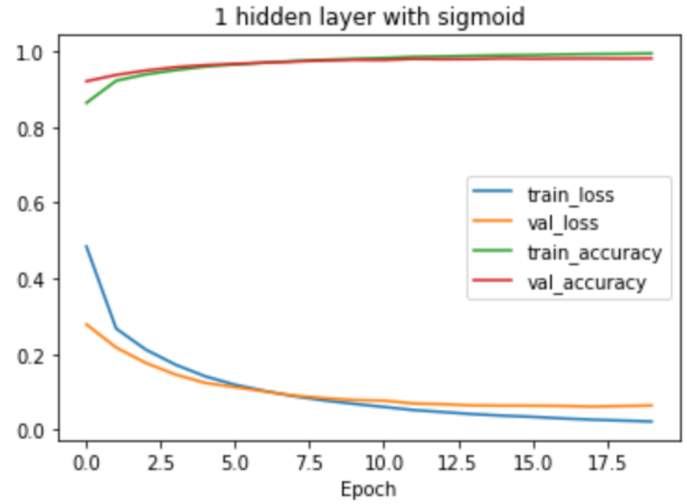


```
models.append(('2 hidden layers with tanh', model))

# model with 2 hidden layers and sigmoid activation
model = Sequential()
model.add(Dense(512, activation='sigmoid', input_shape=(784,)))
model.add(Dropout(0.2))
model.add(Dense(512, activation='sigmoid'))
model.add(Dropout(0.2))
model.add(Dense(num_classes, activation='softmax'))
```

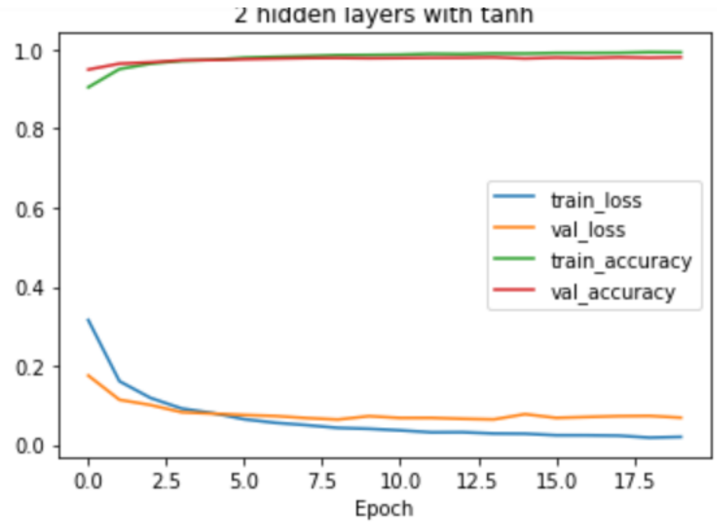
```
models.append(('2 hidden layers with sigmoid', model))

# train each model and plot loss and accuracy curves
for name, model in models:
    model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
    history = model.fit(x_train.reshape(-1, 784), y_train, validation_data=(x_test.reshape(-1, 784), y_test),
                        epochs=20, batch_size=128, verbose=0)
    # plot loss and accuracy curves
    plt.plot(history.history['loss'], label='train_loss')
    plt.plot(history.history['val_loss'], label='val_loss')
    plt.plot(history.history['accuracy'], label='train_accuracy')
    plt.plot(history.history['val_accuracy'], label='val_accuracy')
    plt.title(name)
    plt.xlabel('Epoch')
    plt.legend()
    plt.show()

    # evaluate the model on test data
    loss, accuracy = model.evaluate(x_test.reshape(-1, 784), y_test, verbose=0)
    print('{} - Test loss: {:.4f}, Test accuracy: {:.4f}'.format(name, loss, accuracy))
```



