# Summer 2024: CS-5720
# Neural Networks & Deep Learning - ICP-5

Name: Venkat Tejaswini Pasupuleti

ID: 700755556

GITHUB LINK: https://github.com/Tejaswinipasupuleti45/NN_DL_ICP5

## CODE & SCREENSHOTS FOR RESULTS:

Lesson Overview:
In this lesson, we are going to discuss types and applications of Autoencoder.

Programming elements:
1. Basics of Autoencoders
2. Role of Autoencoders in unsupervised learning
3. Types of Autoencoders
4. Use case: Simple autoencoder-Reconstructing the existing image, which will contain most important features of the image
5. Use case: Stacked autoencoder

In class programming:

1. Add one more hidden layer to autoencoder
2. Do the prediction on the test data and then visualize one of the reconstructed version of that test data. Also, visualize the same test data before reconstruction using **Matplotlib**

3. Repeat the question 2 on the denoisening autoencoder

4. plot loss and accuracy using the history object

```python
from keras.layers import Input, Dense
from keras.models import Model

# this is the size of our encoded representations
encoding_dim = 32  # 32 floats -> compression of factor 24.5, assuming the input is 784 floats

# this is our input placeholder
input_img = Input(shape=(784,))
# "encoded" is the encoded representation of the input
encoded = Dense(encoding_dim, activation='relu')(input_img)
# "decoded" is the lossy reconstruction of the input
decoded = Dense(784, activation='sigmoid')(encoded)
# this model maps an input to its reconstruction
autoencoder = Model(input_img, decoded)
# this model maps an input to its encoded representation
autoencoder.compile(optimizer='adadelta', loss='binary_crossentropy')
from keras.datasets import mnist, fashion_mnist
import numpy as np
(x_train, y_train), (x_test, y_test) = fashion_mnist.load_data()
x_train = x_train.astype('float32') / 255.
```

```python
                epochs=5,
                batch_size=256,
                shuffle=True,
                validation_data=(x_test, x_test))
```

```
ownloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-labels-idx1-ubyte.gz
9515/29515 [==============================] - 0s 0us/step
ownloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-images-idx3-ubyte.gz
6421880/26421880 [==============================] - 0s 0us/step
ownloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-labels-idx1-ubyte.gz
148/5148 [==============================] - 0s 0us/step
ownloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-images-idx3-ubyte.gz
422102/4422102 [==============================] - 0s 0us/step
poch 1/5
35/235 [==============================] - 13s 41ms/step - loss: 0.6944 - val_loss: 0.6944
poch 2/5
35/235 [==============================] - 6s 24ms/step - loss: 0.6943 - val_loss: 0.6942
poch 3/5
35/235 [==============================] - 5s 21ms/step - loss: 0.6942 - val_loss: 0.6941
poch 4/5
35/235 [==============================] - 5s 22ms/step - loss: 0.6941 - val_loss: 0.6940
poch 5/5
35/235 [==============================] - 3s 12ms/step - loss: 0.6939 - val_loss: 0.6939
```

✓ 1m 28s completed at 2:56 PM

# 1.Adding hidden layer to Autoencoder :

## Program :

1.Adding hidden layer to Autoencoder

```
[ ] Start coding or generate with AI.
```

```python
from keras.layers import Input, Dense
from keras.models import Model
from keras.datasets import mnist, fashion_mnist
import numpy as np

# this is the size of our encoded representations
encoding_dim = 32
# this is our input placeholder
input_img = Input(shape=(784,))
# "encoded" is the encoded representation of the input
encoded = Dense(encoding_dim, activation='relu')(input_img)

# Adding an additional hidden layer
hidden_layer_dim = 64
hidden_layer = Dense(hidden_layer_dim, activation='relu')(encoded)
```

```python
‡ Adding an additional hidden layer
ıidden_layer_dim = 64
ıidden_layer = Dense(hidden_layer_dim, activation='relu')(encoded)

‡ "decoded" is the lossy reconstruction of the input, now connected to the hidden layer instead of 'encoded'
ɫecoded = Dense(784, activation='sigmoid')(hidden_layer)

‡ this model maps an input to its reconstruction
ıutoencoder = Model(input_img, decoded)

‡ this model maps an input to its encoded representation
ıutoencoder.compile(optimizer='adadelta', loss='binary_crossentropy')
‡ Load and prepare the data
(x_train, y_train), (x_test, y_test) = fashion_mnist.load_data()
ҳ_train = x_train.astype('float32') / 255.
ҳ_test = x_test.astype('float32') / 255.
ҳ_train = x_train.reshape((len(x_train), np.prod(x_train.shape[1:])))
ҳ_test = x_test.reshape((len(x_test), np.prod(x_test.shape[1:])))
```

```python
# Train the model
autoencoder.fit(x_train, x_train,
                epochs=5,
                batch_size=256,
                shuffle=True,
                validation_data=(x_test, x_test))
```

