

Spring 2024:CS5720 Neural Networks & Deep Learning

ICP-6 Assignment-6

Name: Sai Sushma Sri Bireddy Student ID:700747557

GitHub Link: <https://github.com/SaiSushmaSriBireddy/Assignment6>

Video Link:

<https://drive.google.com/file/d/1m7rKwtf9ErX0aXVZ1WGvI4ltyMQq26C5/view?usp=sharing>

Use Case Description:

Predicting the diabetes disease

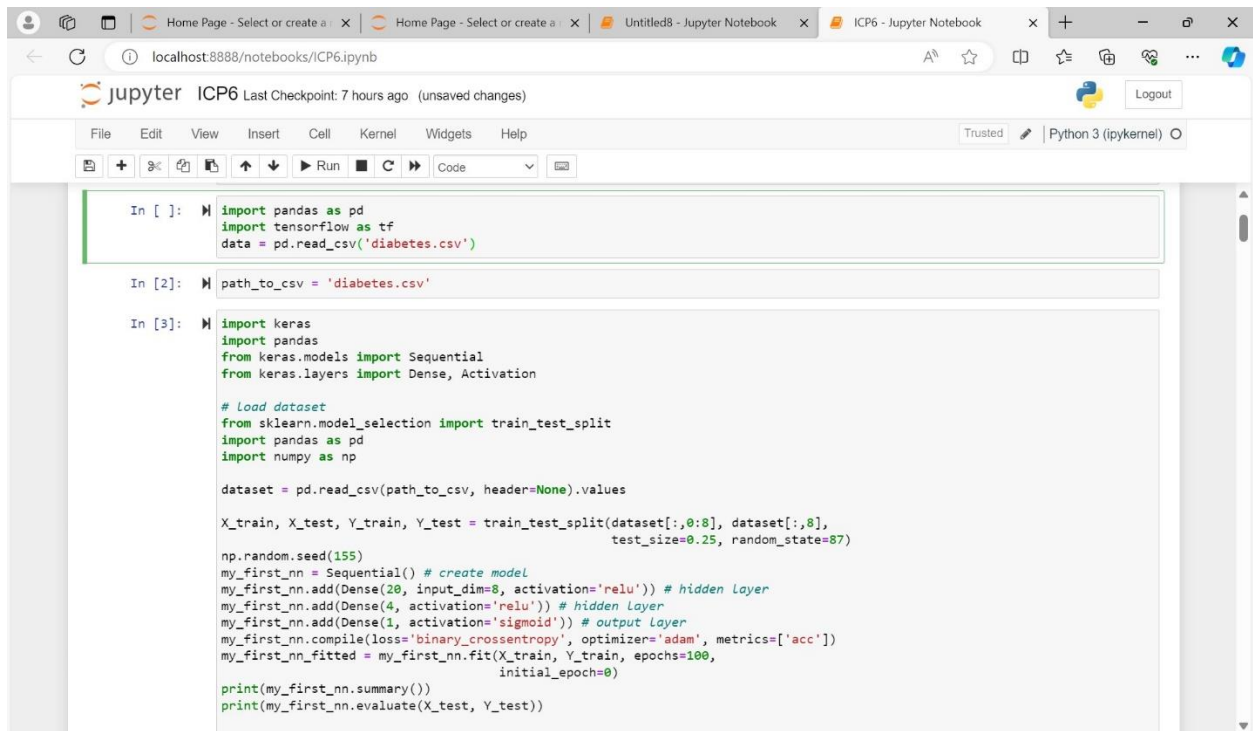
Programming elements:

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.



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

In [2]: path_to_csv = 'diabetes.csv'

In [3]: 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, 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))
```

Output:

```
Epoch 83/100 [=====] - 0s 7ms/step - loss: 0.6001 - acc: 0.7024
Epoch 84/100 [=====] - 0s 5ms/step - loss: 0.6001 - acc: 0.7083
Epoch 85/100 [=====] - 0s 5ms/step - loss: 0.5991 - acc: 0.7066
Epoch 86/100 [=====] - 0s 5ms/step - loss: 0.5987 - acc: 0.7049
Epoch 87/100 [=====] - 0s 5ms/step - loss: 0.6006 - acc: 0.7031
Epoch 88/100 [=====] - 0s 5ms/step - loss: 0.6041 - acc: 0.6979
Epoch 89/100 [=====] - 0s 6ms/step - loss: 0.6013 - acc: 0.6962
Epoch 90/100 [=====] - 0s 5ms/step - loss: 0.6007 - acc: 0.7031
Epoch 91/100 [=====] - 0s 5ms/step - loss: 0.5971 - acc: 0.7083
Epoch 92/100 [=====] - 0s 7ms/step - loss: 0.5972 - acc: 0.7066

In [4]: data = pd.read_csv('breastcancer.csv')

print(my_first_nn.summary())
print(my_first_nn.evaluate(X_test, Y_test))

18/18 [=====] - 0s 4ms/step - loss: 0.5963 - acc: 0.7083
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 7ms/step - loss: 0.6347 - acc: 0.6562
[0.6347281336784363, 0.65625]
```

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

Changing the data set to Breast Cancer:

```
File Edit View Insert Cell Kernel Widgets Help
+ + + + + Run Code
In [4]: data = pd.read_csv('breastcancer.csv')

In [5]: path_to_csv = 'sample_data/breastcancer.csv'

In [6]: 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))
```

Output:

```
Epoch 87/100
14/14 [=====] - 0s 7ms/step - loss: 0.1933 - acc: 0.9296
Epoch 88/100
14/14 [=====] - 0s 6ms/step - loss: 0.1952 - acc: 0.9249
Epoch 89/100
14/14 [=====] - 0s 6ms/step - loss: 0.1861 - acc: 0.9272
Epoch 90/100
14/14 [=====] - 0s 5ms/step - loss: 0.2388 - acc: 0.9131
Epoch 91/100
14/14 [=====] - 0s 5ms/step - loss: 0.1765 - acc: 0.9366
Epoch 92/100
14/14 [=====] - 0s 6ms/step - loss: 0.1699 - acc: 0.9460
Epoch 93/100
14/14 [=====] - 0s 4ms/step - loss: 0.1880 - acc: 0.9413
Epoch 94/100
14/14 [=====] - 0s 4ms/step - loss: 0.1714 - acc: 0.9296
Epoch 95/100
14/14 [=====] - 0s 3ms/step - loss: 0.1655 - acc: 0.9437
Epoch 96/100
```

```
Epoch 97/100
14/14 [=====] - 0s 5ms/step - loss: 0.1937 - acc: 0.9343
Epoch 100/100
14/14 [=====] - 0s 5ms/step - loss: 0.1588 - acc: 0.9319
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: 0.3772 - acc: 0.8881
[0.3772197365760803, 0.8881118893623352]
```

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

```
In [7]: #read the data
data = pd.read_csv('breastcancer.csv')

In [8]: path_to_csv = 'breastcancer.csv'

In [9]: from sklearn.preprocessing import StandardScaler
sc = StandardScaler()

In [10]: 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, activations='relu')) # hidden layer 1
my_nn.add(Dense(1, activations='sigmoid')) # output layer
my_nn.compile(loss='binary_crossentropy', optimizer='adam', metrics=['acc'])
my_nn.fit(X_train, Y_train, epochs=100,
          initial_epoch=0)

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

Output:

```
14/14 [=====] - 0s 8ms/step - loss: 0.8400 - acc: 0.8005
Epoch 4/100
14/14 [=====] - 0s 8ms/step - loss: 0.7796 - acc: 0.8545
Epoch 5/100
14/14 [=====] - 0s 16ms/step - loss: 0.6923 - acc: 0.8779
Epoch 6/100
14/14 [=====] - 0s 13ms/step - loss: 0.6723 - acc: 0.8873
Epoch 7/100
14/14 [=====] - 0s 12ms/step - loss: 0.6227 - acc: 0.8826
Epoch 8/100
14/14 [=====] - 0s 6ms/step - loss: 0.7284 - acc: 0.8967
Epoch 9/100
14/14 [=====] - 0s 6ms/step - loss: 0.7737 - acc: 0.8709
Epoch 10/100
14/14 [=====] - 0s 7ms/step - loss: 0.5930 - acc: 0.8967
Epoch 11/100
14/14 [=====] - 0s 6ms/step - loss: 0.5163 - acc: 0.9108
Epoch 12/100
14/14 [=====] - 0s 6ms/step - loss: 0.5073 - acc: 0.8967
Epoch 13/100
```

```
14/14 [=====] - 0s 6ms/step - loss: 0.1422 - acc: 0.9343
Epoch 100/100
14/14 [=====] - 0s 5ms/step - loss: 0.1408 - acc: 0.9319
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 7ms/step - loss: 0.2976 - acc: 0.9231
[0.29757022857666016, 0.9230769276618958]
```

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.

localhost8888/notebooks/ICP6.ipynb

jupyter ICP6 Last Checkpoint 8 hours ago (autosaved)

Python 3 (pykernel)

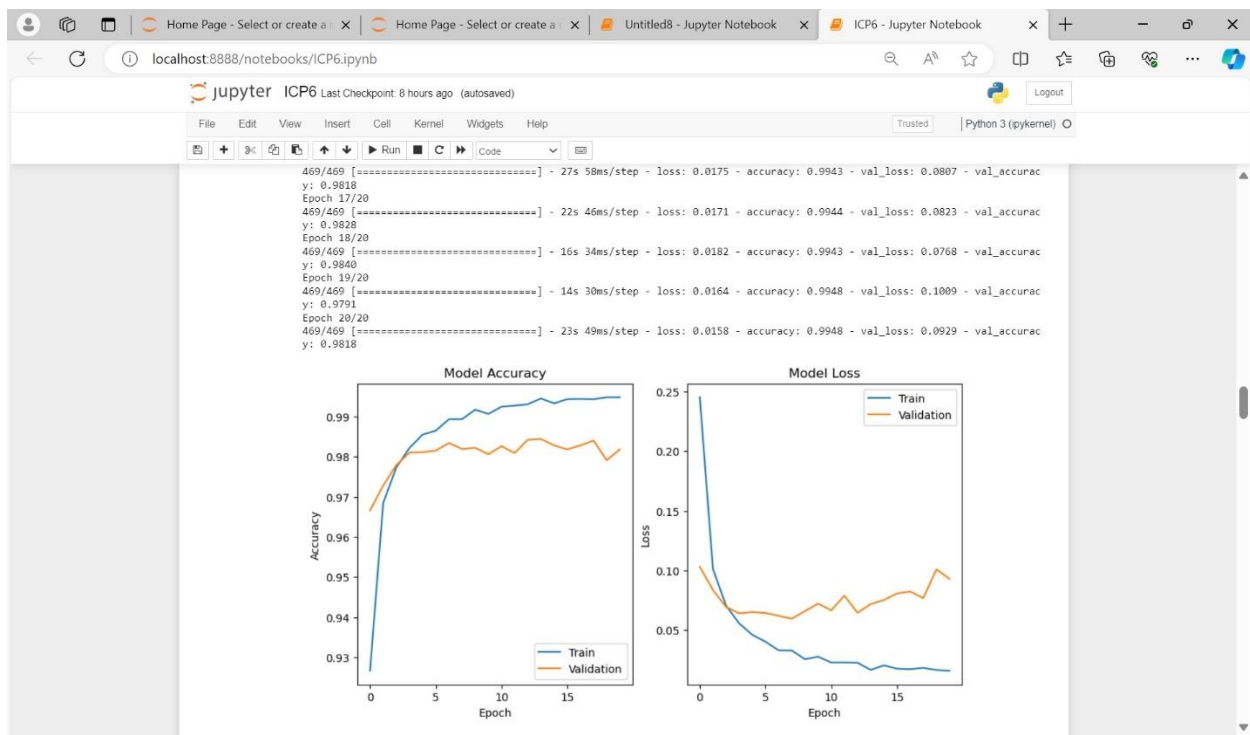
In [11]:

```
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()
```