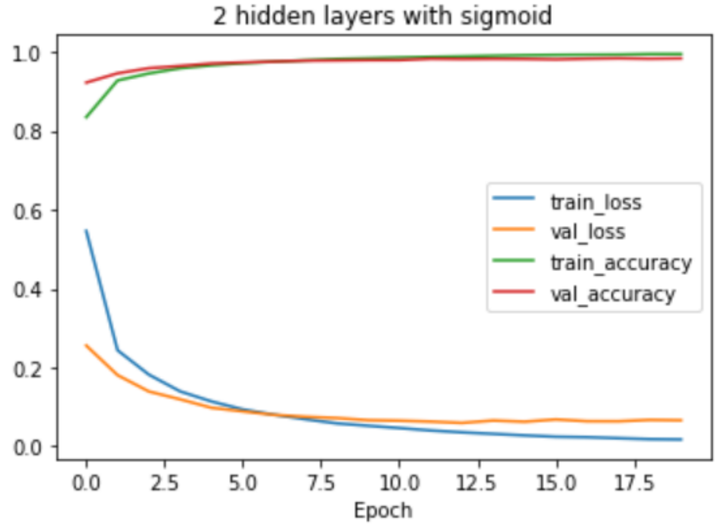
```
1 hidden layer with tanh - Test loss: 0.0716, Test accuracy: 0.9809
```



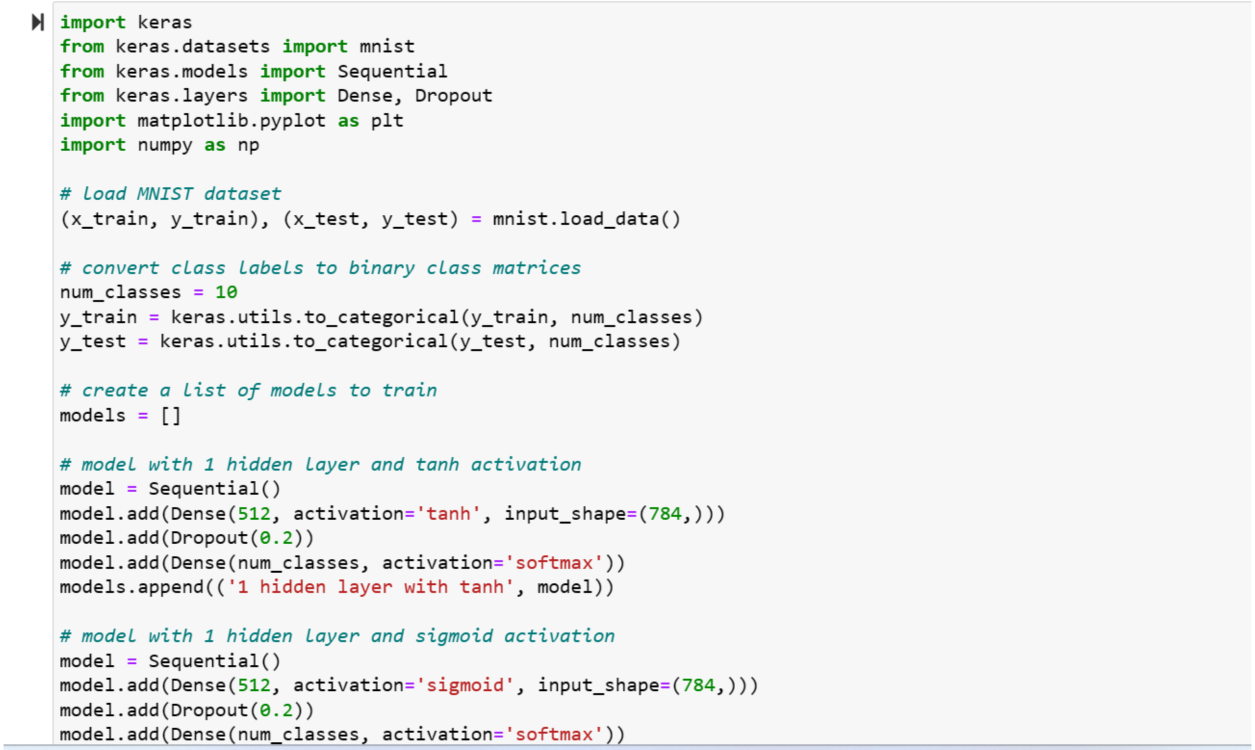
```
1 hidden layer with sigmoid - Test loss: 0.0642, Test accuracy: 0.9809
```



