

Spring 2024:CS5720 Neural Networks & Deep Learning

ICP-6

Name: Afroz Mohammad

Student ID:700758012

GitHub link: <https://github.com/Afrozmohammad19/Assignment6>

Video Link:

<https://drive.google.com/file/d/15k4AWxVYfUiT3uTNR7kiO0X9gJK-MFOg/view?usp=sharing>

Use Case Description:

Predicting the diabetes disease

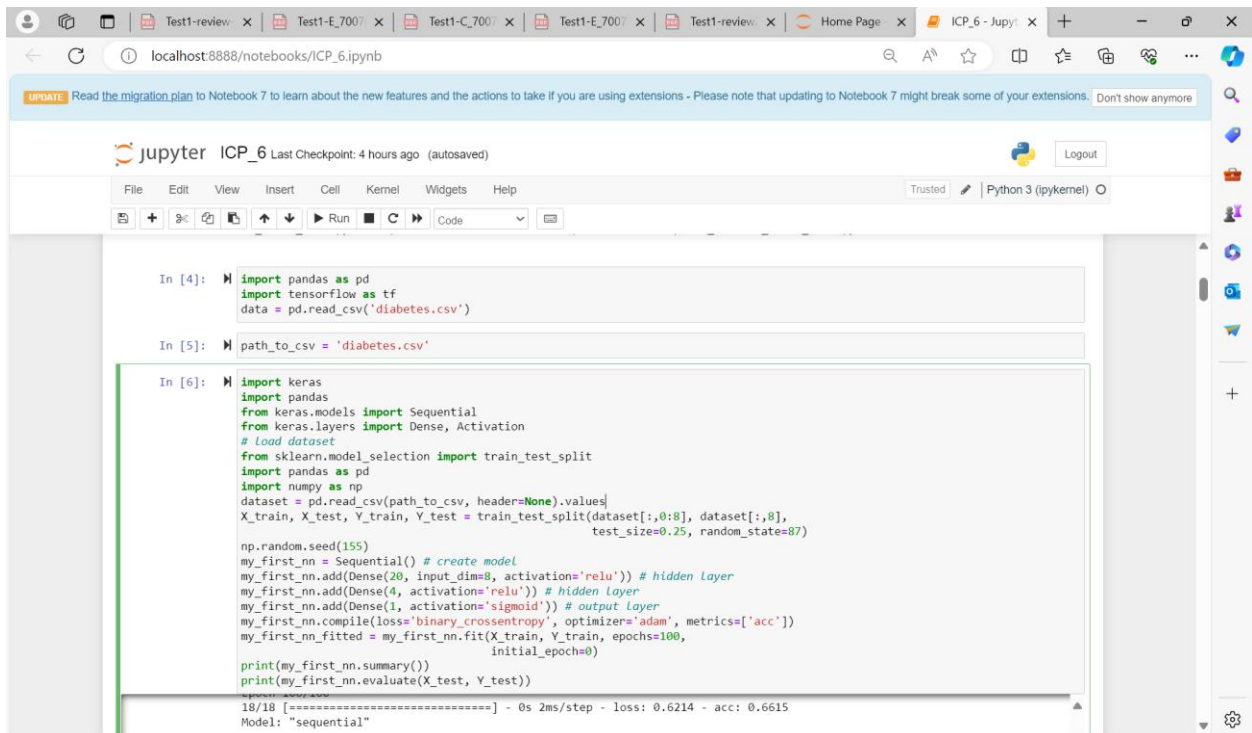
Programming elements

: Keras Basics

In class programming:

1. Use the use case in the class:

a. Add more Dense layers to the existing code and check how the accuracy changes.



The screenshot shows a Jupyter Notebook titled "ICP_6" with a last checkpoint 4 hours ago. The notebook is running on a local host at localhost:8888. The code is as follows:

```
In [4]: import pandas as pd
import tensorflow as tf
data = pd.read_csv('diabetes.csv')
```

```
In [5]: path_to_csv = 'diabetes.csv'
```

```
In [6]: import keras
import pandas
from keras.models import Sequential
from keras.layers 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, activations='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))
```

The output of the code is:

```
18/18 [=====] - 0s 2ms/step - loss: 0.6214 - acc: 0.6615
Model: "sequential"
```

Output:

The screenshot shows a Jupyter Notebook titled 'ICP_6' with a 'Python 3 (pykernel)' environment. The notebook contains several code cells. The first cell shows the output of `print(my_first_nn.evaluate(X_test, Y_test))`, which displays training progress for epochs 25 to 34. The second cell shows `data = pd.read_csv('breastcancer.csv')`. The third cell shows `path_to_csv = 'sample_data/breastcancer.csv'`. The fourth cell shows imports for `keras`, `pandas`, `numpy`, and `Sequential` from `keras.models`, and `Dense`, `Activation` from `keras.layers`. The fifth cell shows the output of `model.compile(loss='categorical_crossentropy', metrics=['accuracy'])`, which displays the model summary and compilation details.

```
Epoch 25/100
18/18 [=====] - 0s 3ms/step - loss: 0.6608 - acc: 0.6615
Epoch 26/100
18/18 [=====] - 0s 3ms/step - loss: 0.6601 - acc: 0.6615
Epoch 27/100
18/18 [=====] - 0s 4ms/step - loss: 0.6597 - acc: 0.6615
Epoch 28/100
18/18 [=====] - 0s 3ms/step - loss: 0.6579 - acc: 0.6615
Epoch 29/100
18/18 [=====] - 0s 3ms/step - loss: 0.6559 - acc: 0.6615
Epoch 30/100
18/18 [=====] - 0s 5ms/step - loss: 0.6546 - acc: 0.6615
Epoch 31/100
18/18 [=====] - 0s 4ms/step - loss: 0.6533 - acc: 0.6615
Epoch 32/100
18/18 [=====] - 0s 3ms/step - loss: 0.6537 - acc: 0.6615
Epoch 33/100
18/18 [=====] - 0s 3ms/step - loss: 0.6511 - acc: 0.6615
Epoch 34/100
18/18 [=====] - 0s 3ms/step - loss: 0.6511 - acc: 0.6615

In [ ]: data = pd.read_csv('breastcancer.csv')

In [7]: path_to_csv = 'sample_data/breastcancer.csv'

In [8]: import keras
import pandas as pd
import numpy as np
from keras.models import Sequential
from keras.layers import Dense, Activation

18/18 [=====] - 0s 2ms/step - loss: 0.6214 - acc: 0.6615
Model: "sequential"

Layer (type)                 Output Shape         Param #
-----
dense (Dense)                 (None, 20)           180
dense_1 (Dense)               (None, 4)            84
dense_2 (Dense)               (None, 1)            5

Total params: 269 (1.05 KB)
Trainable params: 269 (1.05 KB)
Non-trainable params: 0 (0.00 Byte)

None
6/6 [=====] - 0s 6ms/step - loss: 0.6564 - acc: 0.6198
[0.656437873840332, 0.6197916865348816]
```

2. Change the data source to Breast Cancer dataset * available in the source code folder and make required changes. Report accuracy of the model.

The screenshot shows a Jupyter Notebook titled 'ICP_6' with a last checkpoint 4 hours ago. The code in the notebook is as follows:

```
In [ ]: data = pd.read_csv('breastcancer.csv')

In [7]: path_to_csv = 'sample_data/breastcancer.csv'

In [8]: import keras
import pandas as pd
import numpy as np
from keras.models import Sequential
from keras.layers 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))
```

The output of the notebook shows the progress of the training process, with the following visible output:

```
Epoch 15/100
14/14 [=====] - 0s 4ms/step - loss: 0.5710 - acc: 0.8901
```

Output:

The screenshot shows the output of the Jupyter Notebook, displaying the training progress of the neural network. The output is as follows:

```
14/14 [=====] - 0s 6ms/step - loss: 0.1966 - acc: 0.9296
Epoch 63/100
14/14 [=====] - 0s 5ms/step - loss: 0.2044 - acc: 0.9272
Epoch 64/100
14/14 [=====] - 0s 4ms/step - loss: 0.1965 - acc: 0.9366
Epoch 65/100
14/14 [=====] - 0s 2ms/step - loss: 0.2597 - acc: 0.9249
Epoch 66/100
14/14 [=====] - 0s 4ms/step - loss: 0.2477 - acc: 0.9249
Epoch 67/100
14/14 [=====] - 0s 4ms/step - loss: 0.2031 - acc: 0.9366
Epoch 68/100
14/14 [=====] - 0s 4ms/step - loss: 0.2834 - acc: 0.9155
Epoch 69/100
14/14 [=====] - 0s 2ms/step - loss: 0.3308 - acc: 0.9155
Epoch 70/100
14/14 [=====] - 0s 4ms/step - loss: 0.2201 - acc: 0.9343
Epoch 71/100
14/14 [=====] - 0s 4ms/step - loss: 0.1945 - acc: 0.9319
Epoch 72/100
```

```
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 2ms/step - loss: 0.1506 - acc: 0.9413
Epoch 100/100
14/14 [=====] - 0s 4ms/step - loss: 0.1439 - acc: 0.9390
Model: "sequential_1"
Layer (type)                 Output Shape              Param #
-----
dense_3 (Dense)              (None, 20)                620
dense_4 (Dense)              (None, 1)                 21
Total params: 641 (2.50 KB)
Trainable params: 641 (2.50 KB)
Non-trainable params: 0 (0.00 Byte)
None
5/5 [=====] - 0s 8ms/step - loss: 0.1963 - acc: 0.9510
[0.19632473587989807, 0.9510489702224731]
```