Output:

```
Download data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 [=====] - 1s 0us/step
Epoch 1/20
469/469 [=====] - 27s 53ms/step - loss: 0.2453 - accuracy: 0.9266 - val_loss: 0.1030 - val_accuracy: 0.9666
Epoch 2/20
469/469 [=====] - 27s 59ms/step - loss: 0.1013 - accuracy: 0.9685 - val_loss: 0.0836 - val_accuracy: 0.9728
Epoch 3/20
469/469 [=====] - 28s 60ms/step - loss: 0.0704 - accuracy: 0.9773 - val_loss: 0.0693 - val_accuracy: 0.9779
Epoch 4/20
469/469 [=====] - 21s 44ms/step - loss: 0.0555 - accuracy: 0.9822 - val_loss: 0.0640 - val_accuracy: 0.9810
Epoch 5/20
469/469 [=====] - 17s 37ms/step - loss: 0.0459 - accuracy: 0.9855 - val_loss: 0.0651 - val_accuracy: 0.9811
Epoch 6/20
469/469 [=====] - 24s 51ms/step - loss: 0.0402 - accuracy: 0.9865 - val_loss: 0.0643 - val_accuracy: 0.9815
Epoch 7/20
469/469 [=====] - 27s 58ms/step - loss: 0.0329 - accuracy: 0.9893 - val_loss: 0.0619 - val_accuracy: 0.9834
Epoch 8/20
469/469 [=====] - 24s 52ms/step - loss: 0.0328 - accuracy: 0.9893 - val_loss: 0.0595 - val_accuracy: 0.9819
Epoch 9/20
469/469 [=====] - 28s 60ms/step - loss: 0.0255 - accuracy: 0.9917 - val_loss: 0.0658 - val_accuracy: 0.9822
Epoch 10/20
469/469 [=====] - 19s 40ms/step - loss: 0.0276 - accuracy: 0.9906 - val_loss: 0.0722 - val_accuracy: 0.9806
Epoch 11/20
469/469 [=====] - 28s 59ms/step - loss: 0.0227 - accuracy: 0.9924 - val_loss: 0.0665 - val_accuracy: 0.9826
Epoch 12/20
469/469 [=====] - 29s 61ms/step - loss: 0.0227 - accuracy: 0.9927 - val_loss: 0.0789 - val_accuracy: 0.9809
Epoch 13/20
469/469 [=====] - 22s 48ms/step - loss: 0.0225 - accuracy: 0.9930 - val_loss: 0.0644 - val_accuracy: 0.9842
Epoch 14/20
469/469 [=====] - 27s 58ms/step - loss: 0.0165 - accuracy: 0.9945 - val_loss: 0.0717 - val_accuracy: 0.9818
```



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.