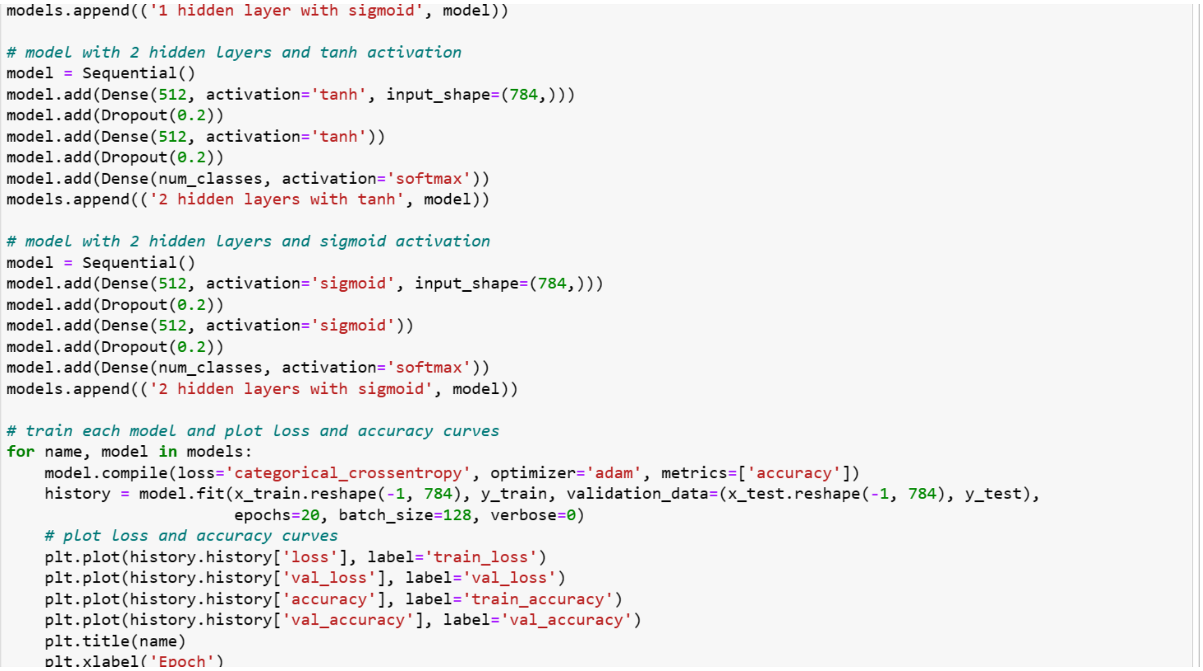
```
2 hidden layers with tanh - Test loss: 0.0686, Test accuracy: 0.9808
```



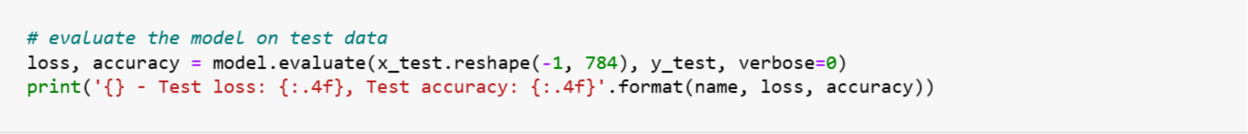
```
2 hidden layers with sigmoid - Test loss: 0.0663, Test accuracy: 0.9830
```