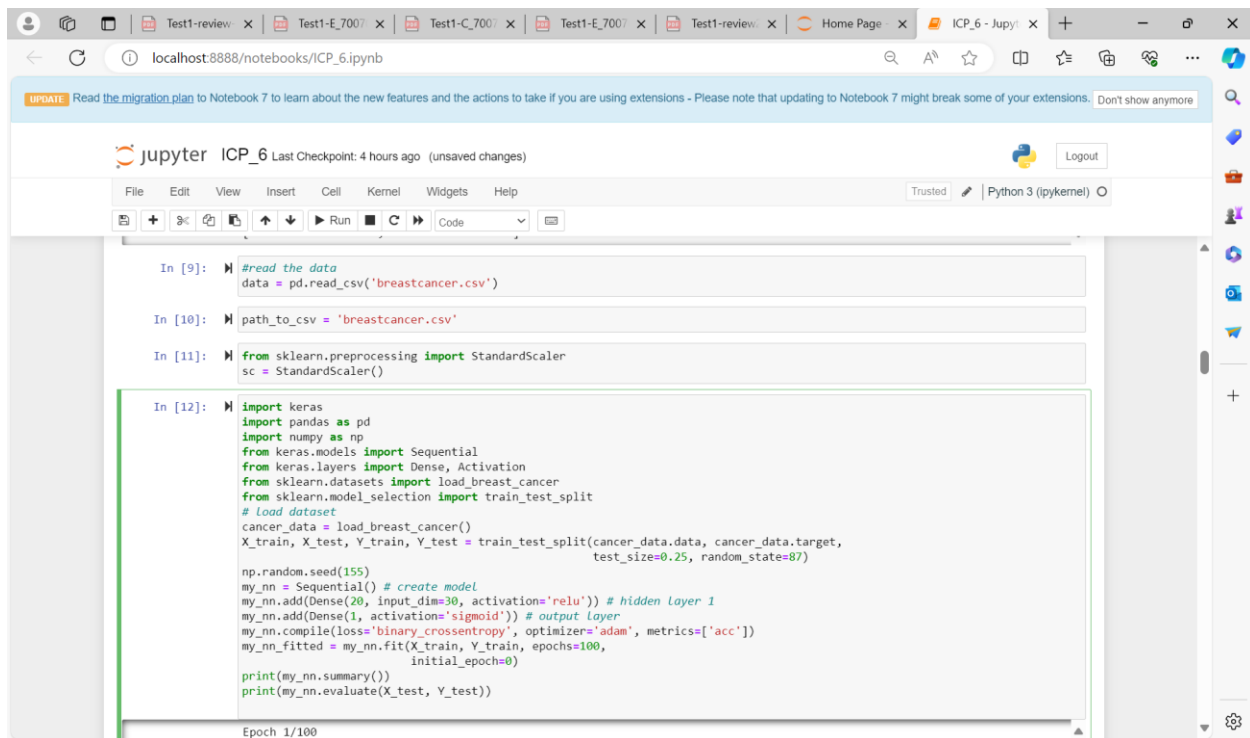
```
In [9]: #read the data
data = pd.read_csv('breastcancer.csv')
```

3. Normalize the data before feeding the data to the model and check how the normalization change your accuracy (code given below).

```
from sklearn.preprocessing import StandardScaler
```

```
sc = StandardScaler()
```

Breast Cancer dataset is designated to predict if a patient has Malignant (M) or Benign = B cancer



The screenshot shows a Jupyter Notebook interface with the following code cells:

```
In [9]: #read the data
data = pd.read_csv('breastcancer.csv')

In [10]: path_to_csv = 'breastcancer.csv'

In [11]: from sklearn.preprocessing import StandardScaler
sc = StandardScaler()

In [12]: import keras
import pandas as pd
import numpy as np
from keras.models import Sequential
from keras.layers import Dense, Activation
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split

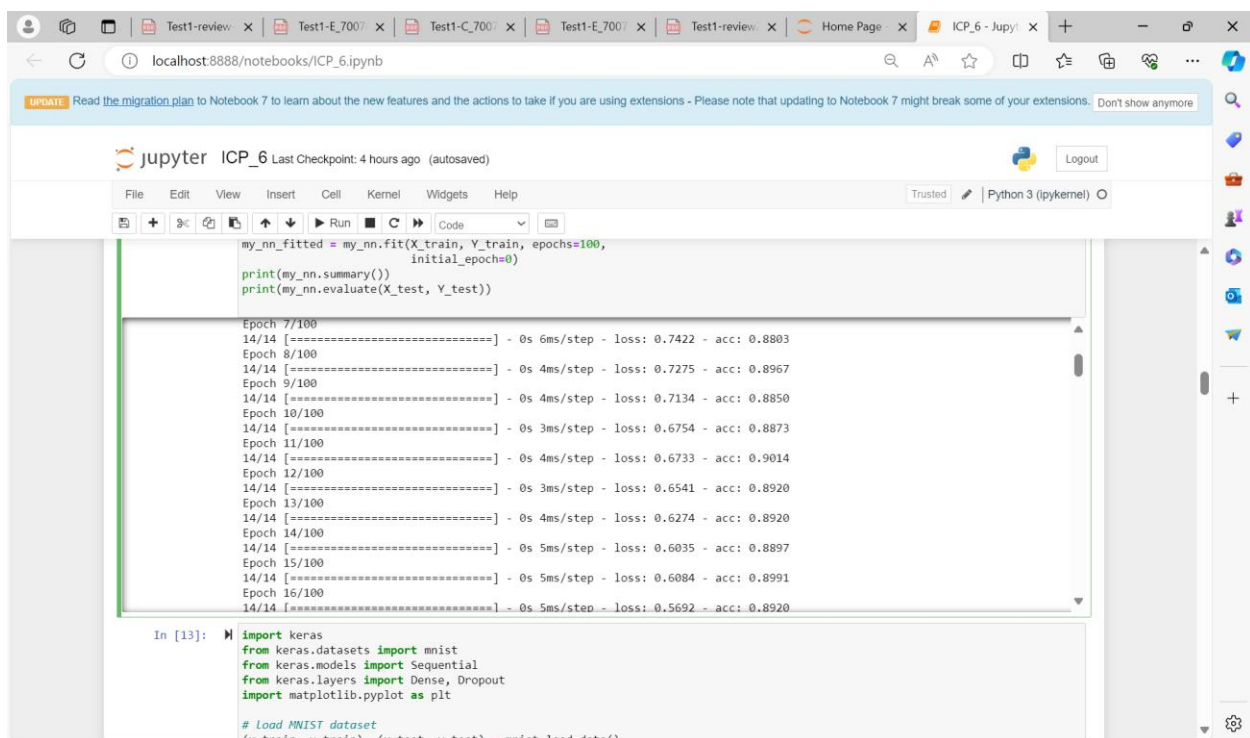
# Load dataset
cancer_data = load_breast_cancer()
X_train, X_test, Y_train, Y_test = train_test_split(cancer_data.data, cancer_data.target,
                                                    test_size=0.25, random_state=87)

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))
```

Epoch 1/100

Output:



The screenshot shows the same Jupyter Notebook interface, but with the output of the previous cell visible. The output displays the training progress for 100 epochs, showing loss and accuracy metrics. The final output shows the model summary and evaluation results.

```
Epoch 7/100
14/14 [=====] - 0s 6ms/step - loss: 0.7422 - acc: 0.8803
Epoch 8/100
14/14 [=====] - 0s 4ms/step - loss: 0.7275 - acc: 0.8967
Epoch 9/100
14/14 [=====] - 0s 4ms/step - loss: 0.7134 - acc: 0.8850
Epoch 10/100
14/14 [=====] - 0s 3ms/step - loss: 0.6754 - acc: 0.8873
Epoch 11/100
14/14 [=====] - 0s 4ms/step - loss: 0.6733 - acc: 0.9014
Epoch 12/100
14/14 [=====] - 0s 3ms/step - loss: 0.6541 - acc: 0.8920
Epoch 13/100
14/14 [=====] - 0s 4ms/step - loss: 0.6274 - acc: 0.8920
Epoch 14/100
14/14 [=====] - 0s 5ms/step - loss: 0.6035 - acc: 0.8897
Epoch 15/100
14/14 [=====] - 0s 5ms/step - loss: 0.6084 - acc: 0.8991
Epoch 16/100
14/14 [=====] - 0s 5ms/step - loss: 0.5692 - acc: 0.8920

In [13]: 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()
```

```
14/14 [=====] - 0s 3ms/step - loss: 0.1371 - acc: 0.9413
Epoch 100/100
14/14 [=====] - 0s 3ms/step - loss: 0.1535 - acc: 0.9343
Model: "sequential_2"

Layer (type)                 Output Shape              Param #
-----
dense_5 (Dense)              (None, 20)                620
dense_6 (Dense)              (None, 1)                 21
-----
Total params: 641 (2.50 KB)
Trainable params: 641 (2.50 KB)
Non-trainable params: 0 (0.00 Byte)

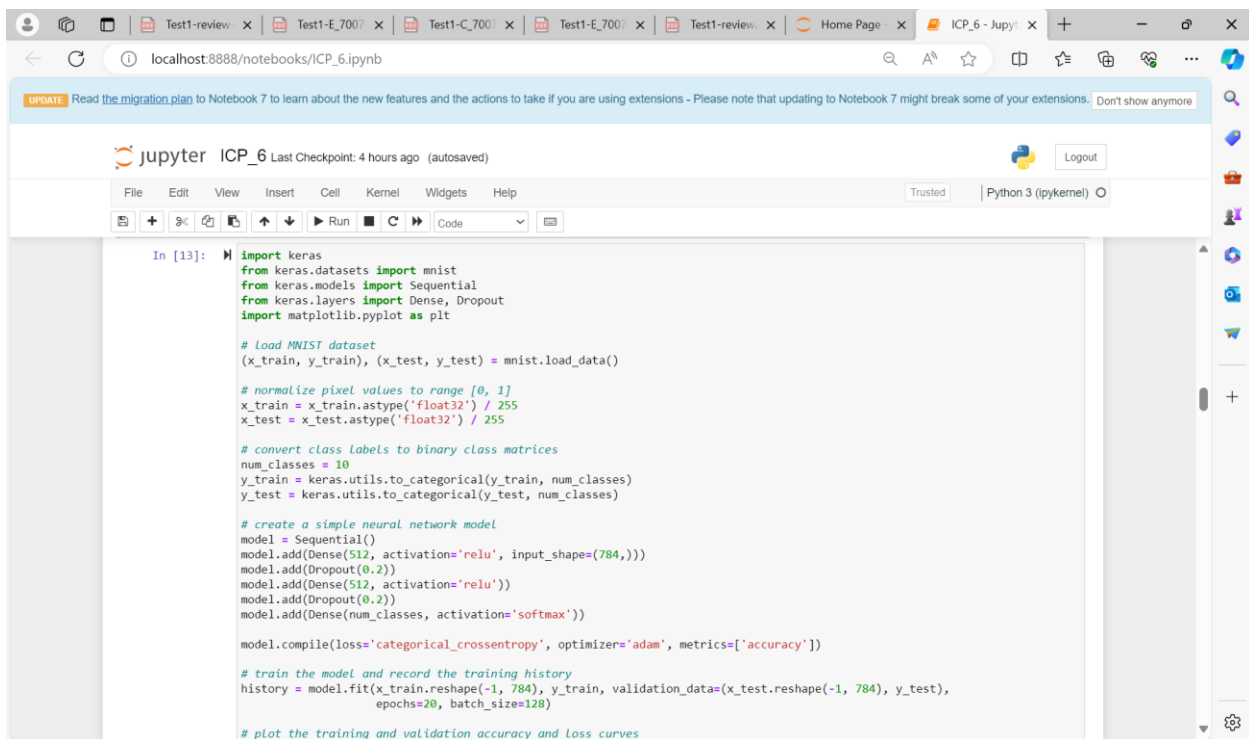
None
5/5 [=====] - 0s 8ms/step - loss: 0.3068 - acc: 0.9091
[0.306810587644577, 0.9090909361839294]
```

In [13]: `import keras`

In class programming:

Use Image Classification on the hand written digits data set (mnist)

1. Plot the loss and accuracy for both training data and validation data using the history object in the source code.



The screenshot shows a Jupyter Notebook titled "ICP_6" with a "Python 3 (ipykernel)" environment. The code cell contains the following Python code:

```
In [13]: 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

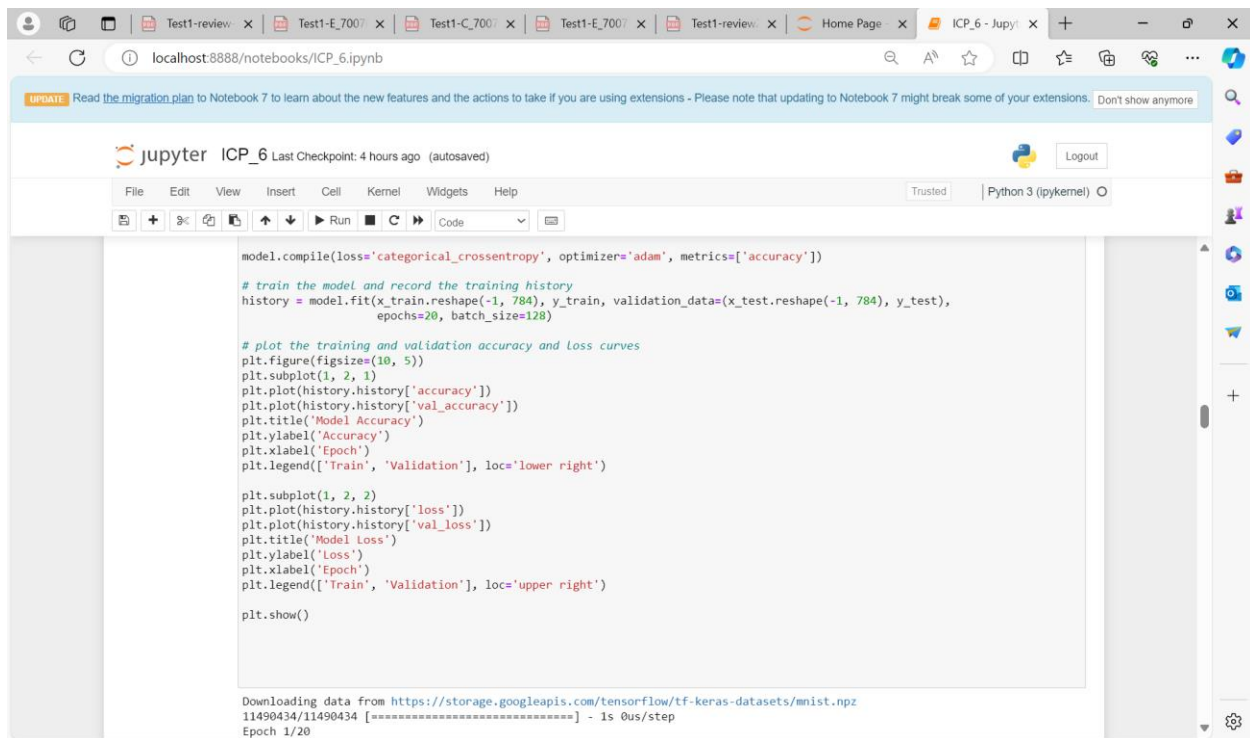
# convert class labels to binary class matrices
num_classes = 10
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)

# 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
```



The screenshot shows a Jupyter Notebook titled "ICP_6" with a "Python 3 (ipykernel)" environment. The code in the cell defines a model, trains it for 20 epochs, and plots the training and validation accuracy and loss curves. The code is as follows:

```
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')

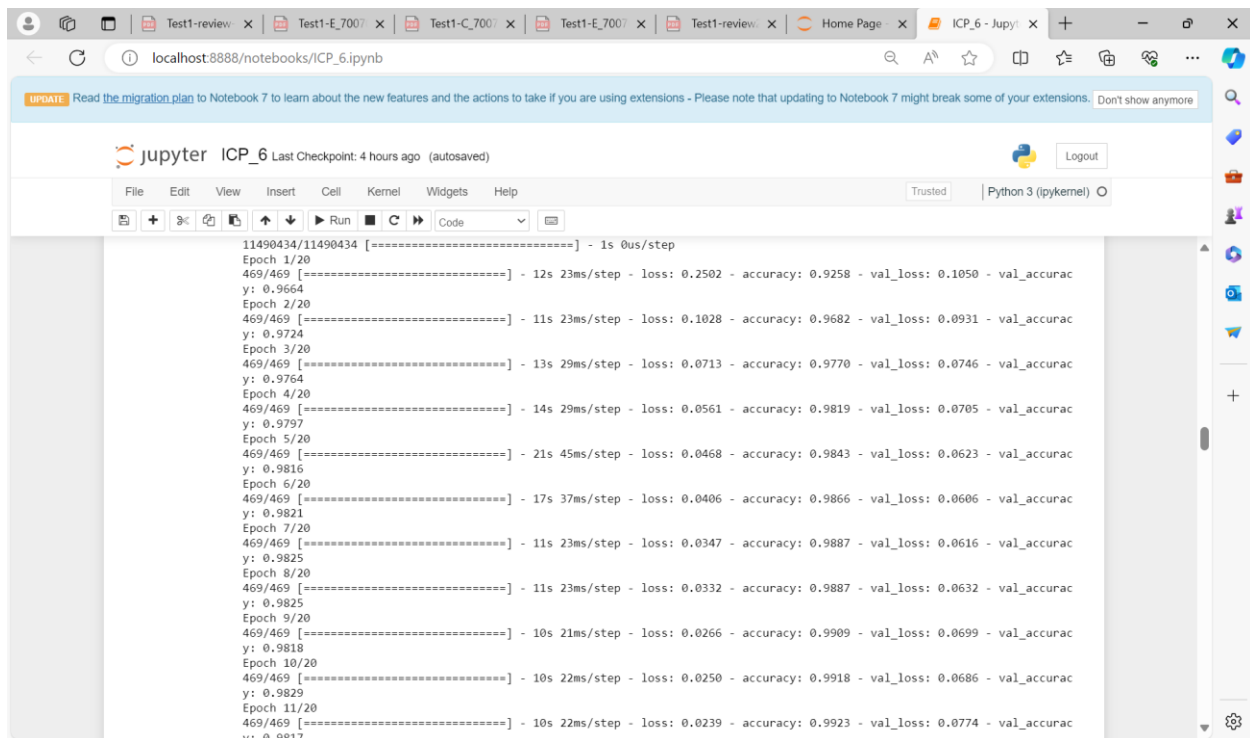
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper right')

plt.show()
```