# Output:

```
Epoch 1/5
235/235 [==============================] - 5s 18ms/step - loss: 0.6937 - val_loss: 0.6937
Epoch 2/5
235/235 [==============================] - 6s 27ms/step - loss: 0.6937 - val_loss: 0.6936
Epoch 3/5
235/235 [==============================] - 3s 12ms/step - loss: 0.6936 - val_loss: 0.6936
Epoch 4/5
235/235 [==============================] - 3s 13ms/step - loss: 0.6935 - val_loss: 0.6935
Epoch 5/5
235/235 [==============================] - 3s 12ms/step - loss: 0.6935 - val_loss: 0.6934
<keras.src.callbacks.History at 0x7eb4774e4250>
```

2.Prediction on the test data and then visualize one of the reconstructed version of that test data. Also, visualize the same test data before reconstruction using Matplotlib.

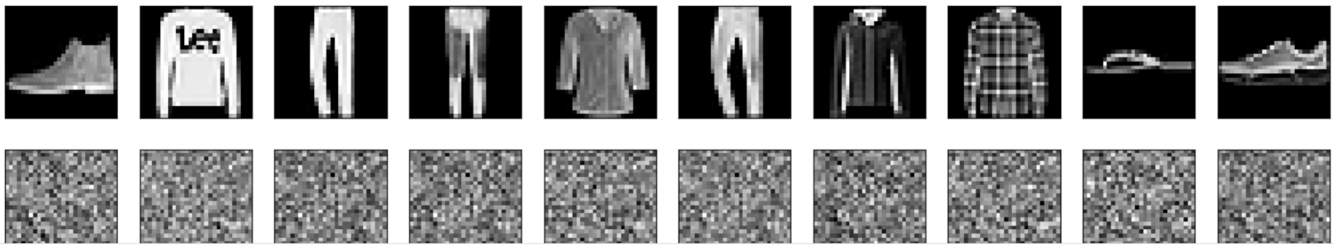
Program :

```python
from keras.layers import Input, Dense
from keras.models import Model
from keras.datasets import mnist, fashion_mnist
import numpy as np
import matplotlib.pyplot as plt

# Define the model architecture
encoding_dim = 32
hidden_layer_dim = 64

input_img = Input(shape=(784,))
encoded = Dense(encoding_dim, activation='relu')(input_img)
hidden_layer = Dense(hidden_layer_dim, activation='relu')(encoded)  # Additional hidden layer
decoded = Dense(784, activation='sigmoid')(hidden_layer)

autoencoder = Model(input_img, decoded)
autoencoder.compile(optimizer='adadelta', loss='binary_crossentropy')
```

```
# Load and prepare data
(x_train, _), (x_test, _) = fashion_mnist.load_data()
x_train = x_train.astype('float32') / 255.
x_test = x_test.astype('float32') / 255.
x_train = x_train.reshape((len(x_train), np.prod(x_train.shape[1:])))
x_test = x_test.reshape((len(x_test), np.prod(x_test.shape[1:])))

# Train the model
autoencoder.fit(x_train, x_train,
                epochs=5,
                batch_size=256,
                shuffle=True,
                validation_data=(x_test, x_test))

# Predict on the test data
decoded_imgs = autoencoder.predict(x_test)

# Visualize the original and reconstructed data
n = 10  # how many digits we will display
plt.figure(figsize=(20, 4))
for i in range(n):
    # display original
    ax = plt.subplot(2, n, i + 1)
    plt.imshow(x_test[i].reshape(28, 28))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)

    # display reconstruction
    ax = plt.subplot(2, n, i + n + 1)
    plt.imshow(decoded_imgs[i].reshape(28, 28))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)
plt.show()
```

Output :

```
Epoch 1/5
235/235 [==============================] - 5s 20ms/step - loss: 0.6937 - val_loss: 0.6937
Epoch 2/5
235/235 [==============================] - 3s 11ms/step - loss: 0.6936 - val_loss: 0.6936
Epoch 3/5
235/235 [==============================] - 3s 13ms/step - loss: 0.6935 - val_loss: 0.6935
Epoch 4/5
235/235 [==============================] - 3s 11ms/step - loss: 0.6934 - val_loss: 0.6934
Epoch 5/5
235/235 [==============================] - 4s 15ms/step - loss: 0.6933 - val_loss: 0.6933
313/313 [==============================] - 1s 2ms/step
```



3.Denoising Autoencoder - prediction on the test data and then visualize one of the reconstructed version of that test data. Also, visualize the same test data before reconstruction using Matplotlib.