localhost:8888/notebooks/ICP6.ipynb

Jupyter ICP6 Last Checkpoint: 8 hours ago (autosaved)

```
In [12]: 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 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'])
model.fit(x_train.reshape(-1, 784), y_train, validation_data=(x_test.reshape(-1, 784), y_test),
          epochs=20, batch_size=128)

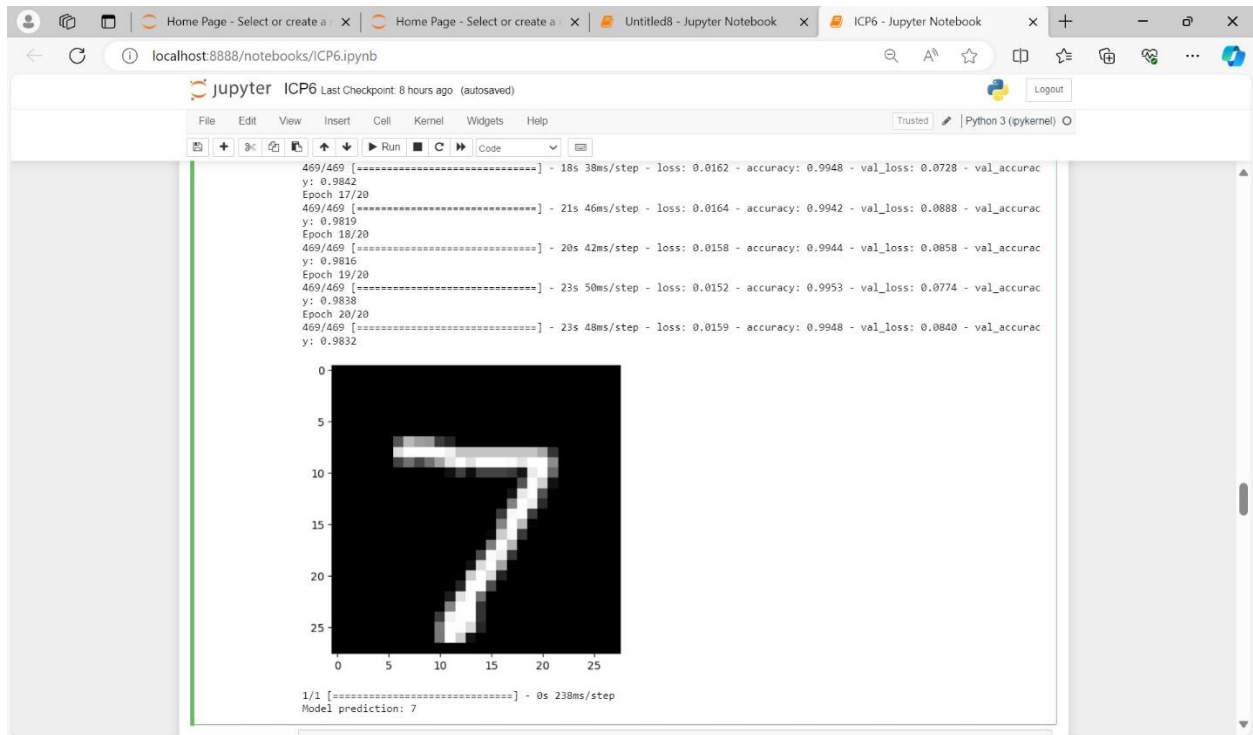
# plot one of the images in the test data
plt.imshow(x_test[0], cmap='gray')
plt.show()

# 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))
```

localhost:8888/notebooks/ICP6.ipynb

Jupyter ICP6 Last Checkpoint: 8 hours ago (autosaved)

```
Epoch 1/20
469/469 [=====] - 30s 60ms/step - loss: 0.2487 - accuracy: 0.9258 - val_loss: 0.0996 - val_accuracy: 0.9670
Epoch 2/20
469/469 [=====] - 20s 42ms/step - loss: 0.0989 - accuracy: 0.9692 - val_loss: 0.0788 - val_accuracy: 0.9762
Epoch 3/20
469/469 [=====] - 20s 43ms/step - loss: 0.0719 - accuracy: 0.9776 - val_loss: 0.0703 - val_accuracy: 0.9783
Epoch 4/20
469/469 [=====] - 23s 49ms/step - loss: 0.0553 - accuracy: 0.9828 - val_loss: 0.0695 - val_accuracy: 0.9788
Epoch 5/20
469/469 [=====] - 18s 39ms/step - loss: 0.0443 - accuracy: 0.9857 - val_loss: 0.0690 - val_accuracy: 0.9790
Epoch 6/20
469/469 [=====] - 26s 56ms/step - loss: 0.0394 - accuracy: 0.9869 - val_loss: 0.0680 - val_accuracy: 0.9801
Epoch 7/20
469/469 [=====] - 26s 55ms/step - loss: 0.0356 - accuracy: 0.9881 - val_loss: 0.0707 - val_accuracy: 0.9805
Epoch 8/20
469/469 [=====] - 20s 42ms/step - loss: 0.0312 - accuracy: 0.9895 - val_loss: 0.0673 - val_accuracy: 0.9809
Epoch 9/20
469/469 [=====] - 20s 43ms/step - loss: 0.0260 - accuracy: 0.9916 - val_loss: 0.0674 - val_accuracy: 0.9817
Epoch 10/20
469/469 [=====] - 26s 56ms/step - loss: 0.0244 - accuracy: 0.9916 - val_loss: 0.0818 - val_accuracy: 0.9798
Epoch 11/20
469/469 [=====] - 24s 51ms/step - loss: 0.0231 - accuracy: 0.9924 - val_loss: 0.0632 - val_accuracy: 0.9843
Epoch 12/20
469/469 [=====] - 27s 58ms/step - loss: 0.0224 - accuracy: 0.9921 - val_loss: 0.0609 - val_accuracy: 0.9844
Epoch 13/20
469/469 [=====] - 28s 61ms/step - loss: 0.0188 - accuracy: 0.9932 - val_loss: 0.0746 - val_accuracy: 0.9819
Epoch 14/20
469/469 [=====] - 24s 52ms/step - loss: 0.0185 - accuracy: 0.9937 - val_loss: 0.0715 - val_accuracy: 0.9836
Epoch 15/20
```