Below the code, the output shows the progress of the training process:

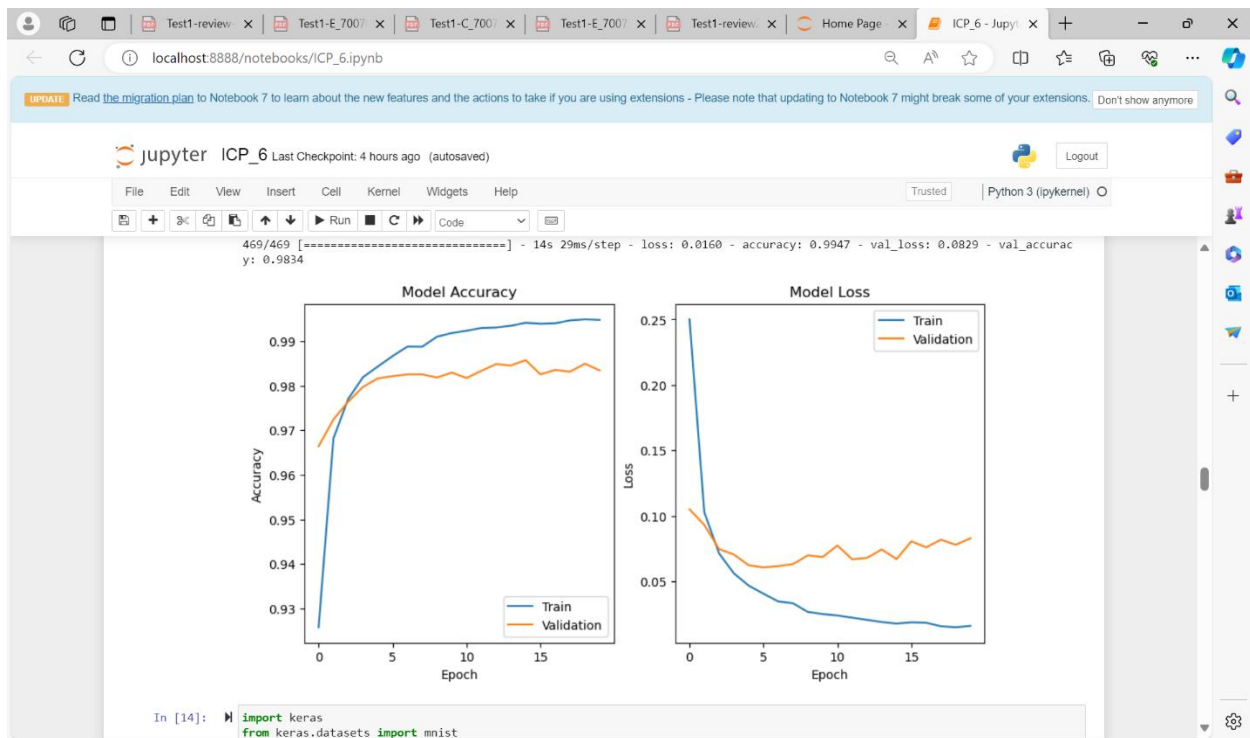
```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 [=====] - 1s 0us/step
Epoch 1/20
```

Output:



The screenshot shows the same Jupyter Notebook interface, but now displaying the output of the training process. The output shows the progress of the training process, including the epoch number, the time taken for each step, the loss, and the accuracy. The output is as follows:

```
11490434/11490434 [=====] - 1s 0us/step
Epoch 1/20
469/469 [=====] - 12s 23ms/step - loss: 0.2502 - accuracy: 0.9258 - val_loss: 0.1050 - val_accuac
y: 0.9664
Epoch 2/20
469/469 [=====] - 11s 23ms/step - loss: 0.1028 - accuracy: 0.9682 - val_loss: 0.0931 - val_accuac
y: 0.9724
Epoch 3/20
469/469 [=====] - 13s 29ms/step - loss: 0.0713 - accuracy: 0.9770 - val_loss: 0.0746 - val_accuac
y: 0.9764
Epoch 4/20
469/469 [=====] - 14s 29ms/step - loss: 0.0561 - accuracy: 0.9819 - val_loss: 0.0705 - val_accuac
y: 0.9797
Epoch 5/20
469/469 [=====] - 21s 45ms/step - loss: 0.0468 - accuracy: 0.9843 - val_loss: 0.0623 - val_accuac
y: 0.9816
Epoch 6/20
469/469 [=====] - 17s 37ms/step - loss: 0.0406 - accuracy: 0.9866 - val_loss: 0.0606 - val_accuac
y: 0.9821
Epoch 7/20
469/469 [=====] - 11s 23ms/step - loss: 0.0347 - accuracy: 0.9887 - val_loss: 0.0616 - val_accuac
y: 0.9825
Epoch 8/20
469/469 [=====] - 11s 23ms/step - loss: 0.0332 - accuracy: 0.9887 - val_loss: 0.0632 - val_accuac
y: 0.9825
Epoch 9/20
469/469 [=====] - 10s 21ms/step - loss: 0.0266 - accuracy: 0.9909 - val_loss: 0.0699 - val_accuac
y: 0.9818
Epoch 10/20
469/469 [=====] - 10s 22ms/step - loss: 0.0250 - accuracy: 0.9918 - val_loss: 0.0686 - val_accuac
y: 0.9829
Epoch 11/20
469/469 [=====] - 10s 22ms/step - loss: 0.0239 - accuracy: 0.9923 - val_loss: 0.0774 - val_accuac
y: 0.9817
```

2. Plot one of the images in the test data, and then do inferencing to check what is the prediction of the model on that single image.

Output:

localhost:8888/notebooks/ICP_6.ipynb

UPDATE Read the [migration plan](#) to Notebook 7 to learn about the new features and the actions to take if you are using extensions - Please note that updating to Notebook 7 might break some of your extensions. Don't show anymore

jupyter ICP_6 Last Checkpoint: 4 hours ago (autosaved) Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel)

```
Epoch 1/20
469/469 [=====] - 13s 25ms/step - loss: 0.2441 - accuracy: 0.9264 - val_loss: 0.1092 - val_accu-
racy: 0.9664
Epoch 2/20
469/469 [=====] - 11s 23ms/step - loss: 0.1017 - accuracy: 0.9680 - val_loss: 0.0915 - val_accu-
racy: 0.9708
Epoch 3/20
469/469 [=====] - 10s 22ms/step - loss: 0.0722 - accuracy: 0.9773 - val_loss: 0.0716 - val_accu-
racy: 0.9780
Epoch 4/20
469/469 [=====] - 10s 21ms/step - loss: 0.0573 - accuracy: 0.9818 - val_loss: 0.0654 - val_accu-
racy: 0.9807
Epoch 5/20
469/469 [=====] - 15s 32ms/step - loss: 0.0457 - accuracy: 0.9851 - val_loss: 0.0648 - val_accu-
racy: 0.9806
Epoch 6/20
469/469 [=====] - 10s 22ms/step - loss: 0.0384 - accuracy: 0.9871 - val_loss: 0.0674 - val_accu-
racy: 0.9804
Epoch 7/20
469/469 [=====] - 11s 23ms/step - loss: 0.0360 - accuracy: 0.9881 - val_loss: 0.0633 - val_accu-
racy: 0.9819
Epoch 8/20
469/469 [=====] - 10s 20ms/step - loss: 0.0302 - accuracy: 0.9903 - val_loss: 0.0678 - val_accu-
racy: 0.9822
Epoch 9/20
469/469 [=====] - 11s 23ms/step - loss: 0.0268 - accuracy: 0.9912 - val_loss: 0.0857 - val_accu-
racy: 0.9777
Epoch 10/20
469/469 [=====] - 10s 22ms/step - loss: 0.0257 - accuracy: 0.9916 - val_loss: 0.0682 - val_accu-
racy: 0.9847
Epoch 11/20
469/469 [=====] - 11s 23ms/step - loss: 0.0230 - accuracy: 0.9924 - val_loss: 0.0751 - val_accu-
```