Program :

```
from keras.layers import Input, Dense
from keras.models import Model
from keras.datasets import fashion_mnist
import numpy as np
import matplotlib.pyplot as plt

# Define the model architecture
encoding_dim = 32

input_img = Input(shape=(784,))
encoded = Dense(encoding_dim, activation='relu')(input_img)
decoded = Dense(784, activation='sigmoid')(encoded)

autoencoder = Model(input_img, decoded)
autoencoder.compile(optimizer='adadelta', loss='binary_crossentropy')
# Load data
(x_train, _), (x_test, _) = fashion_mnist.load_data()
x_train = x_train.astype('float32') / 255.
x_test = x_test.astype('float32') / 255.
x_train = x_train.reshape((len(x_train), np.prod(x_train.shape[1:])))
```

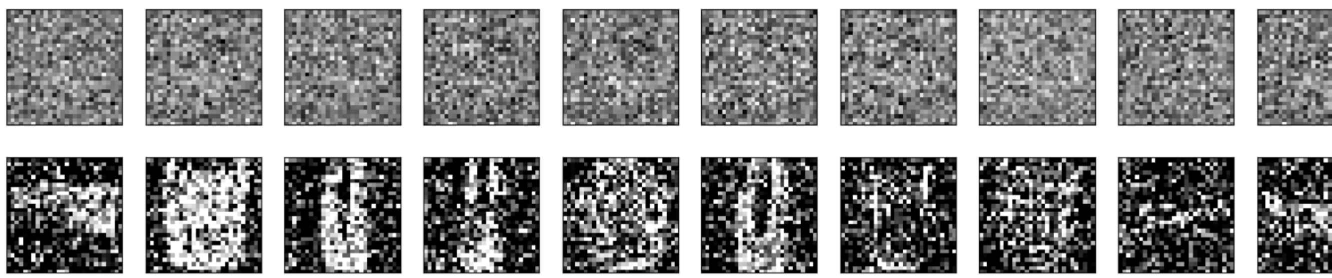
```python
x_test = x_test.reshape((len(x_test), np.prod(x_test.shape[1:])))
# Introducing noise
noise_factor = 0.5
x_train_noisy = x_train + noise_factor * np.random.normal(loc=0.0, scale=1.0, s
x_test_noisy = x_test + noise_factor * np.random.normal(loc=0.0, scale=1.0, size
x_train_noisy = np.clip(x_train_noisy, 0., 1.)
x_test_noisy = np.clip(x_test_noisy, 0., 1.)

# Train the model
autoencoder.fit(x_train_noisy, x_train,
                epochs=20,
                batch_size=256,
                shuffle=True,
                validation_data=(x_test_noisy, x_test))

# Predict on the noisy test data
decoded_imgs = autoencoder.predict(x_test_noisy)

# Visualize the noisy input and the reconstructed data
n = 10  # How many digits we will display
plt.figure(figsize=(20, 4))
for i in range(n):
```

```python
# Visualize the noisy input and the reconstructed data
n = 10  # How many digits we will display
plt.figure(figsize=(20, 4))
for i in range(n):
    # Display noisy input
    ax = plt.subplot(2, n, i + 1)
    plt.imshow(x_test_noisy[i].reshape(28, 28))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)

    # Display reconstruction
    ax = plt.subplot(2, n, i + 1 + n)
    plt.imshow(decoded_imgs[i].reshape(28, 28))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)
```