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.


```
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

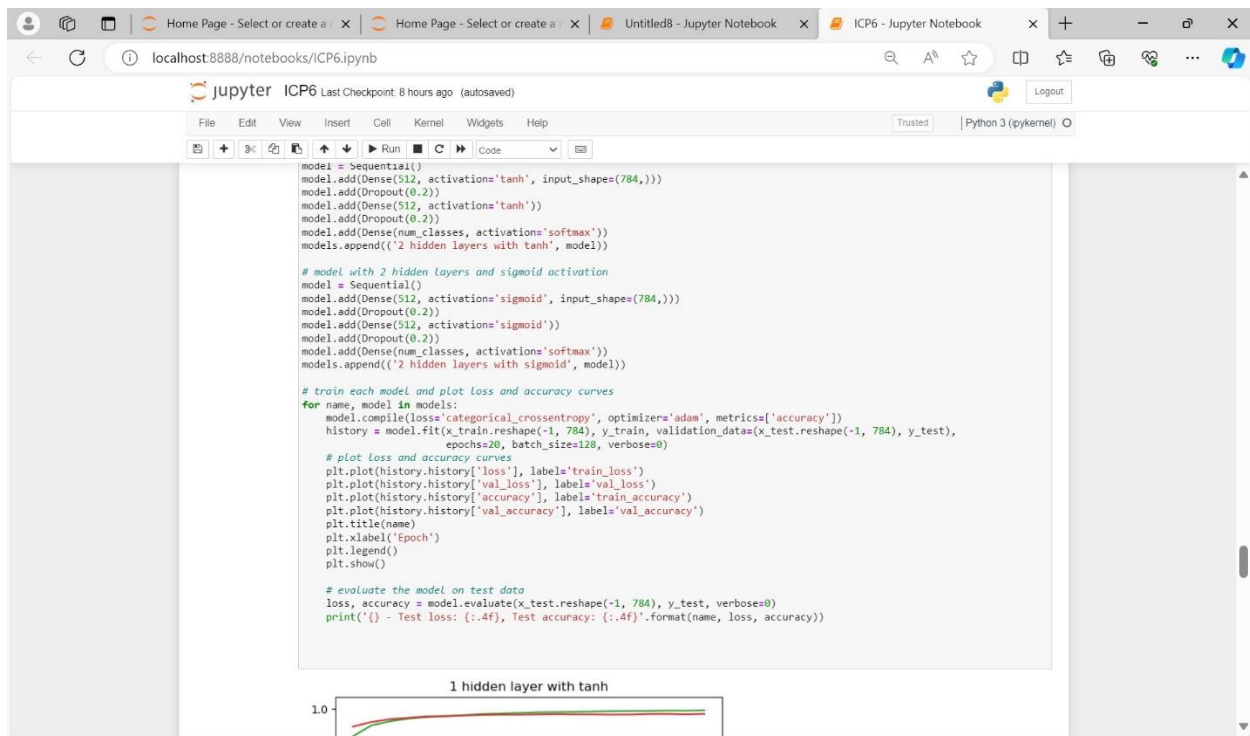
# convert class labels to binary class matrices
num_classes = 10
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)

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

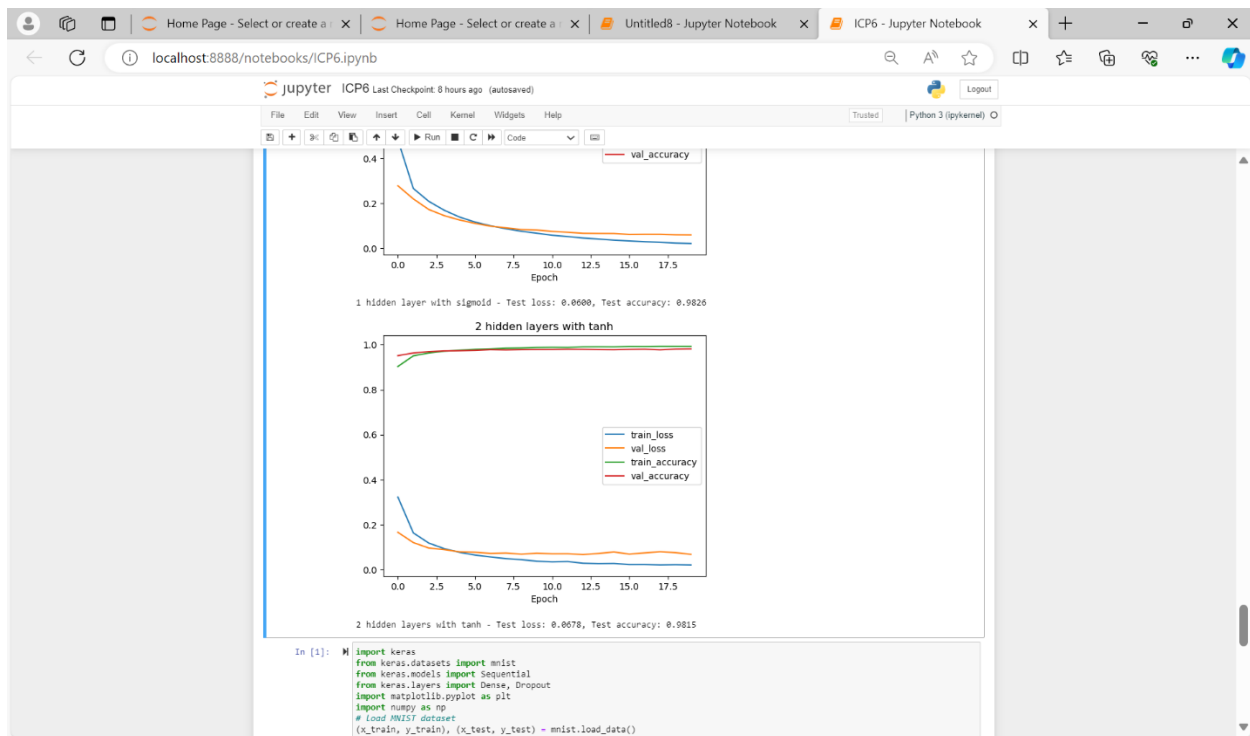
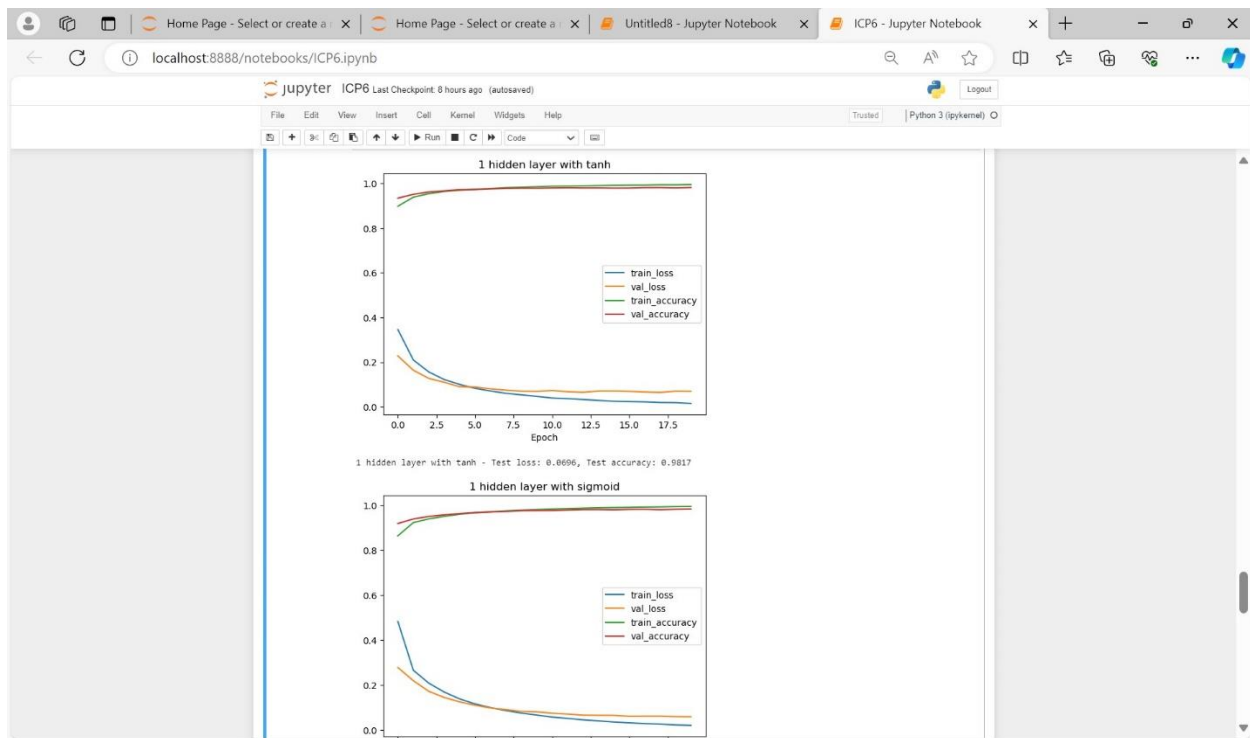
# model with 1 hidden layer and tanh activation
model = Sequential()
model.add(Dense(512, activation='tanh', input_shape=(784,)))
model.add(Dropout(0.2))
model.add(Dense(num_classes, activation='softmax'))
models.append(('1 hidden layer with tanh', model))

# 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(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))
```