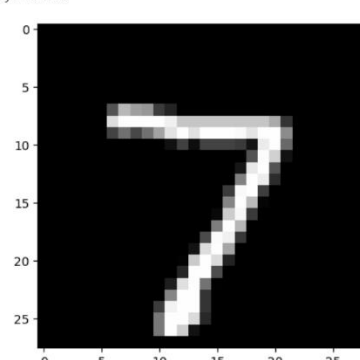
localhost:8888/notebooks/ICP_6.ipynb

UPDATE Read the [migration plan](#) to Notebook 7 to learn about the new features and the actions to take if you are using extensions - Please note that updating to Notebook 7 might break some of your extensions. Don't show anymore

jupyter ICP_6 Last Checkpoint: 4 hours ago (autosaved) Logout

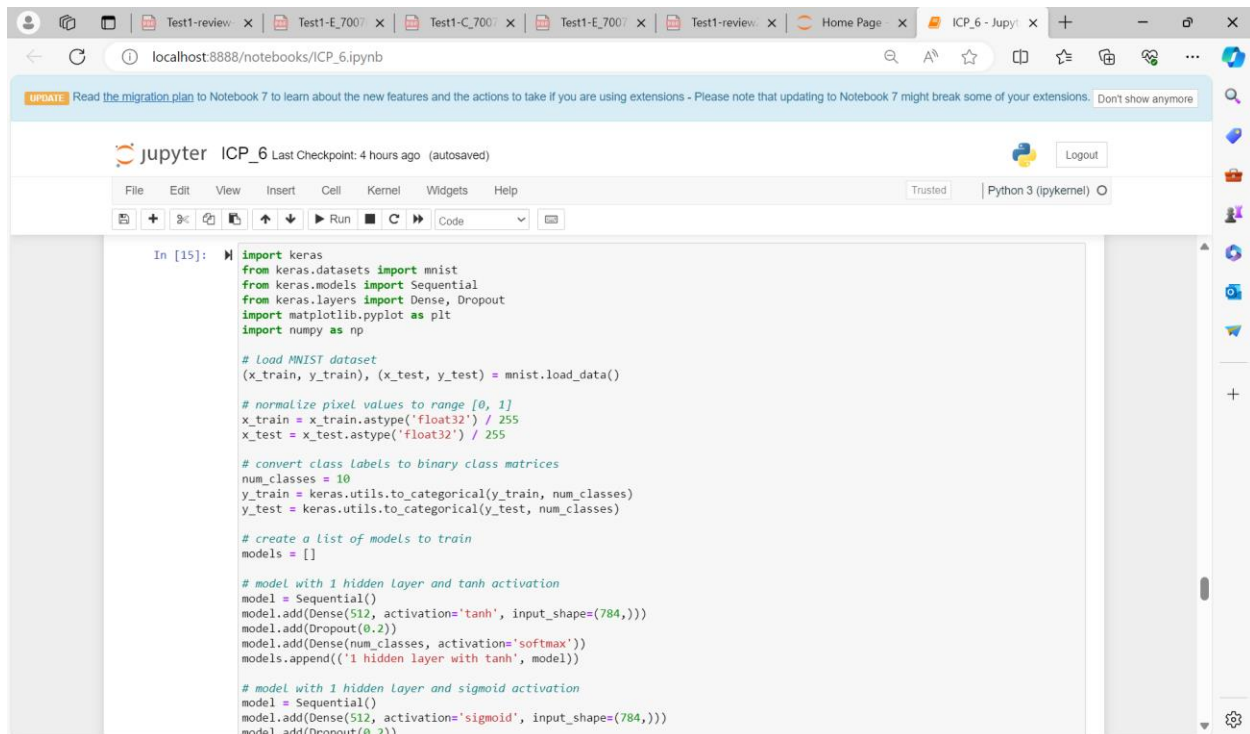
File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel)

```
Epoch 19/20
469/469 [=====] - 10s 22ms/step - loss: 0.0150 - accuracy: 0.9950 - val_loss: 0.0902 - val_accu-
racy: 0.9821
Epoch 20/20
469/469 [=====] - 10s 21ms/step - loss: 0.0153 - accuracy: 0.9949 - val_loss: 0.0814 - val_accu-
racy: 0.9839
```



```
1/1 [=====] - 0s 202ms/step
Model prediction: 7
```

3. We had used 2 hidden layers and Relu activation. Try to change the number of hidden layer and the activation to tanh or sigmoid and see what happens.



The screenshot shows a Jupyter Notebook titled 'ICP_6' with a 'Python 3 (ipykernel)' kernel. The code in the cell is as follows:

```
In [15]: 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()

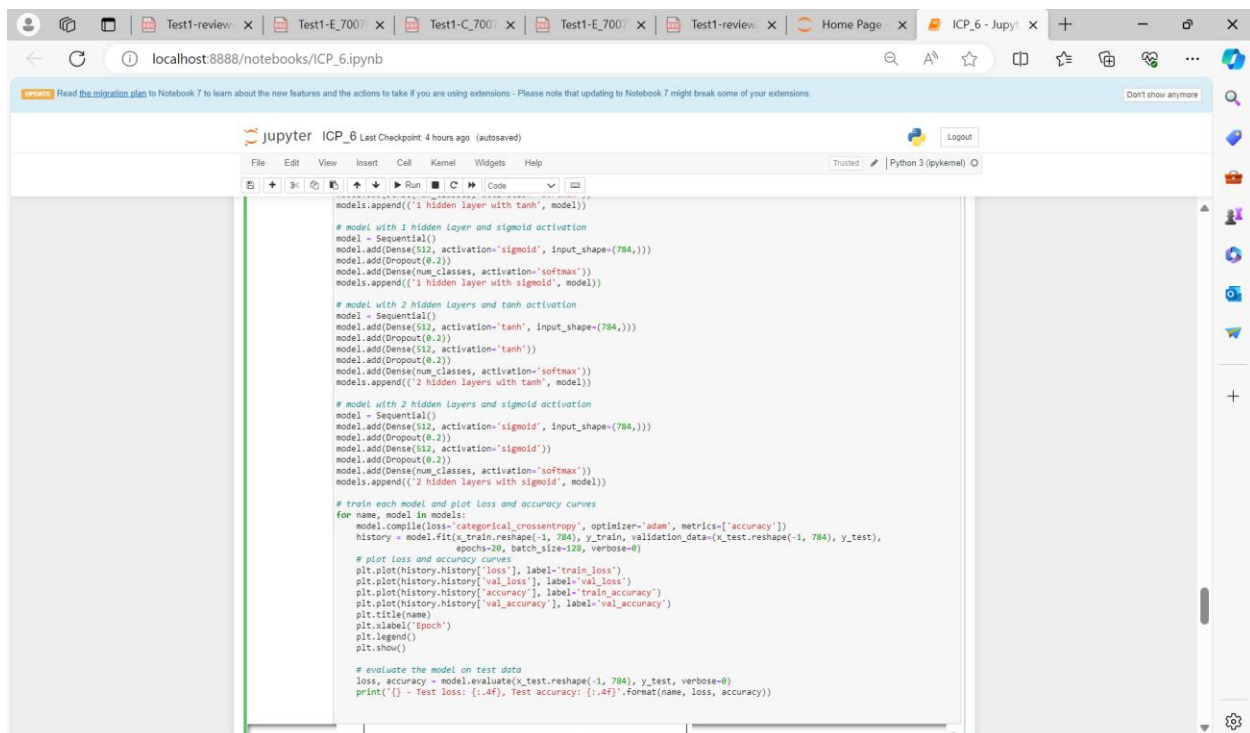
# normalize pixel values to range [0, 1]
x_train = x_train.astype('float32') / 255
x_test = x_test.astype('float32') / 255

# 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))
```



The screenshot shows the same Jupyter Notebook with the code extended to include training and plotting. The code is as follows:

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

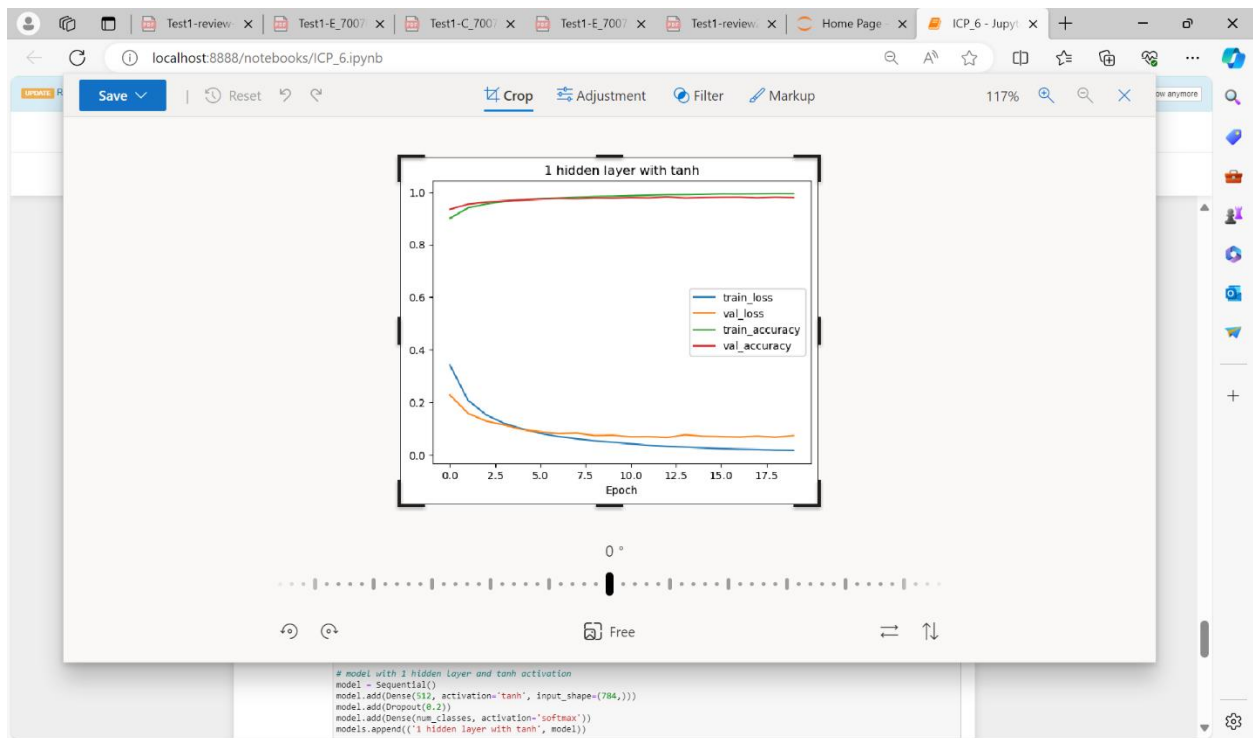
# model with 2 hidden layers and tanh activation
model = Sequential()
model.add(Dense(512, activation='tanh', input_shape=(784,)))
model.add(Dropout(0.2))
model.add(Dense(512, activation='tanh'))
model.add(Dropout(0.2))
model.add(Dense(num_classes, activation='softmax'))
models.append(('2 hidden layers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