Output :

```
235/235 [==============================] - 3s 11ms/step - loss: 0.6929 - val_loss: 0.6929
Epoch 13/20
235/235 [==============================] - 3s 15ms/step - loss: 0.6928 - val_loss: 0.6927
Epoch 14/20
235/235 [==============================] - 3s 12ms/step - loss: 0.6926 - val_loss: 0.6925
Epoch 15/20
235/235 [==============================] - 2s 10ms/step - loss: 0.6925 - val_loss: 0.6924
Epoch 16/20
235/235 [==============================] - 3s 11ms/step - loss: 0.6923 - val_loss: 0.6922
Epoch 17/20
235/235 [==============================] - 3s 11ms/step - loss: 0.6922 - val_loss: 0.6921
Epoch 18/20
235/235 [==============================] - 3s 14ms/step - loss: 0.6920 - val_loss: 0.6919
Epoch 19/20
235/235 [==============================] - 3s 13ms/step - loss: 0.6919 - val_loss: 0.6918
Epoch 20/20
235/235 [==============================] - 2s 10ms/step - loss: 0.6917 - val_loss: 0.6916
313/313 [==============================] - 1s 2ms/step
```

```
Epoch 1/20
235/235 [==============================] - 4s 13ms/step - loss: 0.6949 - val_loss: 0.6948
Epoch 2/20
235/235 [==============================] - 3s 11ms/step - loss: 0.6947 - val_loss: 0.6946
Epoch 3/20
235/235 [==============================] - 2s 11ms/step - loss: 0.6945 - val_loss: 0.6944
Epoch 4/20
235/235 [==============================] - 3s 14ms/step - loss: 0.6943 - val_loss: 0.6942
Epoch 5/20
235/235 [==============================] - 3s 12ms/step - loss: 0.6941 - val_loss: 0.6940
Epoch 6/20
235/235 [==============================] - 3s 12ms/step - loss: 0.6939 - val_loss: 0.6939
Epoch 7/20
235/235 [==============================] - 2s 10ms/step - loss: 0.6938 - val_loss: 0.6937
Epoch 8/20
235/235 [==============================] - 3s 11ms/step - loss: 0.6936 - val_loss: 0.6935
Epoch 9/20
235/235 [==============================] - 4s 18ms/step - loss: 0.6934 - val_loss: 0.6933
Epoch 10/20
235/235 [==============================] - 3s 11ms/step - loss: 0.6933 - val_loss: 0.6932
Epoch 11/20
235/235 [==============================] - 3s 12ms/step - loss: 0.6931 - val_loss: 0.6930
Epoch 12/20
```



4.Plot loss and accuracy using the history object

Program :

```
from keras.layers import Input, Dense
from keras.models import Model
from keras.datasets import fashion_mnist
from keras.utils import to_categorical
import numpy as np
import matplotlib.pyplot as plt
from keras.optimizers import Adam

# Load and prepare the Fashion MNIST data
(x_train, y_train), (x_test, y_test) = fashion_mnist.load_
x_train = x_train.reshape(-1, 784).astype('float32') / 255
x_test = x_test.reshape(-1, 784).astype('float32') / 255

# Convert labels to one-hot encoding
num_classes = 10
y_train = to_categorical(y_train, num_classes)
y_test = to_categorical(y_test, num_classes)
# Model architecture
input_img = Input(shape=(784,))
encoded = Dense(128, activation='relu')(input_img)
decoded = Dense(10, activation='softmax')(encoded)  # Clas
```

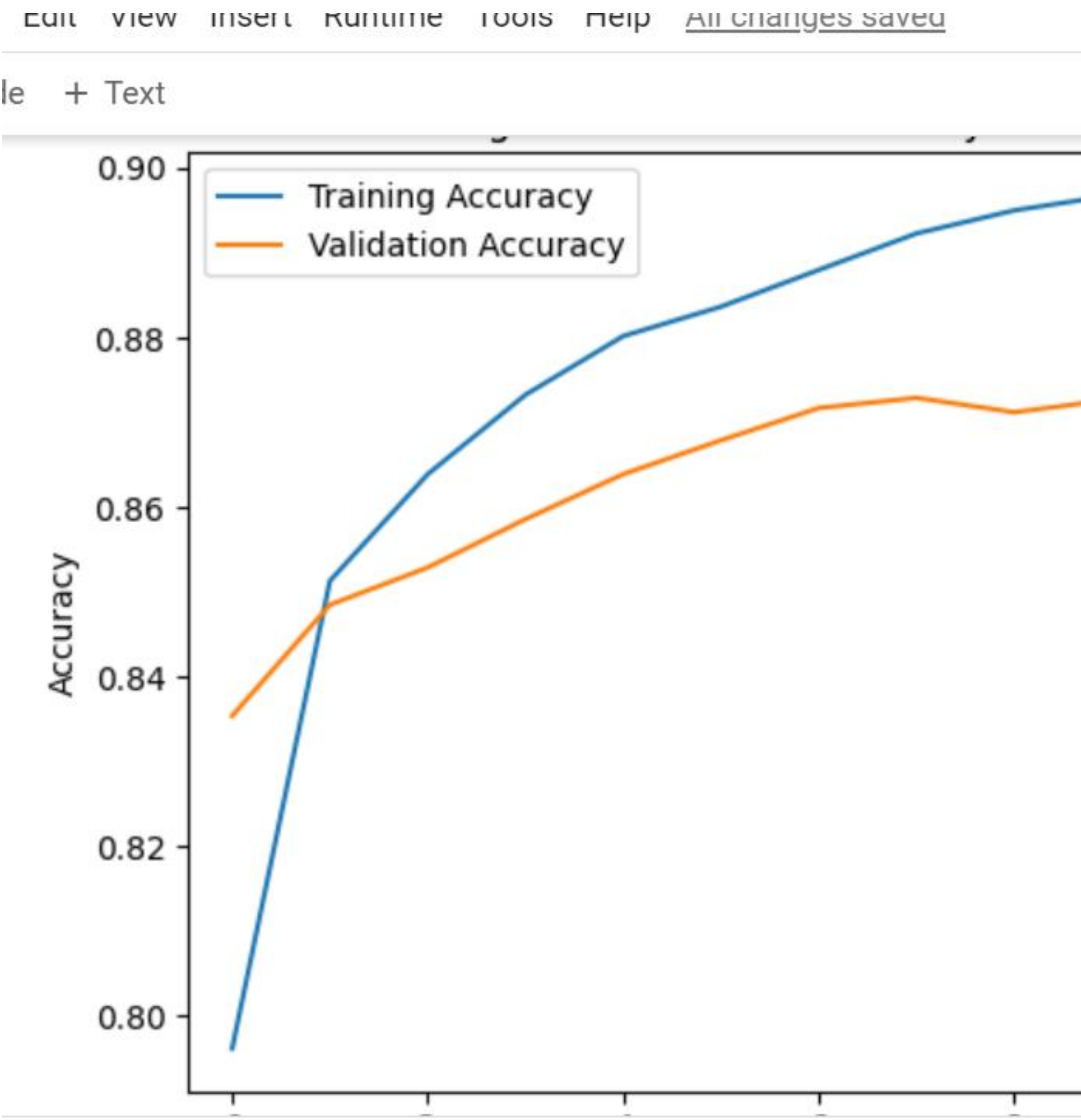
```python
# Convert labels to one-hot encoding
num_classes = 10
y_train = to_categorical(y_train, num_classes)
y_test = to_categorical(y_test, num_classes)
# Model architecture
input_img = Input(shape=(784,))
encoded = Dense(128, activation='relu')(input_img)
decoded = Dense(10, activation='softmax')(encoded)  # Clas

model = Model(input_img, decoded)
model.compile(optimizer=Adam(learning_rate=0.001), loss='c

# Train the model
history = model.fit(x_train, y_train,
                    epochs=10,
                    batch_size=256,
                    shuffle=True,
                    validation_data=(x_test, y_test))
# Plotting the training and validation loss
plt.figure(figsize=(10, 5))

# Plotting training and validation accuracy
```

```
# Train the model
history = model.fit(x_train, y_train,
                    epochs=10,
                    batch_size=256,
                    shuffle=True,
                    validation_data=(x_test, y_test))
# Plotting the training and validation loss
plt.figure(figsize=(10, 5))

# Plotting training and validation accuracy
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Training Accu
plt.plot(history.history['val_accuracy'], label='Validatio
plt.title('Training and Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

# Plotting training and validation loss
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Training Loss')
```

```
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Training Accura
plt.plot(history.history['val_accuracy'], label='Validation
plt.title('Training and Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

# Plotting training and validation loss
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss
plt.title('Training and Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.tight_layout()
plt.show()
```

Output :

```
Epoch 1/10
235/235 [==============================] - 5s 17ms/step - lo
Epoch 2/10
235/235 [==============================] - 2s 7ms/step - los
Epoch 3/10
235/235 [==============================] - 2s 7ms/step - los
Epoch 4/10
235/235 [==============================] - 2s 7ms/step - los
Epoch 5/10
235/235 [==============================] - 1s 6ms/step - los
Epoch 6/10
235/235 [==============================] - 1s 6ms/step - los
Epoch 7/10
235/235 [==============================] - 2s 7ms/step - los
Epoch 8/10
235/235 [==============================] - 3s 12ms/step - lo
Epoch 9/10
235/235 [==============================] - 2s 7ms/step - los
Epoch 10/10
235/235 [==============================] - 1s 6ms/step - los
```


Edit View Insert Runtime Tools Help All changes saved
+ Text
Training Accuracy
Validation Accuracy
Accuracy
0.90
0.88
0.86
0.84
0.82
0.80