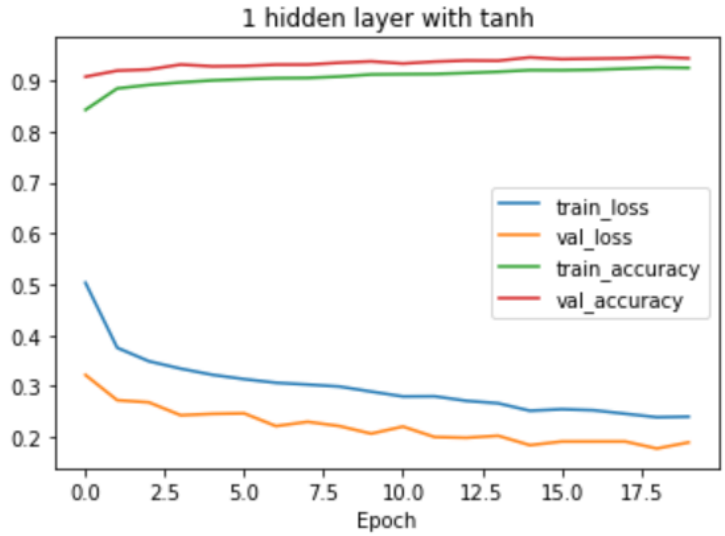
```python
import keras
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Dropout
import matplotlib.pyplot as plt
import numpy as np

# load MNIST dataset
(x_train, y_train), (x_test, y_test) = mnist.load_data()

# convert class labels 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)

# create a list of models to train
models = []

# model with 1 hidden layer and tanh activation
model = Sequential()
model.add(Dense(512, activation='tanh', input_shape=(784,)))
model.add(Dropout(0.2))
model.add(Dense(num_classes, activation='softmax'))
models.append(('1 hidden layer with tanh', model))

# model with 1 hidden layer and sigmoid activation
model = Sequential()
model.add(Dense(512, activation='sigmoid', input_shape=(784,)))
model.add(Dropout(0.2))
model.add(Dense(num_classes, activation='softmax'))
```



```python
models.append(('1 hidden layer with sigmoid', model))

# model with 2 hidden layers and tanh activation
model = Sequential()
model.add(Dense(512, activation='tanh', input_shape=(784,)))
model.add(Dropout(0.2))
model.add(Dense(512, activation='tanh'))
model.add(Dropout(0.2))
model.add(Dense(num_classes, activation='softmax'))
models.append(('2 hidden layers with tanh', model))

# model with 2 hidden layers and sigmoid activation
model = Sequential()
model.add(Dense(512, activation='sigmoid', input_shape=(784,)))
model.add(Dropout(0.2))
model.add(Dense(512, activation='sigmoid'))
model.add(Dropout(0.2))
model.add(Dense(num_classes, activation='softmax'))
models.append(('2 hidden layers with sigmoid', model))

# train each model and plot loss and accuracy curves
for name, model in models:
    model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
    history = model.fit(x_train.reshape(-1, 784), y_train, validation_data=(x_test.reshape(-1, 784), y_test),
                        epochs=20, batch_size=128, verbose=0)
    # plot loss and accuracy curves
    plt.plot(history.history['loss'], label='train_loss')
    plt.plot(history.history['val_loss'], label='val_loss')
    plt.plot(history.history['accuracy'], label='train_accuracy')
    plt.plot(history.history['val_accuracy'], label='val_accuracy')
    plt.title(name)
    plt.xlabel('Epoch')
```

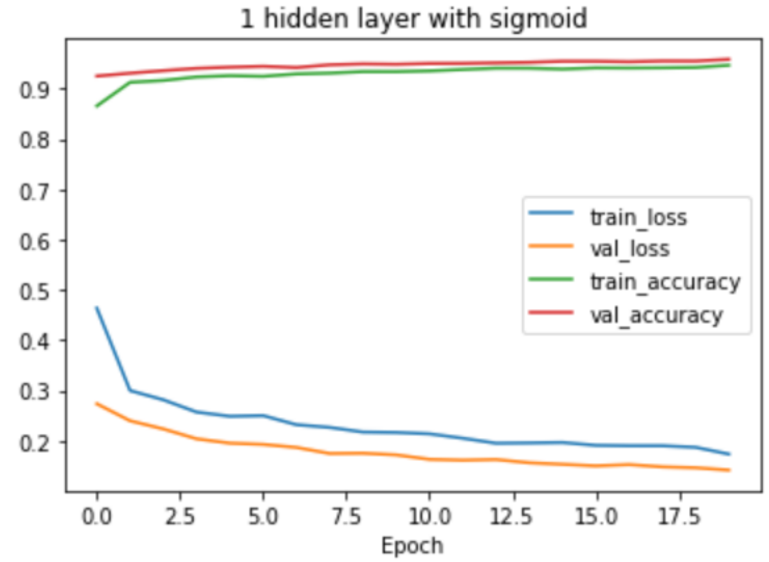


```
# evaluate the model on test data
loss, accuracy = model.evaluate(x_test.reshape(-1, 784), y_test, verbose=0)
print('{} - Test loss: {:.4f}, Test accuracy: {:.4f}'.format(name, loss, accuracy))
```