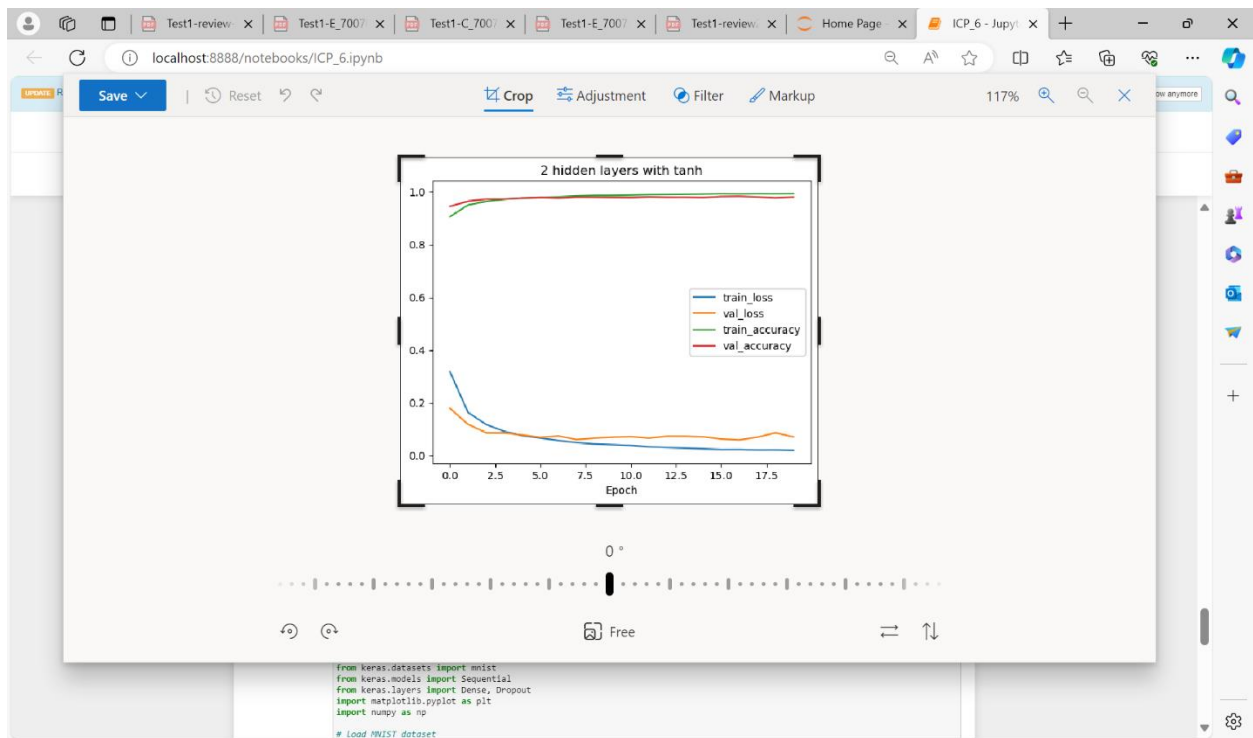
# evaluate the model on test data
loss, accuracy = model.evaluate(x_test.reshape(-1, 784), y_test, verbose=0)
print('{} - Test loss: {:.4f}, Test accuracy: {:.4f}'.format(name, loss, accuracy))
```



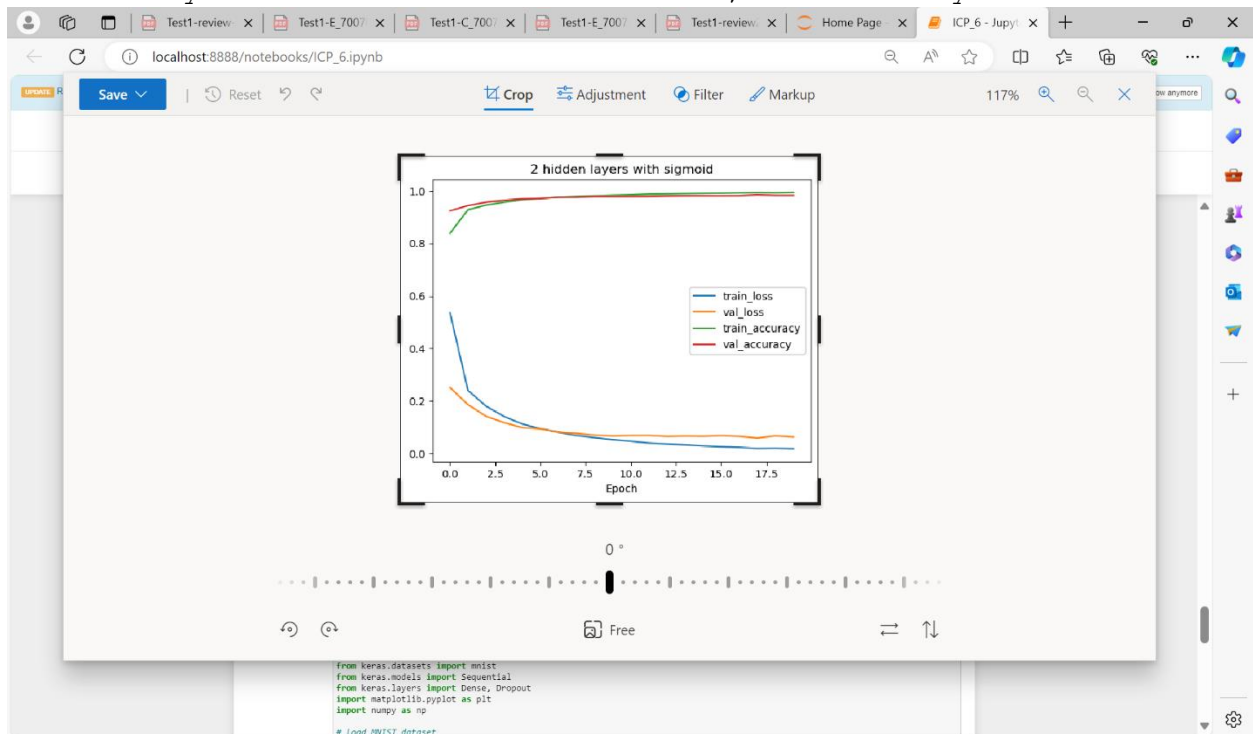
1 hidden layer with tanh - Test loss: 0.0738, Test accuracy: 0.9792



1 hidden layer with sigmoid - Test loss: 0.0578, Test accuracy: 0.9821

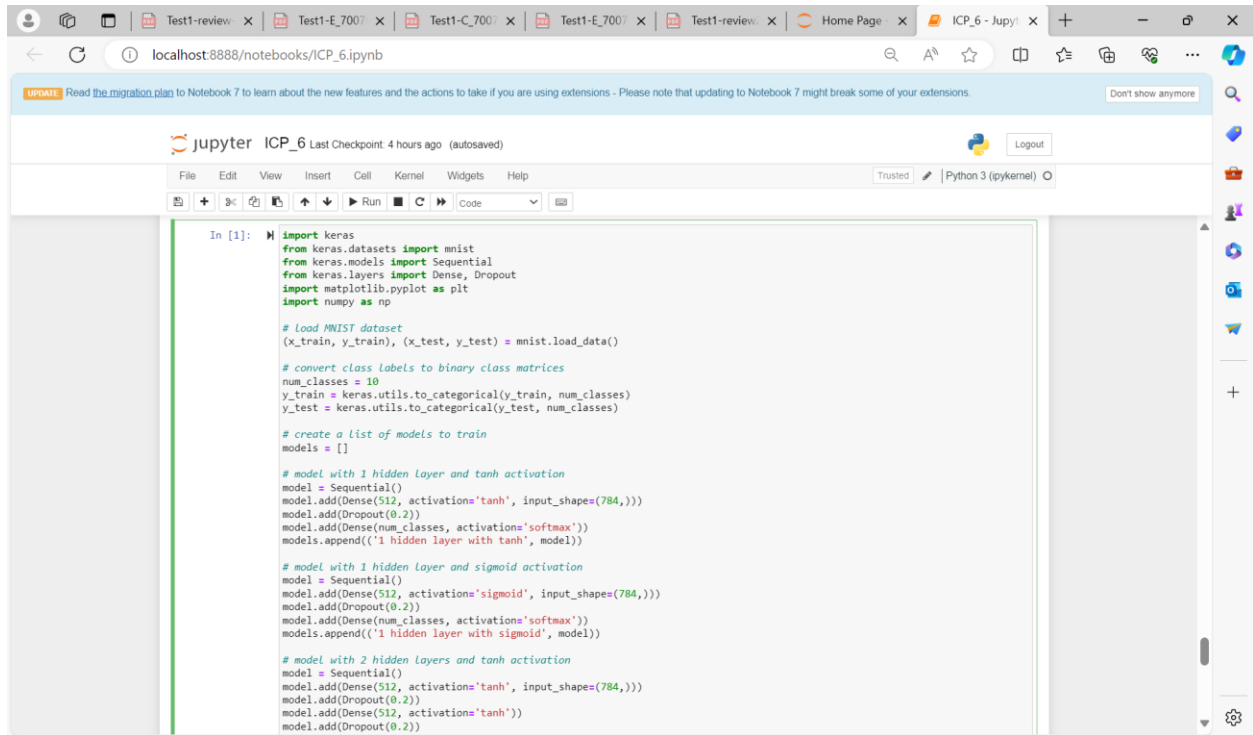


2 hidden layers with tanh - Test loss: 0.0711, Test accuracy: 0.9803



2 hidden layers with sigmoid - Test loss: 0.0629, Test accuracy: 0.9829

4. Run the same code without scaling the images and check the performance?