Output:



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

```
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'))
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))

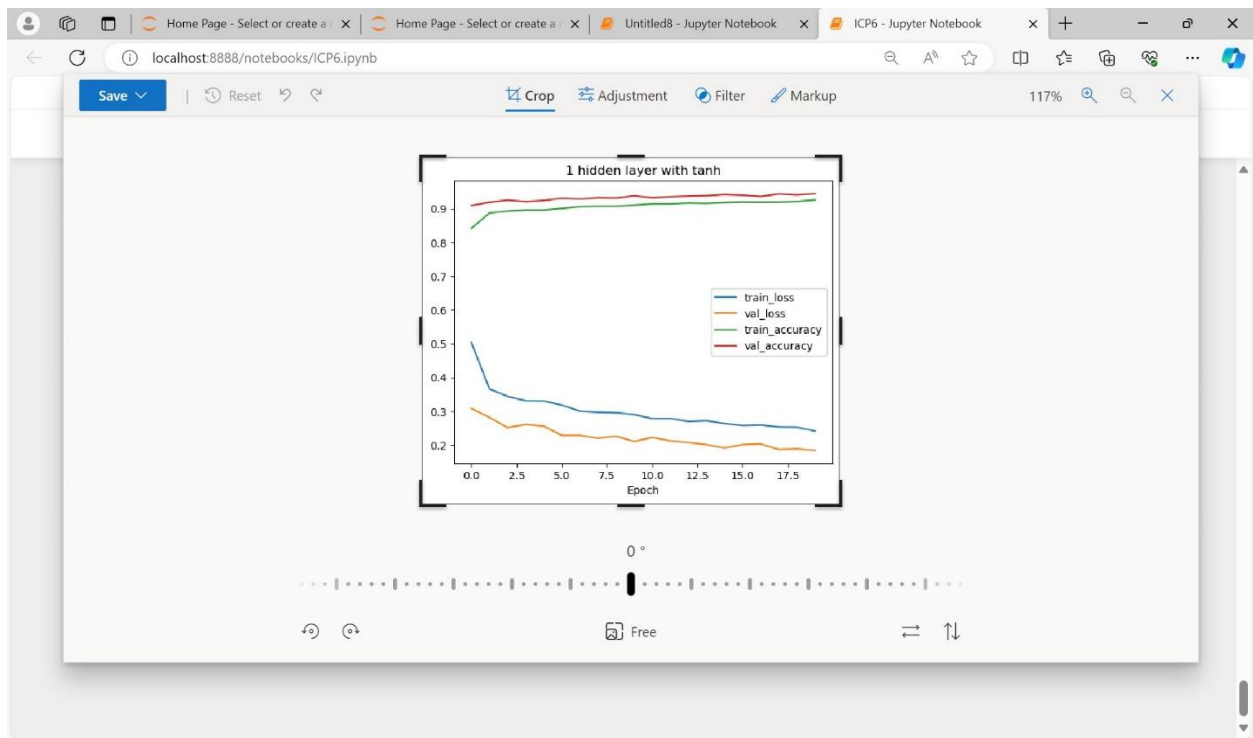
# model with 2 hidden layers and sigmoid activation
model = Sequential()
model.add(Dense(512, activation='sigmoid', input_shape=(784,)))
model.add(Dropout(0.2))
model.add(Dense(512, activation='sigmoid'))
model.add(Dropout(0.2))
model.add(Dense(num_classes, activation='softmax'))
models.append(('2 hidden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

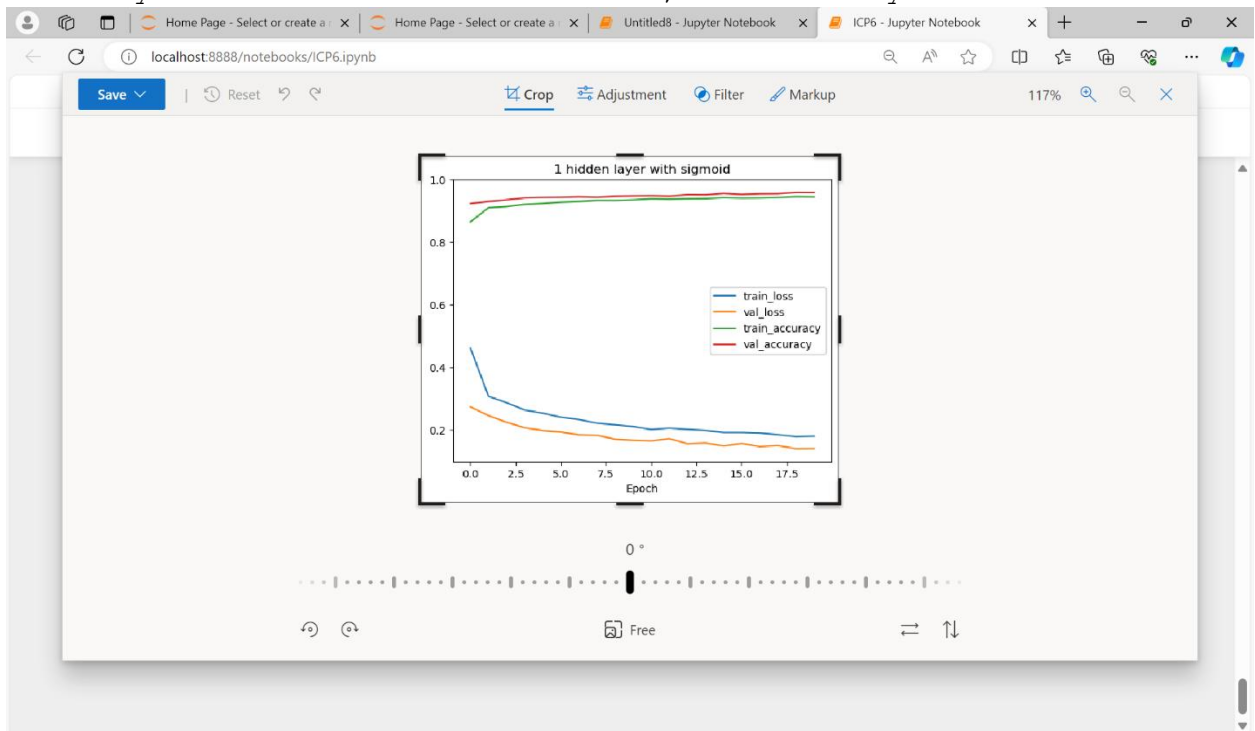
    # plot loss and accuracy curves