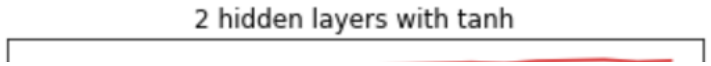
```
1 hidden layer with tanh - Test loss: 0.1895, Test accuracy: 0.9439
```

The sigmoid activation function (also called logistic function) takes any real value as input and outputs a value in the range

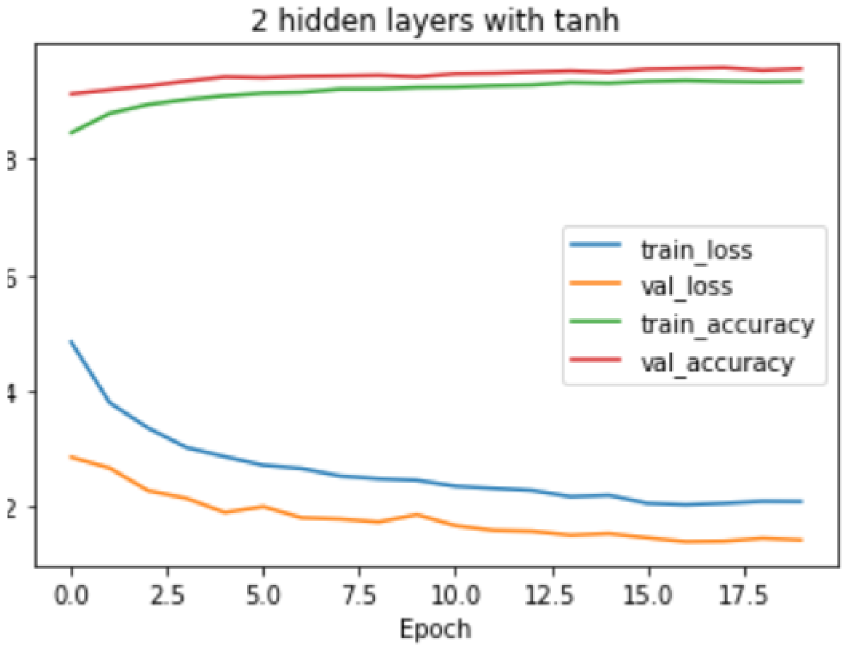


1 hidden layer with sigmoid

```
1 hidden layer with sigmoid - Test loss: 0.1420, Test accuracy: 0.9582
```

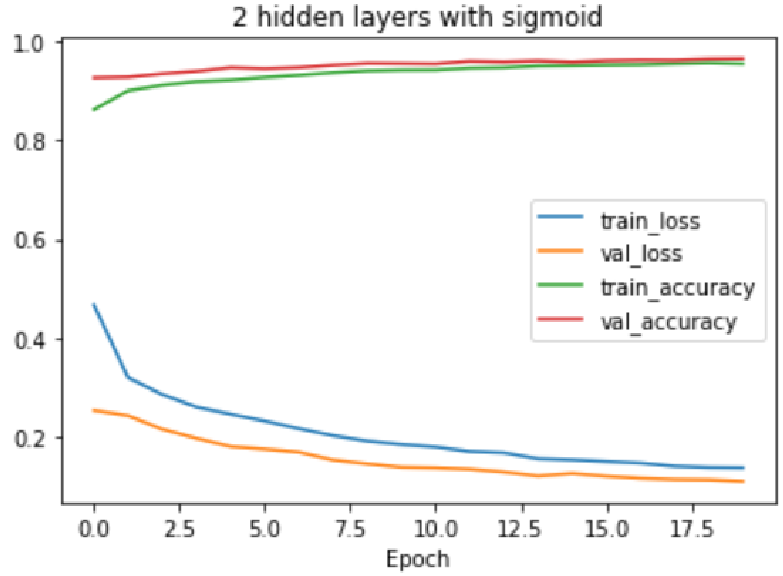


2 hidden layers with tanh



2 hidden layers with tanh

```
ıidden layers with tanh - Test loss: 0.1422, Test accuracy: 0.9563
```



```
2 hidden layers with sigmoid - Test loss: 0.1095, Test accuracy: 0.9652
```