```
In [1]: 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()

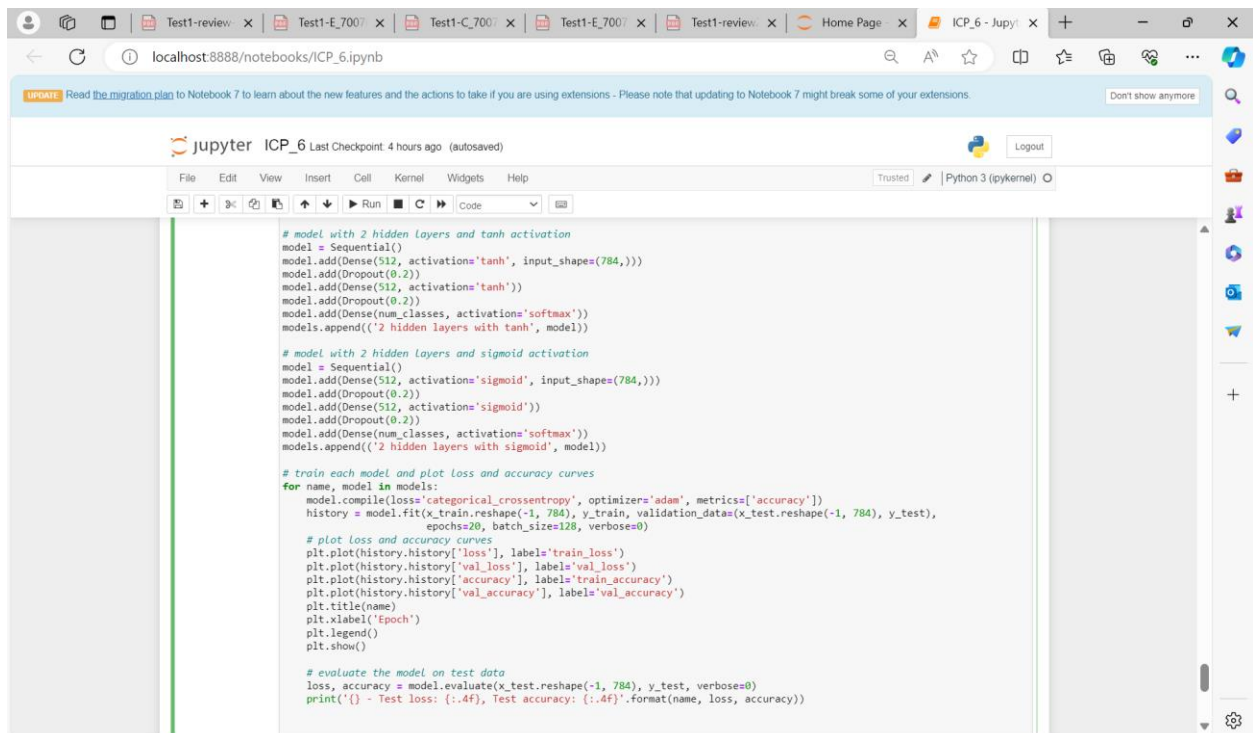
# convert class labels to binary class matrices
num_classes = 10
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)

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

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

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

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



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

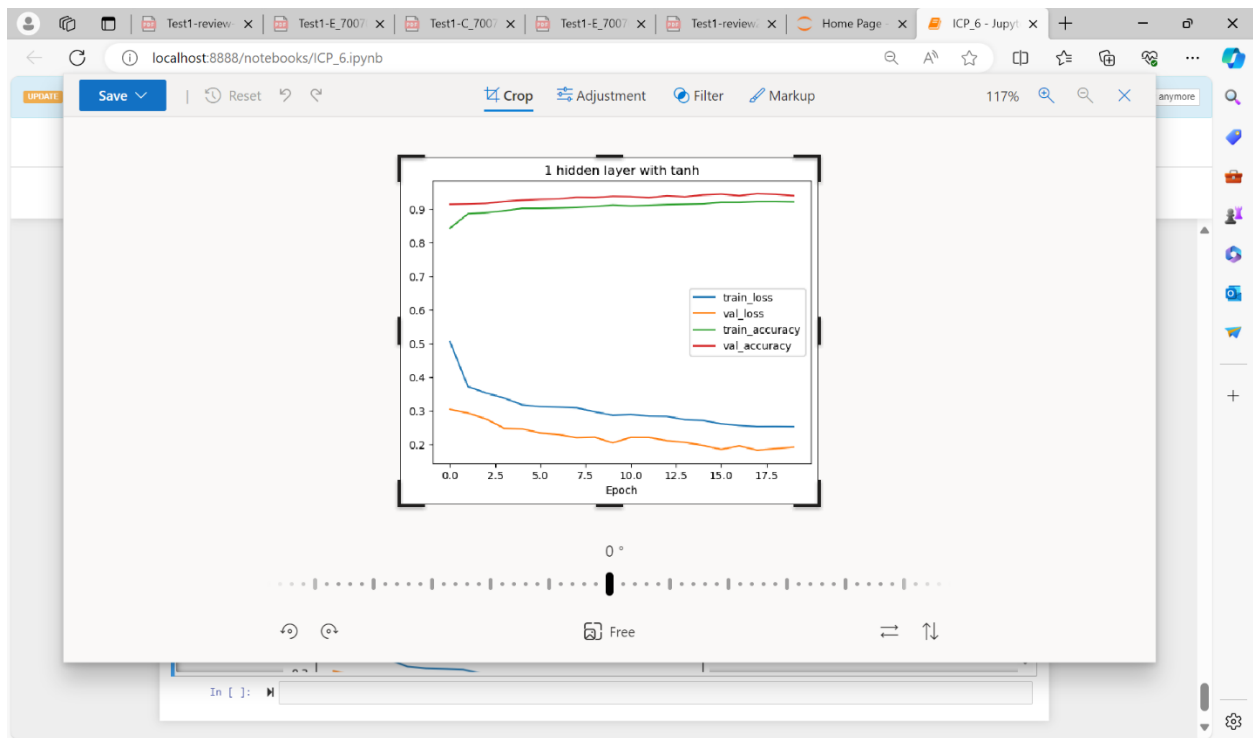
# model with 2 hidden layers and sigmoid activation
model = Sequential()
model.add(Dense(512, activations='sigmoid', input_shape=(784,)))
model.add(Dropout(0.2))
model.add(Dense(512, activations='sigmoid'))
model.add(Dropout(0.2))
model.add(Dense(num_classes, activations='softmax'))
models.append(('2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