    plt.plot(history.history['loss'], label='train_loss')
    plt.plot(history.history['val_loss'], label='val_loss')
    plt.plot(history.history['accuracy'], label='train_accuracy')
    plt.plot(history.history['val_accuracy'], label='val_accuracy')
    plt.title(name)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

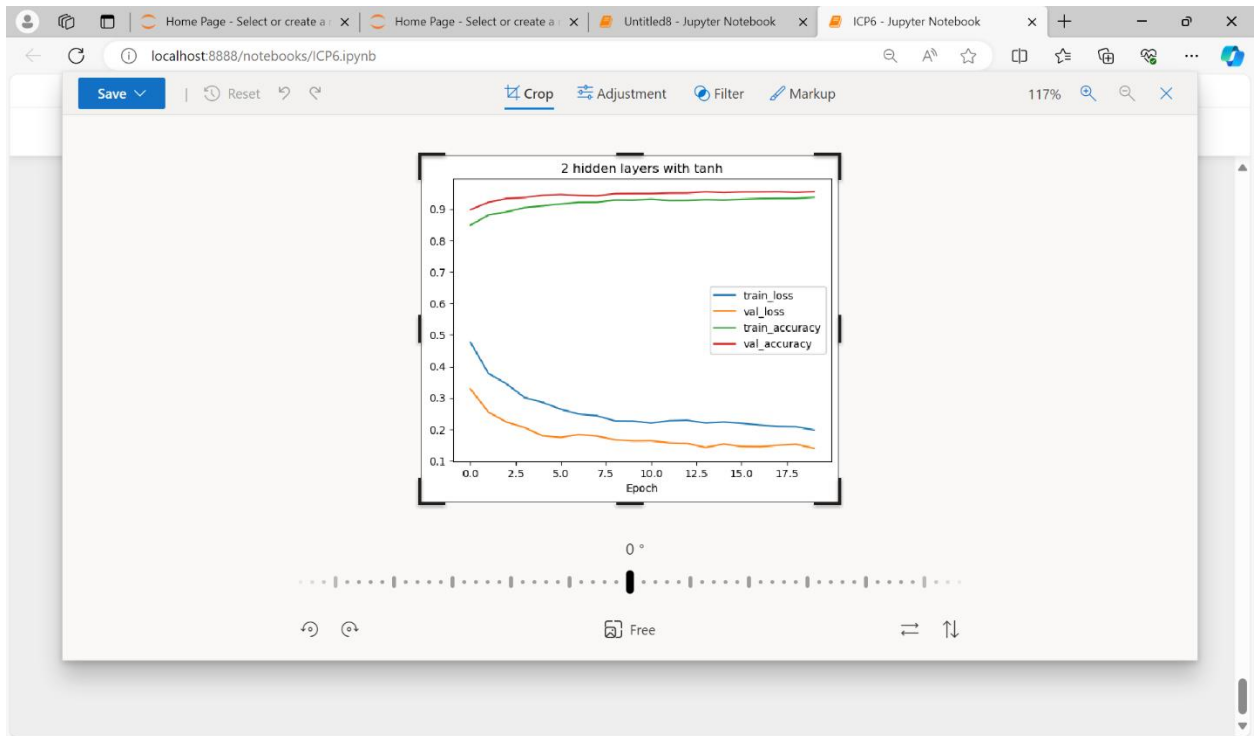
Output:



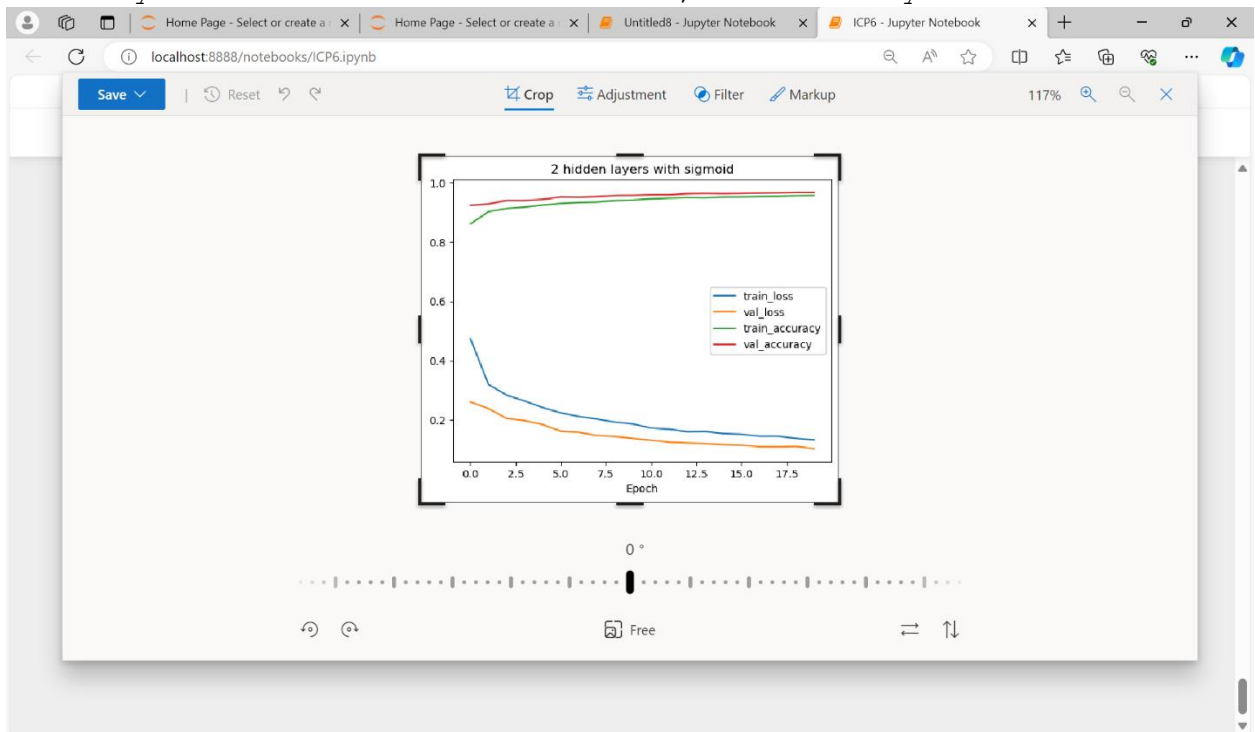
1 hidden layer with tanh - Test loss: 0.1844, Test accuracy: 0.9449



1 hidden layer with sigmoid - Test loss: 0.1392, Test accuracy: 0.9591



2 hidden layers with tanh - Test loss: 0.1394, Test accuracy: 0.9551



2 hidden layers with sigmoid - Test loss: 0.1039, Test accuracy: 0.9667