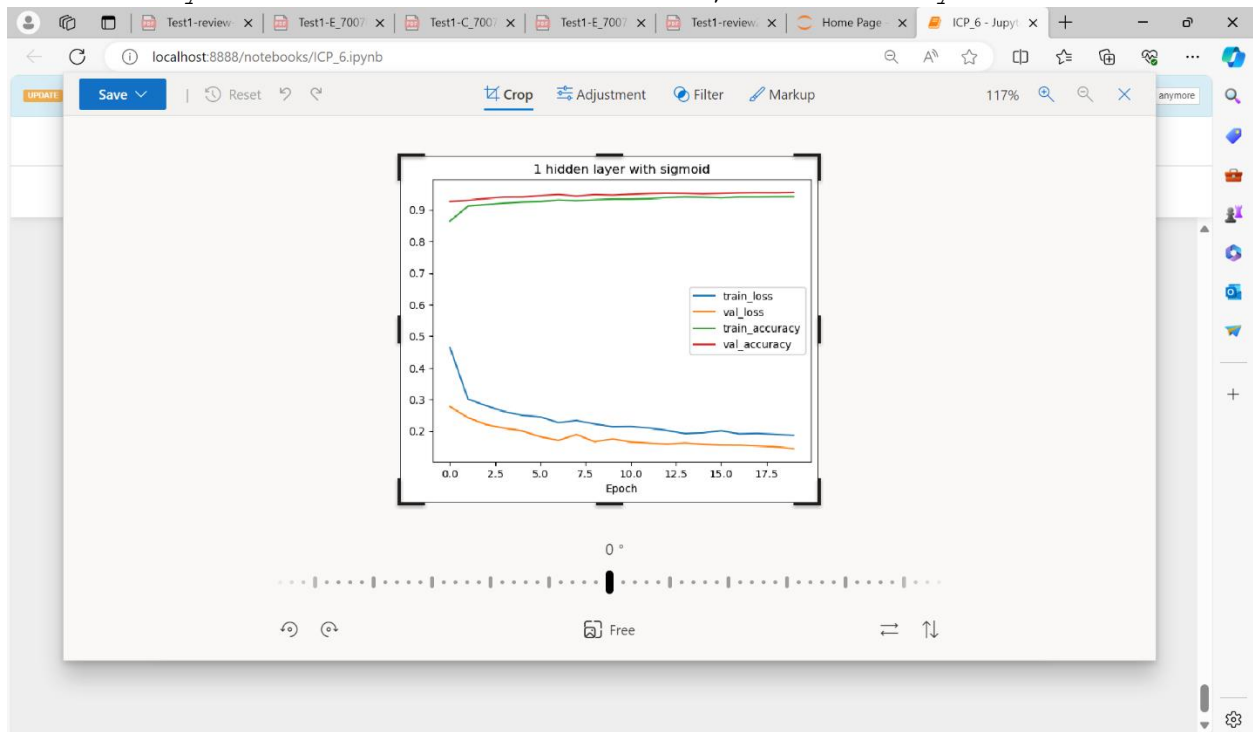
    # plot loss and accuracy curves
    plt.plot(history.history['loss'], label='train_loss')
    plt.plot(history.history['val_loss'], label='val_loss')
    plt.plot(history.history['accuracy'], label='train_accuracy')
    plt.plot(history.history['val_accuracy'], label='val_accuracy')
    plt.title(name)
    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))
```

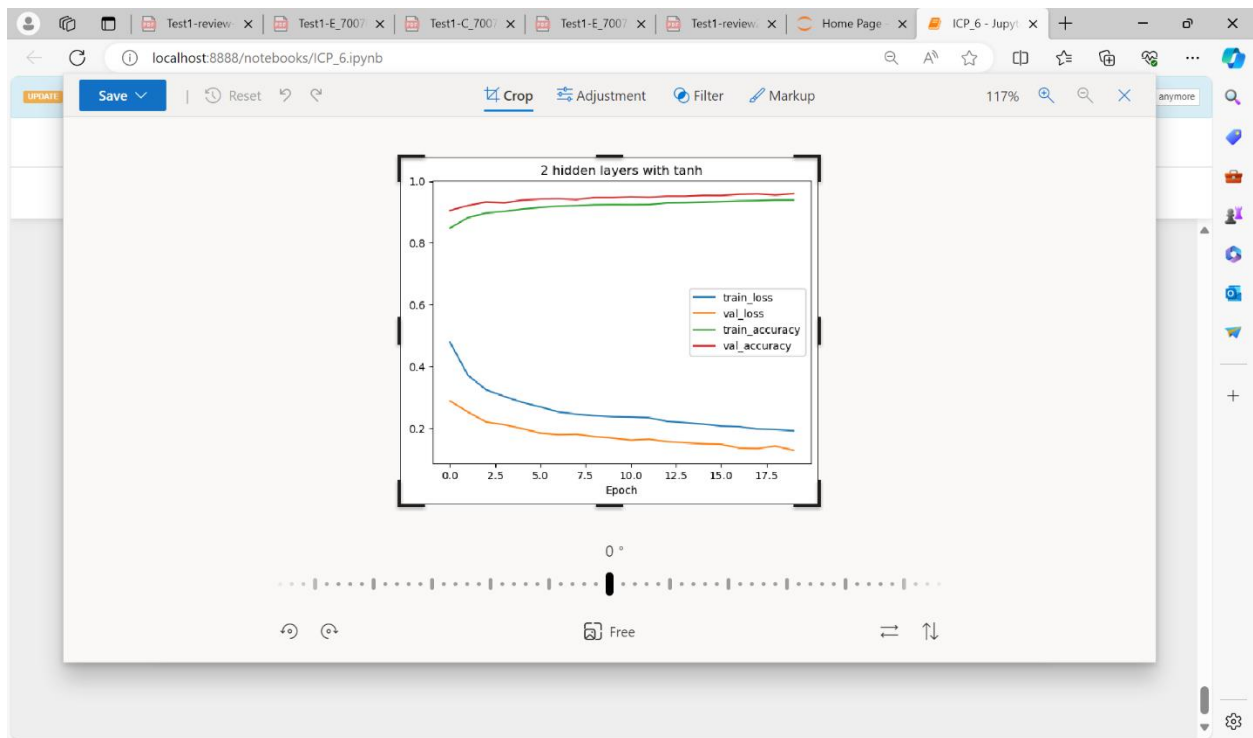
Output:



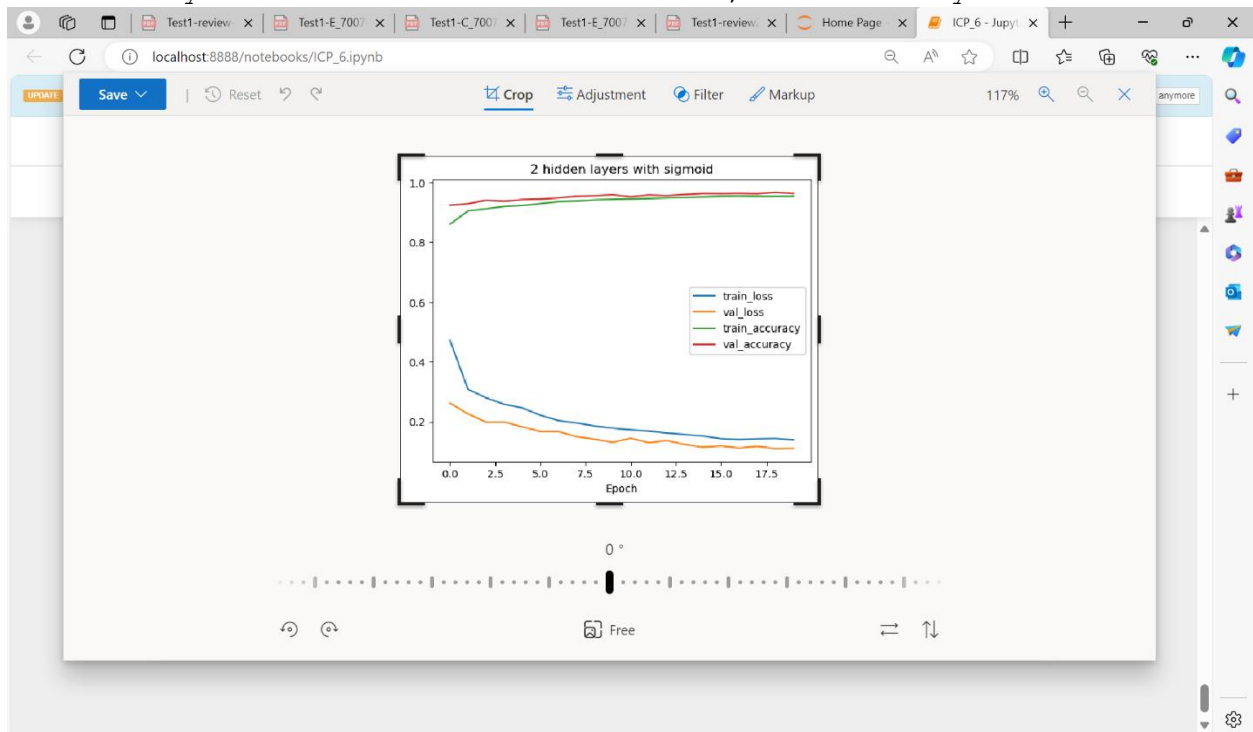
1 hidden layer with tanh - Test loss: 0.1912, Test accuracy: 0.9400



1 hidden layer with sigmoid - Test loss: 0.1437, Test accuracy: 0.9552



2 hidden layers with tanh - Test loss: 0.1293, Test accuracy: 0.9595



2 hidden layers with sigmoid - Test loss: 0.1119, Test accuracy: 0.9636

In []:

