21BAI1364 VIT – CHENNAI VITTA VISHNU DATTA

# BCSE332P – DEEP LEARNING LAB
# DEEP LEARNING LAB ASSIGNMENT – 7

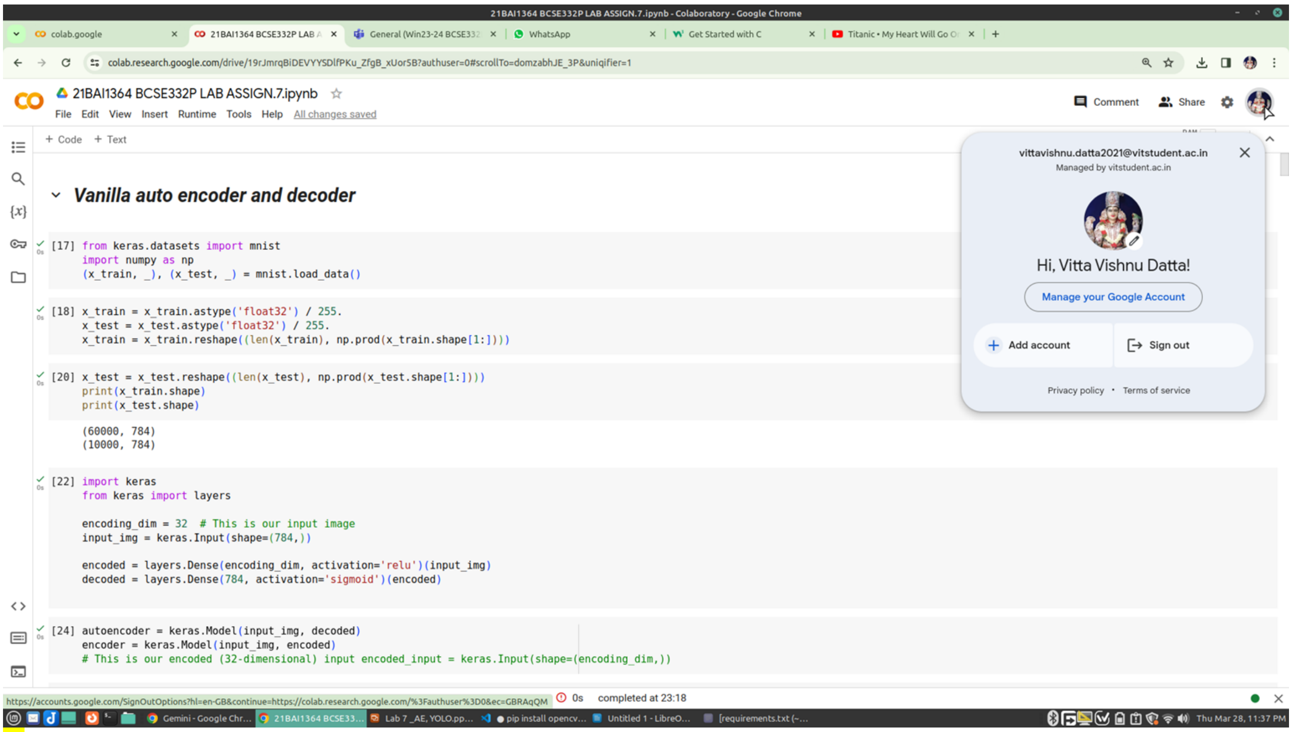
## VANILLA AUTO ENCODERS,YOLO

## CODE:

```
from keras.datasets import mnist
import numpy as np
(x_train, _), (x_test, _) = 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:])))
print(x_train.shape)
print(x_test.shape)
import keras
from keras import layers

encoding_dim = 32 # This is our input image
input_img = keras.Input(shape=(784,))

encoded = layers.Dense(encoding_dim, activation='relu')(input_img)
decoded = layers.Dense(784, activation='sigmoid')(encoded)

autoencoder = keras.Model(input_img, decoded)
encoder = keras.Model(input_img, encoded)
# This is our encoded (32-dimensional) input encoded_input = keras.Input(shape=(encoding_dim,))
decoder_layer = autoencoder.layers[-1]
# Create the decoder model
decoder = keras.Model(encoded_input, decoder_layer(encoded_input))
autoencoder.compile(optimizer='adam', loss='binary_crossentropy')
autoencoder.fit(x_train, x_train,
epochs=50,
batch_size=256,
shuffle=True, validation_data=(x_test, x_test))
encoder.predict(x_test)
decoded_imgs = decoder.predict(encoded_imgs)
import matplotlib.pyplot as plt
n = 10 # How many digits we will display
plt.figure(figsize=(20, 4))
import matplotlib.pyplot as plt # Import for plotting

for i in range(n):
# Display original (assuming x_test is your original data)
ax = plt.subplot(2, n, i + 1)
plt.imshow(x_test[i].reshape(28, 28), cmap="gray") # Ensure grayscale
ax.get_xaxis().set_visible(False)
```

```
ax.get_yaxis().set_visible(False)

# Display reconstruction (assuming decoded_imgs holds reconstructions)
ax = plt.subplot(2, n, i + 1 + n)
plt.imshow(decoded_imgs[i].reshape(28, 28), cmap="gray") # Ensure grayscale
ax.get_xaxis().set_visible(False)
ax.get_yaxis().set_visible(False)

plt.show()

from keras.datasets import mnist
import numpy as np
(x_train, _), (x_test, _) = 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:])))
noise_factor = 0.5
x_train_noisy = x_train + noise_factor * np.random.normal(loc=0.0, scale=1.0, size=x_train.shape)
x_test_noisy = x_test + noise_factor * np.random.normal(loc=0.0, scale=1.0, size=x_test.shape)
x_train_noisy = np.clip(x_train_noisy, 0., 1.)
x_test_noisy = np.clip(x_test_noisy, 0., 1.)
x_test = x_test.reshape((len(x_test), np.prod(x_test.shape[1:])))
noise_factor = 0.5
x_train_noisy = x_train + noise_factor * np.random.normal(loc=0.0, scale=1.0, size=x_train.shape)
x_test_noisy = x_test + noise_factor * np.random.normal(loc=0.0, scale=1.0, size=x_test.shape)
x_train_noisy = np.clip(x_train_noisy, 0., 1.)
x_test_noisy = np.clip(x_test_noisy, 0., 1.)
# Train your Keras autoencoder model (assuming you have the code)

# After training, obtain reconstructed images (replace with your actual logic)
decoded_imgs = autoencoder.predict(x_test_noisy)

# Visualization code (assuming x_test_noisy holds noisy images)
import matplotlib.pyplot as plt
n = 10

# ... (rest of the visualization code as before)

plt.show()

from keras import layers
import tensorflow as tf
from keras.callbacks import TensorBoard
encoding_dim = 32 # This is our input image input_img = keras.Input(shape=(784,))
encoded = layers.Dense(encoding_dim, activation='relu')(input_img)
decoded = layers.Dense(784, activation='sigmoid')(encoded)
autoencoder = keras.Model(input_img, decoded)
autoencoder.compile(optimizer='adam', loss='binary_crossentropy')
```

```
autoencoder.fit(x_train_noisy, x_train, epochs=100, batch_size=128, shuffle=True,validation_data=(x_test_noisy,
x_test),
)
encoder = keras.Model(input_img, encoded)
# This is our encoded (32-dimensional) input
encoded_input = keras.Input(shape=(encoding_dim,))
# Retrieve the last layer of the autoencoder model
decoder_layer = autoencoder.layers[-1]
# Create the decoder model
decoder = keras.Model(encoded_input, decoder_layer(encoded_input))
encoded_imgs = encoder.predict(x_test_noisy)
decoded_imgs = decoder.predict(encoded_imgs)
import matplotlib.pyplot as plt

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_noisy[i].reshape(28, 28))
plt.gray()

plt.show()

import matplotlib.pyplot as plt

n = 10 # Number of images to display
plt.figure(figsize=(20, 4)) # Adjust figure size as needed

for i in range(n):
# Display original
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)

plt.show()
```



21BAI1364 BCSE332P LAB ASSIGN.7.ipynb

## Vanilla auto encoder and decoder

```
[17] from keras.datasets import mnist
     import numpy as np
     (x_train, _), (x_test, _) = mnist.load_data()
```

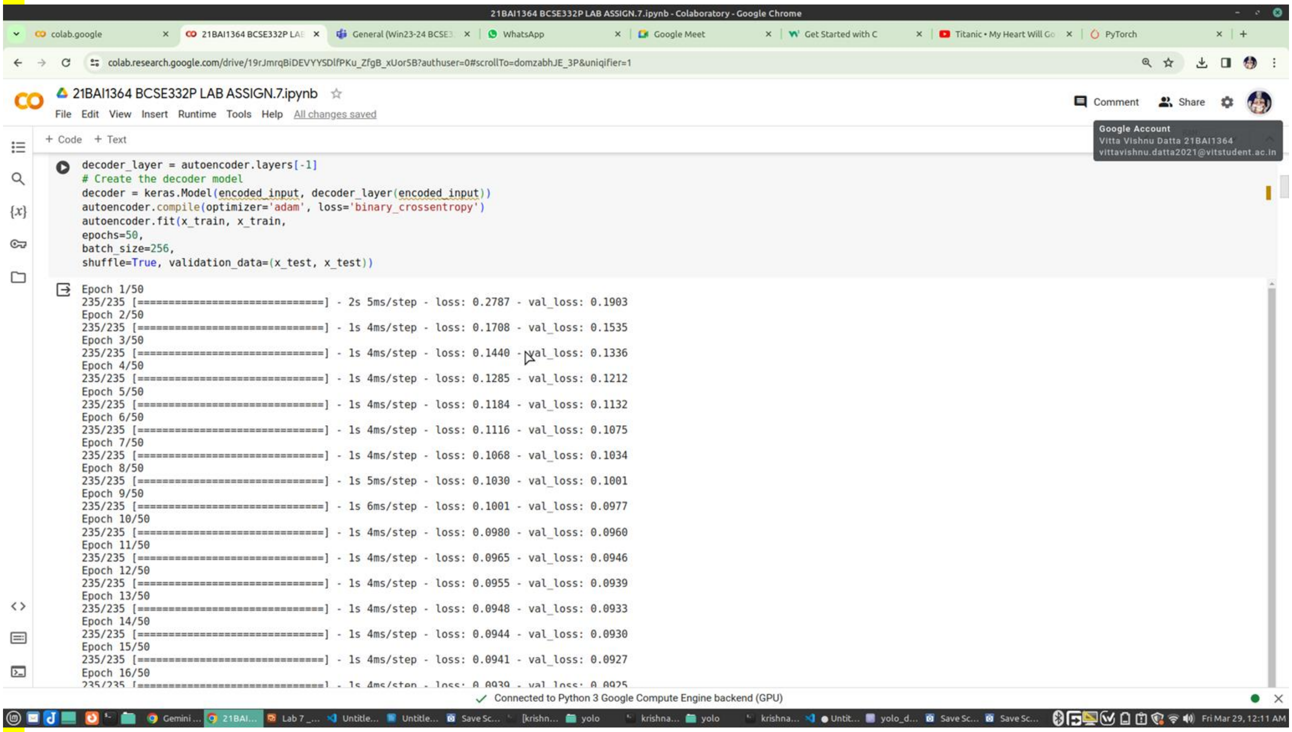
```
[18] x_train = x_train.astype('float32') / 255.
     x_test = x_test.astype('float32') / 255.
     x_train = x_train.reshape((len(x_train), np.prod(x_train.shape[1:])))
```

```
[20] x_test = x_test.reshape((len(x_test), np.prod(x_test.shape[1:])))
     print(x_train.shape)
     print(x_test.shape)

     (60000, 784)
     (10000, 784)
```

```
[22] import keras
     from keras import layers

     encoding_dim = 32  # This is our input image
     input_img = keras.Input(shape=(784,))

     encoded = layers.Dense(encoding_dim, activation='relu')(input_img)
     decoded = layers.Dense(784, activation='sigmoid')(encoded)
```

```
[24] autoencoder = keras.Model(input_img, decoded)
     encoder = keras.Model(input_img, encoded)
     # This is our encoded (32-dimensional) input encoded_input = keras.Input(shape=(encoding_dim,))
```

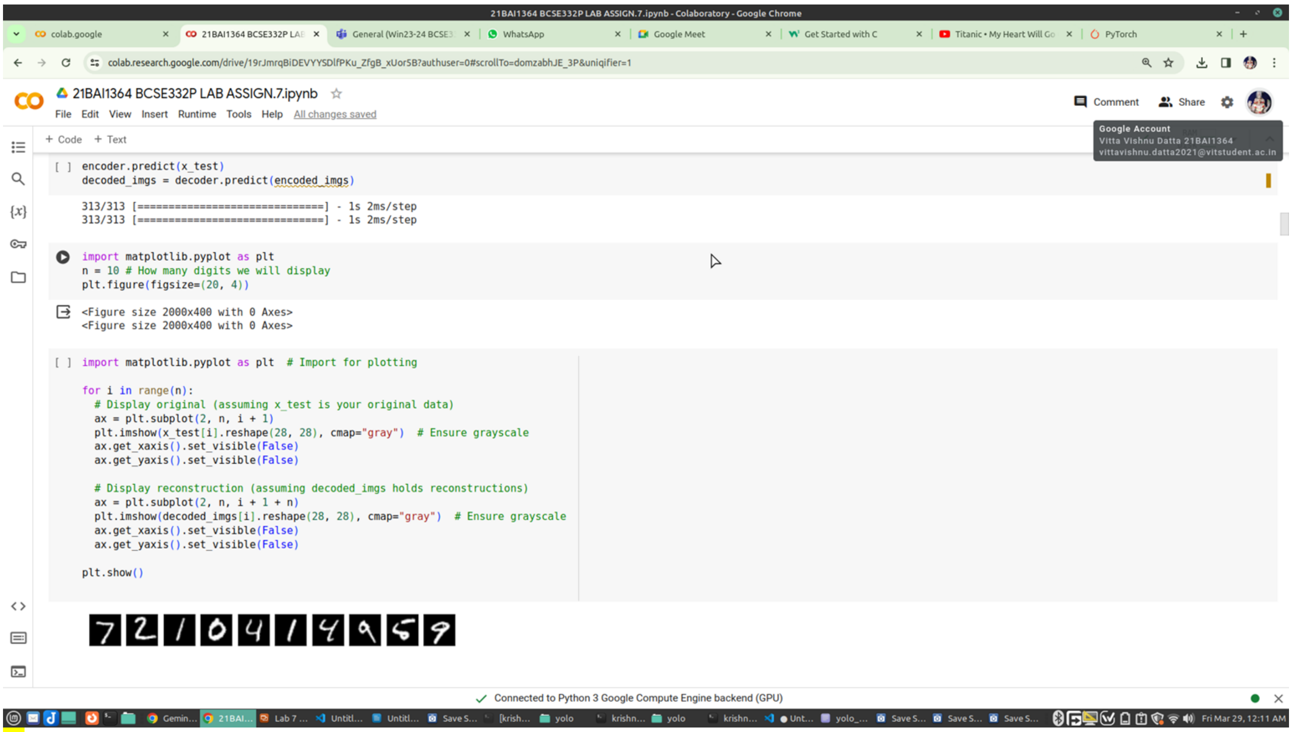
2



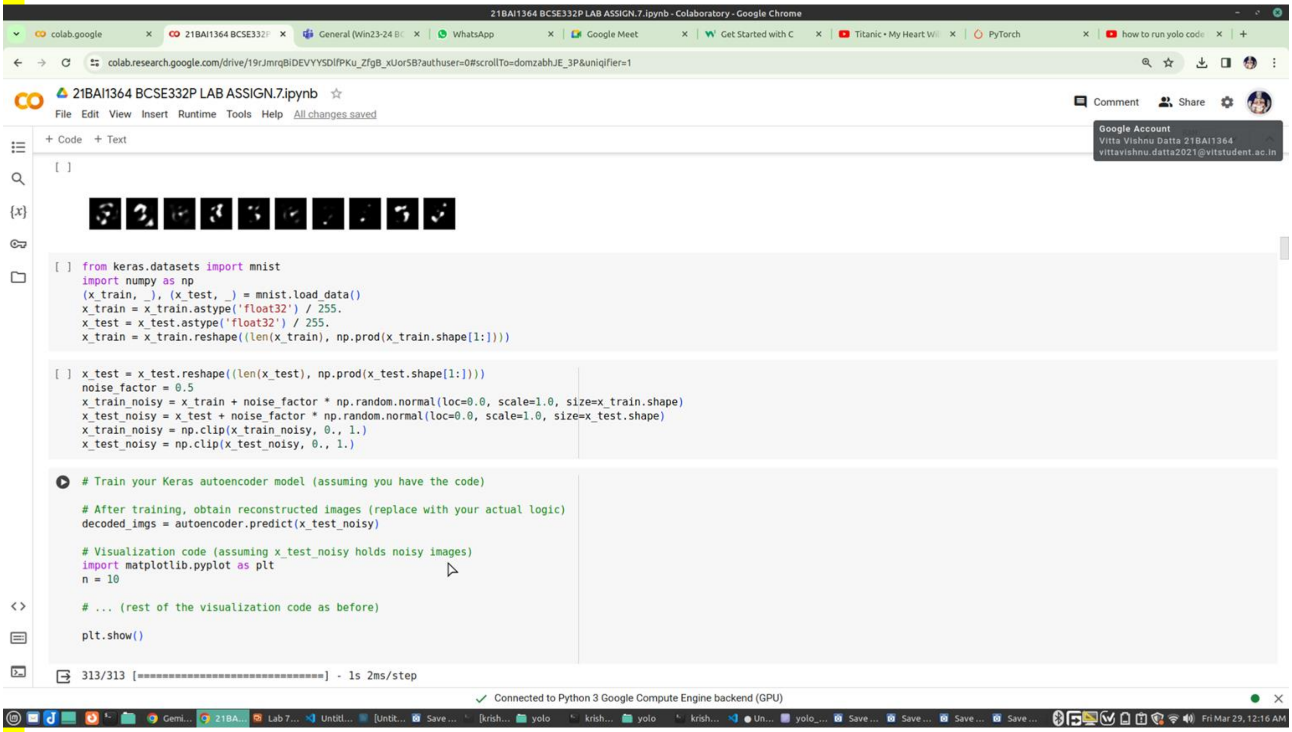
```
decoder_layer = autoencoder.layers[-1]
# Create the decoder model
decoder = keras.Model(encoded_input, decoder_layer(encoded_input))
autoencoder.compile(optimizer='adam', loss='binary_crossentropy')
autoencoder.fit(x_train, x_train,
epochs=50,
batch_size=256,
shuffle=True, validation_data=(x_test, x_test))
```

```
Epoch 1/50
235/235 [==============================] - 2s 5ms/step - loss: 0.2787 - val_loss: 0.1903
Epoch 2/50
235/235 [==============================] - 1s 4ms/step - loss: 0.1708 - val_loss: 0.1535
Epoch 3/50
235/235 [==============================] - 1s 4ms/step - loss: 0.1440 - val_loss: 0.1336
Epoch 4/50
235/235 [==============================] - 1s 4ms/step - loss: 0.1285 - val_loss: 0.1212
Epoch 5/50
235/235 [==============================] - 1s 4ms/step - loss: 0.1184 - val_loss: 0.1132
Epoch 6/50
235/235 [==============================] - 1s 4ms/step - loss: 0.1116 - val_loss: 0.1075
Epoch 7/50
235/235 [==============================] - 1s 4ms/step - loss: 0.1068 - val_loss: 0.1034
Epoch 8/50
235/235 [==============================] - 1s 5ms/step - loss: 0.1030 - val_loss: 0.1001
Epoch 9/50
235/235 [==============================] - 1s 6ms/step - loss: 0.1001 - val_loss: 0.0977
Epoch 10/50
235/235 [==============================] - 1s 4ms/step - loss: 0.0980 - val_loss: 0.0960
Epoch 11/50
235/235 [==============================] - 1s 4ms/step - loss: 0.0965 - val_loss: 0.0946
Epoch 12/50
235/235 [==============================] - 1s 4ms/step - loss: 0.0955 - val_loss: 0.0939
Epoch 13/50
235/235 [==============================] - 1s 4ms/step - loss: 0.0948 - val_loss: 0.0933
Epoch 14/50
235/235 [==============================] - 1s 4ms/step - loss: 0.0944 - val_loss: 0.0930
Epoch 15/50
235/235 [==============================] - 1s 4ms/step - loss: 0.0941 - val_loss: 0.0927
Epoch 16/50
235/235 [==============================] - 1s 4ms/step - loss: 0.0939 - val_loss: 0.0925
```

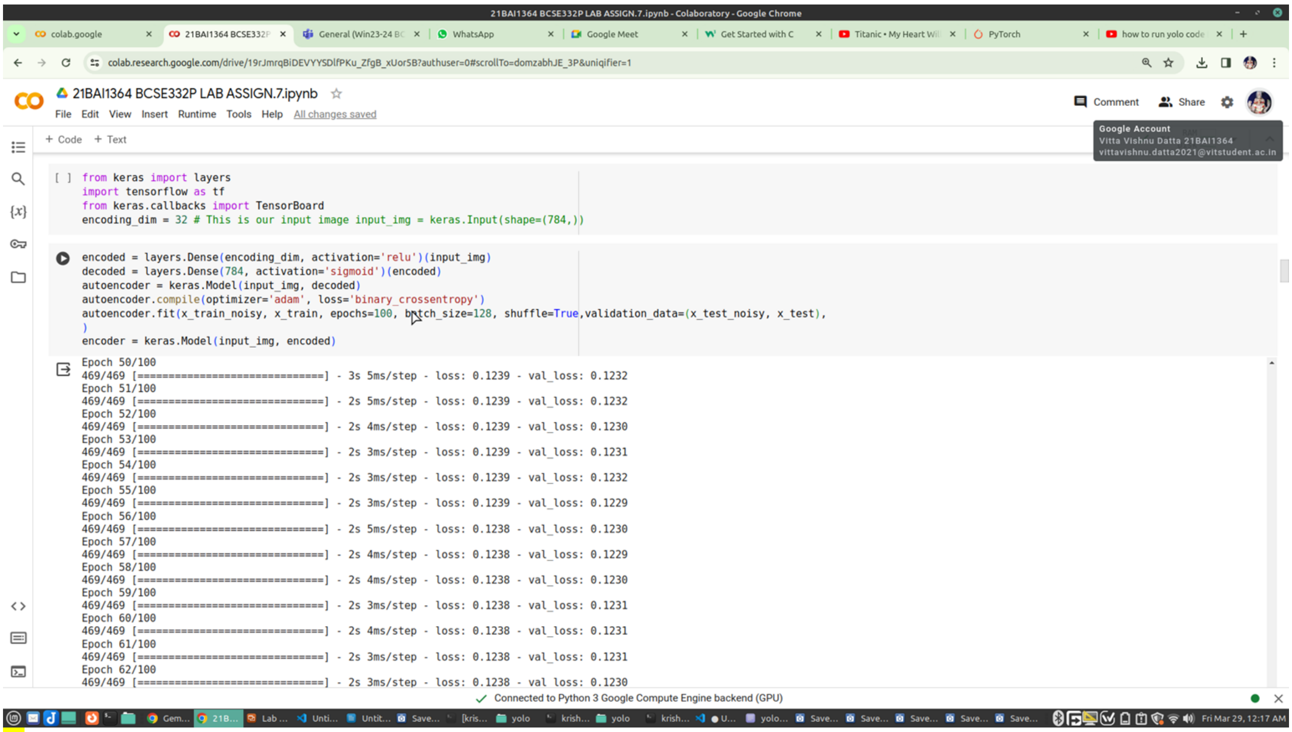
3


21BAI1364 BCSE332P LAB ASSIGN.7.ipynb
File Edit View Insert Runtime Tools Help All changes saved
encoder.predict(x_test)
decoded_imgs = decoder.predict(encoded_imgs)
313/313 [==============================] - 1s 2ms/step
313/313 [==============================] - 1s 2ms/step
import matplotlib.pyplot as plt
n = 10 # How many digits we will display
plt.figure(figsize=(20, 4))
<Figure size 2000x400 with 0 Axes>
<Figure size 2000x400 with 0 Axes>
import matplotlib.pyplot as plt # Import for plotting
for i in range(n):
# Display original (assuming x_test is your original data)
ax = plt.subplot(2, n, i + 1)
plt.imshow(x_test[i].reshape(28, 28), cmap="gray") # Ensure grayscale
ax.get_xaxis().set_visible(False)
ax.get_yaxis().set_visible(False)
# Display reconstruction (assuming decoded_imgs holds reconstructions)
ax = plt.subplot(2, n, i + 1 + n)
plt.imshow(decoded_imgs[i].reshape(28, 28), cmap="gray") # Ensure grayscale
ax.get_xaxis().set_visible(False)
ax.get_yaxis().set_visible(False)
plt.show()
Connected to Python 3 Google Compute Engine backend (GPU)

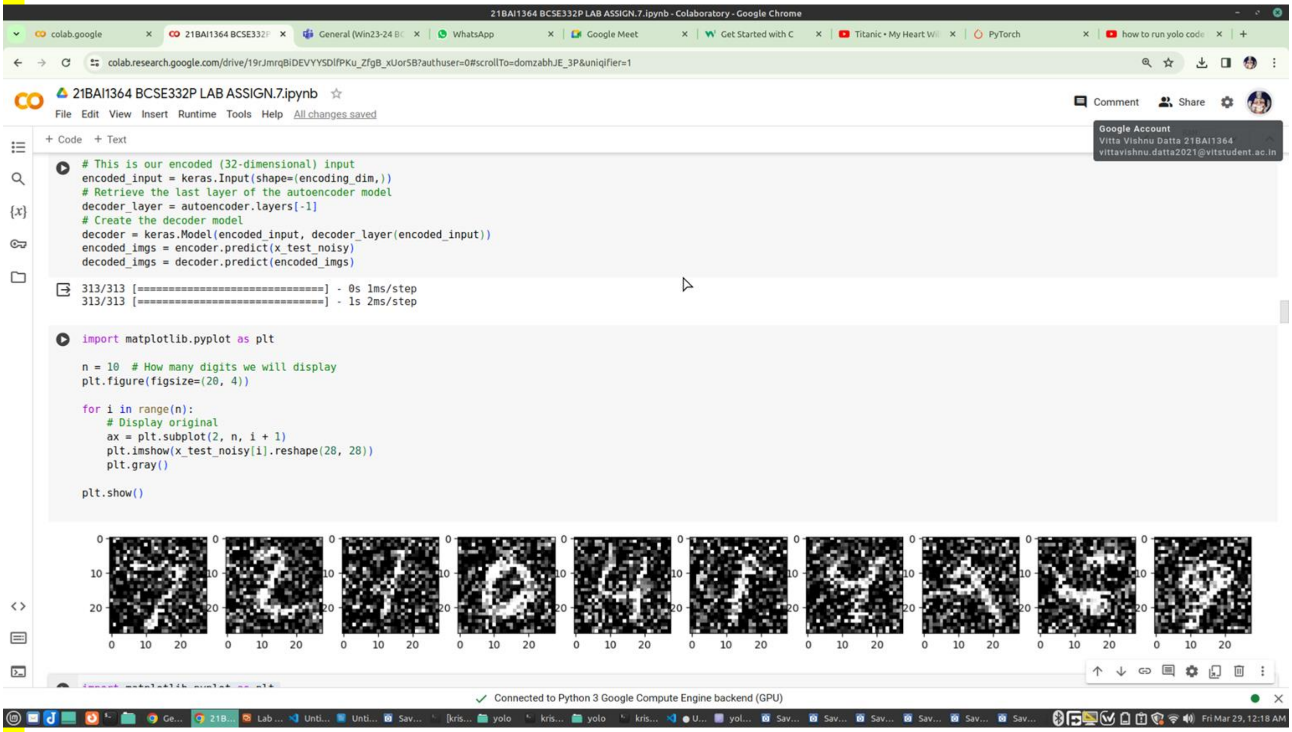
4


21BAI1364 BCSE332P LAB ASSIGN.7.ipynb
File Edit View Insert Runtime Tools Help All changes saved
from keras.datasets import mnist
import numpy as np
(x_train, _), (x_test, _) = 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:])))
noise_factor = 0.5
x_train_noisy = x_train + noise_factor * np.random.normal(loc=0.0, scale=1.0, size=x_train.shape)
x_test_noisy = x_test + noise_factor * np.random.normal(loc=0.0, scale=1.0, size=x_test.shape)
x_train_noisy = np.clip(x_train_noisy, 0., 1.)
x_test_noisy = np.clip(x_test_noisy, 0., 1.)
# Train your Keras autoencoder model (assuming you have the code)
# After training, obtain reconstructed images (replace with your actual logic)
decoded_imgs = autoencoder.predict(x_test_noisy)
# Visualization code (assuming x_test_noisy holds noisy images)
import matplotlib.pyplot as plt
n = 10
# ... (rest of the visualization code as before)
plt.show()
313/313 [==============================] - 1s 2ms/step
Connected to Python 3 Google Compute Engine backend (GPU)

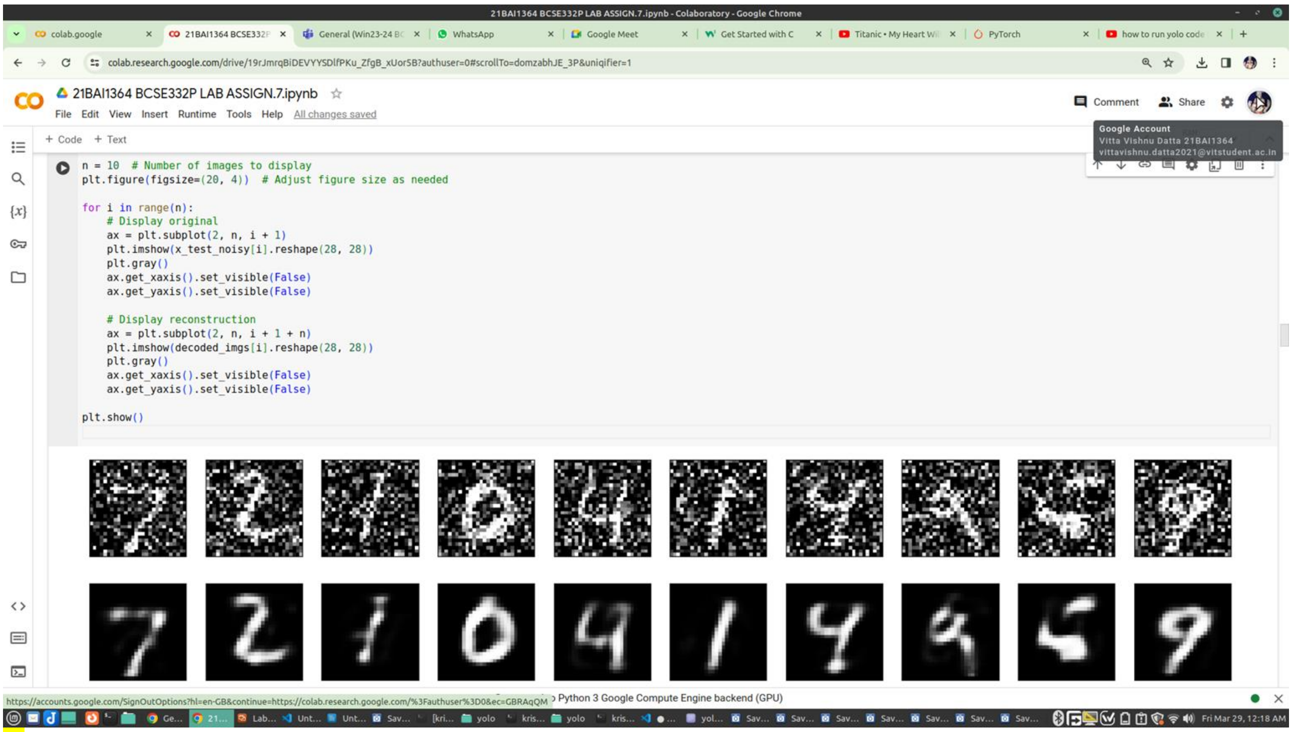
5


21BAI1364 BCSE332P LAB ASSIGN.7.ipynb - Colaboratory - Google Chrome
21BAI1364 BCSE332P LAB ASSIGN.7.ipynb
File Edit View Insert Runtime Tools Help All changes saved
from keras import layers
import tensorflow as tf
from keras.callbacks import TensorBoard
encoding_dim = 32 # This is our input image input_img = keras.Input(shape=(784,))
encoded = layers.Dense(encoding_dim, activation='relu')(input_img)
decoded = layers.Dense(784, activation='sigmoid')(encoded)
autoencoder = keras.Model(input_img, decoded)
autoencoder.compile(optimizer='adam', loss='binary_crossentropy')
autoencoder.fit(x_train_noisy, x_train, epochs=100, batch_size=128, shuffle=True,validation_data=(x_test_noisy, x_test),
)
encoder = keras.Model(input_img, encoded)
Epoch 50/100
469/469 [==============================] - 3s 5ms/step - loss: 0.1239 - val_loss: 0.1232
Epoch 51/100
469/469 [==============================] - 2s 5ms/step - loss: 0.1239 - val_loss: 0.1232
Epoch 52/100
469/469 [==============================] - 2s 4ms/step - loss: 0.1239 - val_loss: 0.1230
Epoch 53/100
469/469 [==============================] - 2s 3ms/step - loss: 0.1239 - val_loss: 0.1231
Epoch 54/100
469/469 [==============================] - 2s 3ms/step - loss: 0.1239 - val_loss: 0.1232
Epoch 55/100
469/469 [==============================] - 2s 3ms/step - loss: 0.1239 - val_loss: 0.1229
Epoch 56/100
469/469 [==============================] - 2s 5ms/step - loss: 0.1238 - val_loss: 0.1230
Epoch 57/100
469/469 [==============================] - 2s 4ms/step - loss: 0.1238 - val_loss: 0.1229
Epoch 58/100
469/469 [==============================] - 2s 4ms/step - loss: 0.1238 - val_loss: 0.1230
Epoch 59/100
469/469 [==============================] - 2s 3ms/step - loss: 0.1238 - val_loss: 0.1231
Epoch 60/100
469/469 [==============================] - 2s 4ms/step - loss: 0.1238 - val_loss: 0.1231
Epoch 61/100
469/469 [==============================] - 2s 3ms/step - loss: 0.1238 - val_loss: 0.1231
Epoch 62/100
469/469 [==============================] - 2s 3ms/step - loss: 0.1238 - val_loss: 0.1230
Connected to Python 3 Google Compute Engine backend (GPU)

6


21BAI1364 BCSE332P LAB ASSIGN.7.ipynb - Colaboratory - Google Chrome
21BAI1364 BCSE332P LAB ASSIGN.7.ipynb
File Edit View Insert Runtime Tools Help All changes saved
# This is our encoded (32-dimensional) input
encoded_input = keras.Input(shape=(encoding_dim,))
# Retrieve the last layer of the autoencoder model
decoder_layer = autoencoder.layers[-1]
# Create the decoder model
decoder = keras.Model(encoded_input, decoder_layer(encoded_input))
encoded_imgs = encoder.predict(x_test_noisy)
decoded_imgs = decoder.predict(encoded_imgs)
313/313 [==============================] - 0s 1ms/step
313/313 [==============================] - 1s 2ms/step
import matplotlib.pyplot as plt
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_noisy[i].reshape(28, 28))
    plt.gray()
plt.show()
Connected to Python 3 Google Compute Engine backend (GPU)

7




8

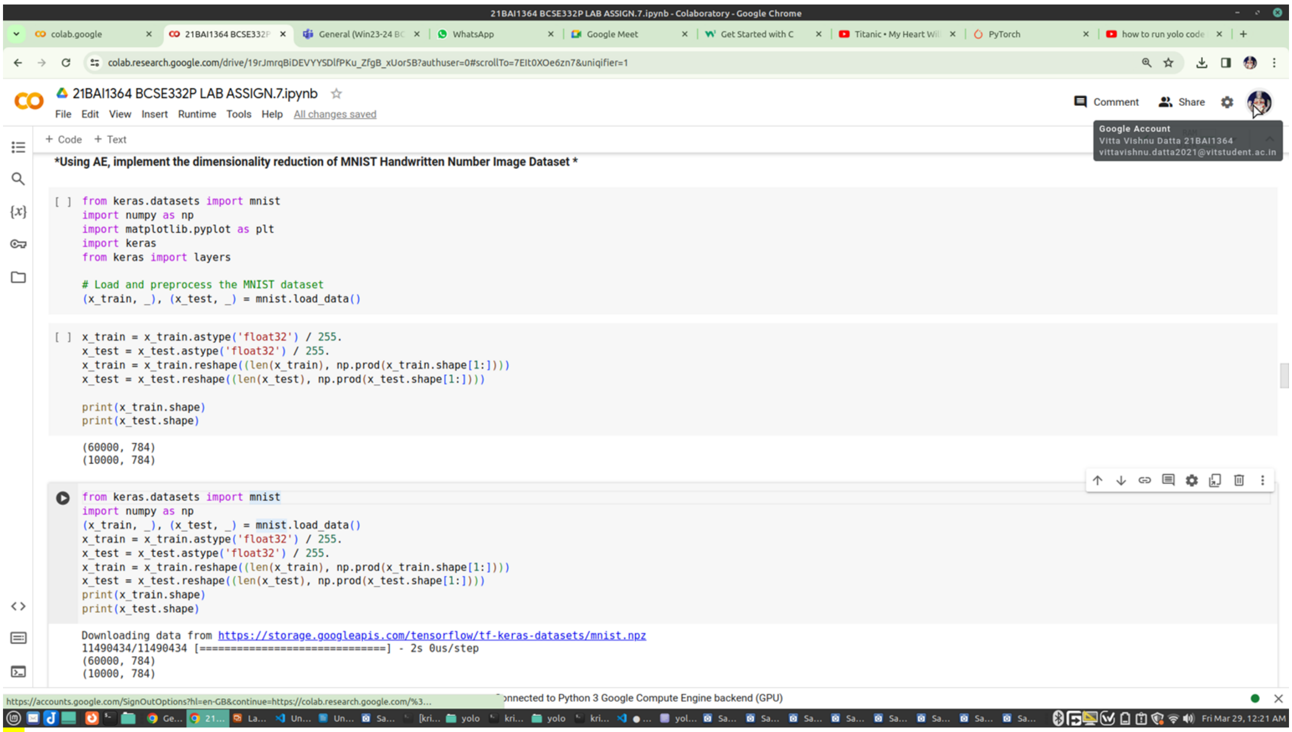
Using AE, implement the dimensionality reduction of MNIST Handwritten Number Image Dataset

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

# Load the MNIST dataset
(x_train, _), (x_test, _) = mnist.load_data()

# Normalize the pixel values to be between 0 and 1
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:])))

# Define the autoencoder model
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)

# Compile the autoencoder model
autoencoder.compile(optimizer='adam', loss='binary_crossentropy')

# Train the autoencoder model
autoencoder.fit(x_train, x_train,
epochs=50,
```

```
batch_size=256,
shuffle=True,
validation_data=(x_test, x_test))

# Create a separate encoder model
encoder = Model(input_img, encoded)

# Encode and decode some digits
encoded_imgs = encoder.predict(x_test)
decoded_imgs = autoencoder.predict(x_test)

# Plot the results
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 + 1 + n)
plt.imshow(decoded_imgs[i].reshape(28, 28))
plt.gray()
ax.get_xaxis().set_visible(False)
ax.get_yaxis().set_visible(False)
plt.show()
```

# LINK TO THE GOOGLE COLAB NOTEBOOK:

https://colab.research.google.com/drive/19rJmrqBiDEVYYSDlfPKu_ZfgB_xUor5B?authuser=0#scrollTo=5kdHD2bf_PIz&uniqifier=1



21BAI1364 LAB ASSIGN.7.ipynb

*Using AE, implement the dimensionality reduction of MNIST Handwritten Number Image Dataset *

```
from keras.datasets import mnist
import numpy as np
import matplotlib.pyplot as plt
import keras
from keras import layers

# Load and preprocess the MNIST dataset
(x_train, _), (x_test, _) = mnist.load_data()
```

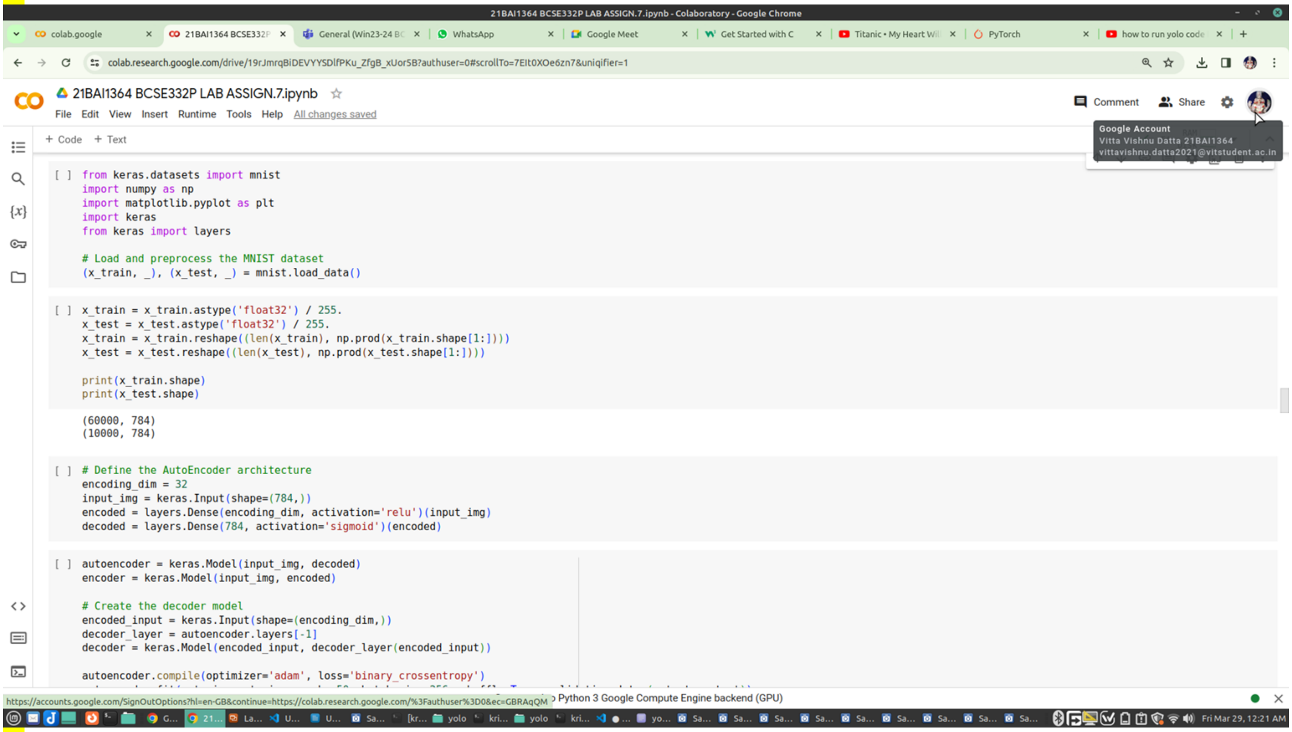
```
x_train = x_train.astype('float32') / 255.
x_test = x_test.astype('float32') / 255.
x_train = x_train.reshape((len(x_train), np.prod(x_train.shape[1:])))
x_test = x_test.reshape((len(x_test), np.prod(x_test.shape[1:])))

print(x_train.shape)
print(x_test.shape)
```

```
(60000, 784)
(10000, 784)
```

```
from keras.datasets import mnist
import numpy as np
(x_train, _), (x_test, _) = 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:])))
print(x_train.shape)
print(x_test.shape)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 [==============================] - 2s 0us/step
(60000, 784)
(10000, 784)
```

2



21BAI1364 LAB ASSIGN.7.ipynb

```
from keras.datasets import mnist
import numpy as np
import matplotlib.pyplot as plt
import keras
from keras import layers

# Load and preprocess the MNIST dataset
(x_train, _), (x_test, _) = mnist.load_data()
```

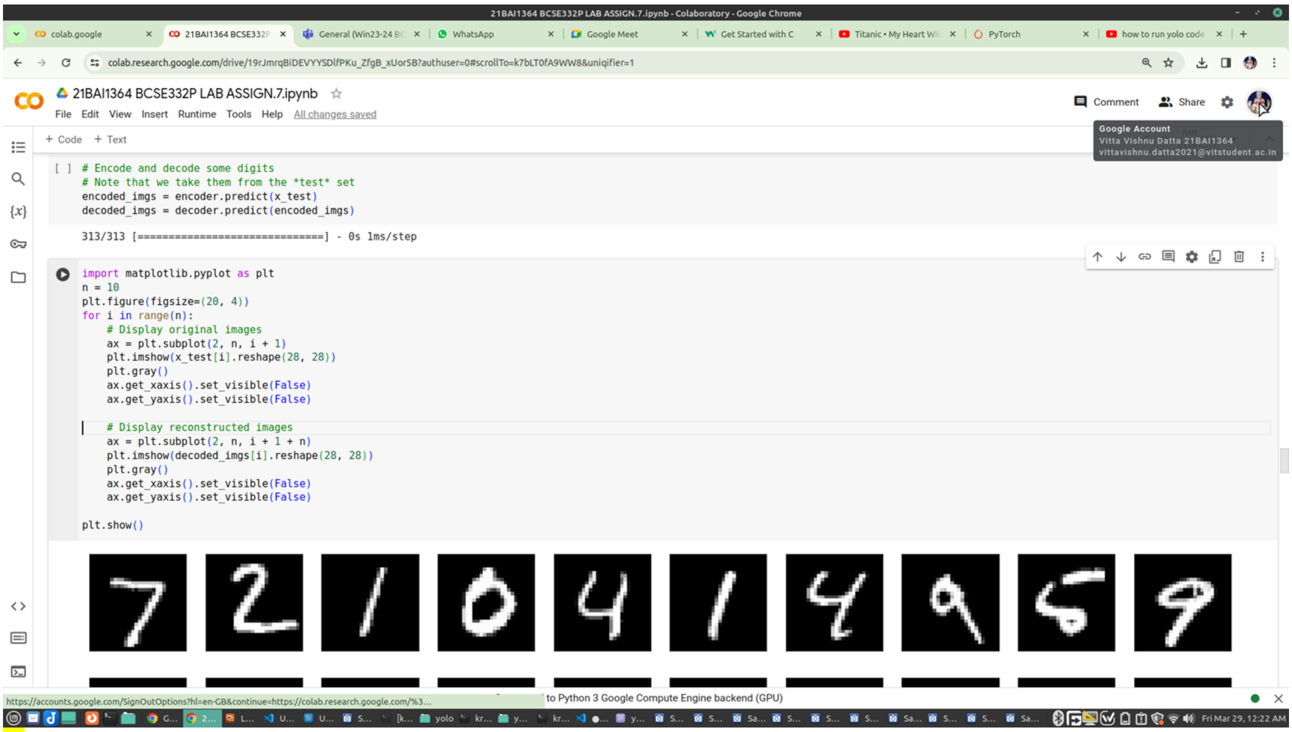
```
x_train = x_train.astype('float32') / 255.
x_test = x_test.astype('float32') / 255.
x_train = x_train.reshape((len(x_train), np.prod(x_train.shape[1:])))
x_test = x_test.reshape((len(x_test), np.prod(x_test.shape[1:])))

print(x_train.shape)
print(x_test.shape)
```

```
(60000, 784)
(10000, 784)
```

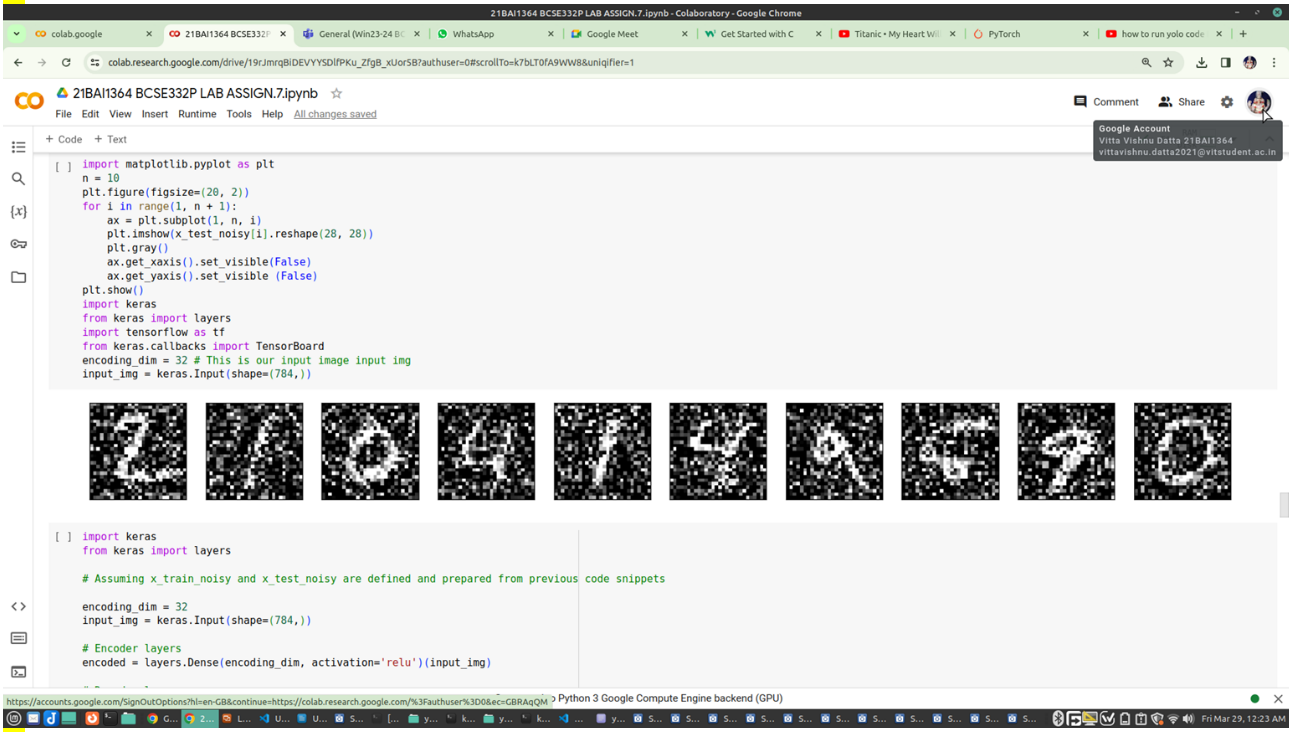
```
# Define the AutoEncoder architecture
encoding_dim = 32
input_img = keras.Input(shape=(784,))
encoded = layers.Dense(encoding_dim, activation='relu')(input_img)
decoded = layers.Dense(784, activation='sigmoid')(encoded)
```

```
autoencoder = keras.Model(input_img, decoded)
encoder = keras.Model(input_img, encoded)

# Create the decoder model
encoded_input = keras.Input(shape=(encoding_dim,))
decoder_layer = autoencoder.layers[-1]
decoder = keras.Model(encoded_input, decoder_layer(encoded_input))

autoencoder.compile(optimizer='adam', loss='binary_crossentropy')
```

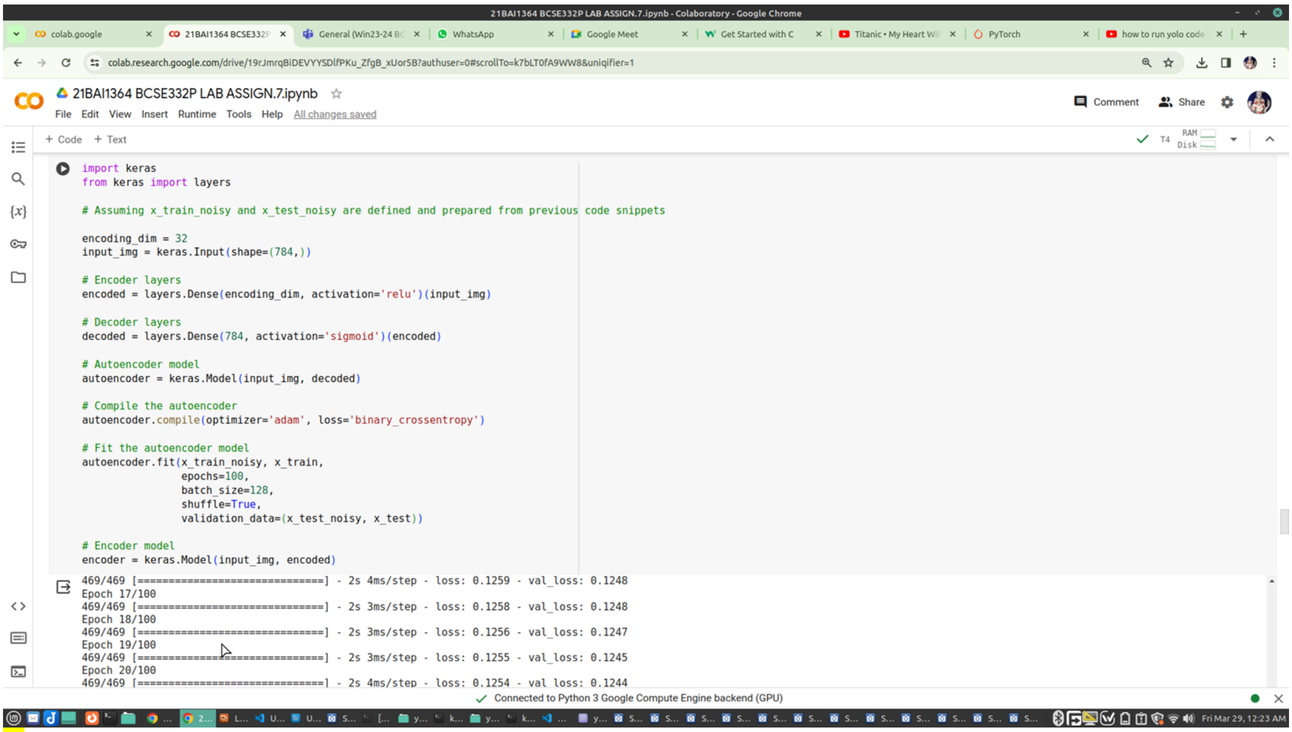
3


21BAI1364 BCSE332P LAB ASSIGN.7.ipynb
File Edit View Insert Runtime Tools Help All changes saved
# Encode and decode some digits
# Note that we take them from the *test* set
encoded_imgs = encoder.predict(x_test)
decoded_imgs = decoder.predict(encoded_imgs)
313/313 [==============================] - 0s 1ms/step
import matplotlib.pyplot as plt
n = 10
plt.figure(figsize=(20, 4))
for i in range(n):
# Display original images
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 reconstructed images
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)
plt.show()
to Python 3 Google Compute Engine backend (GPU)

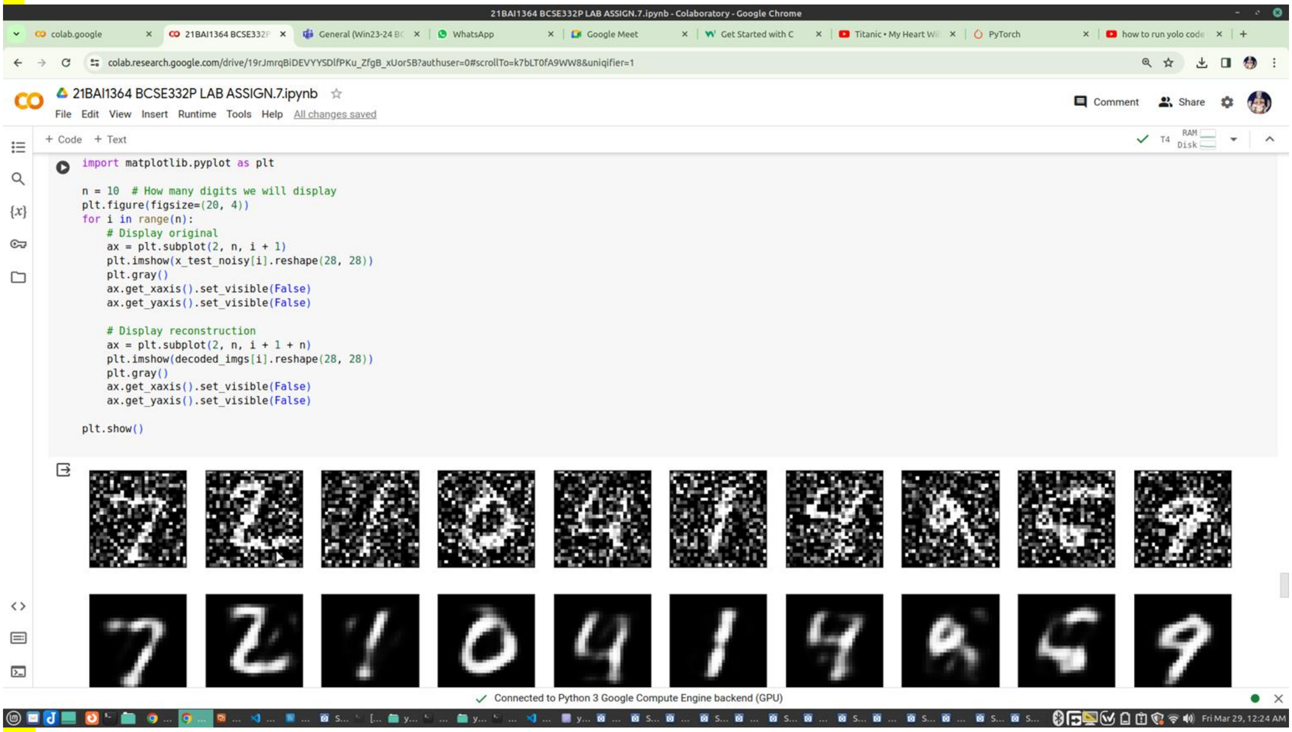
4


21BAI1364 BCSE332P LAB ASSIGN.7.ipynb
File Edit View Insert Runtime Tools Help All changes saved
import matplotlib.pyplot as plt
n = 10
plt.figure(figsize=(20, 2))
for i in range(1, n + 1):
ax = plt.subplot(1, n, i)
plt.imshow(x_test_noisy[i].reshape(28, 28))
plt.gray()
ax.get_xaxis().set_visible(False)
ax.get_yaxis().set_visible (False)
plt.show()
import keras
from keras import layers
import tensorflow as tf
from keras.callbacks import TensorBoard
encoding_dim = 32 # This is our input image input img
input_img = keras.Input(shape=(784,))
import keras
from keras import layers
# Assuming x_train_noisy and x_test_noisy are defined and prepared from previous code snippets
encoding_dim = 32
input_img = keras.Input(shape=(784,))
# Encoder layers
encoded = layers.Dense(encoding_dim, activation='relu')(input_img)
Python 3 Google Compute Engine backend (GPU)

5



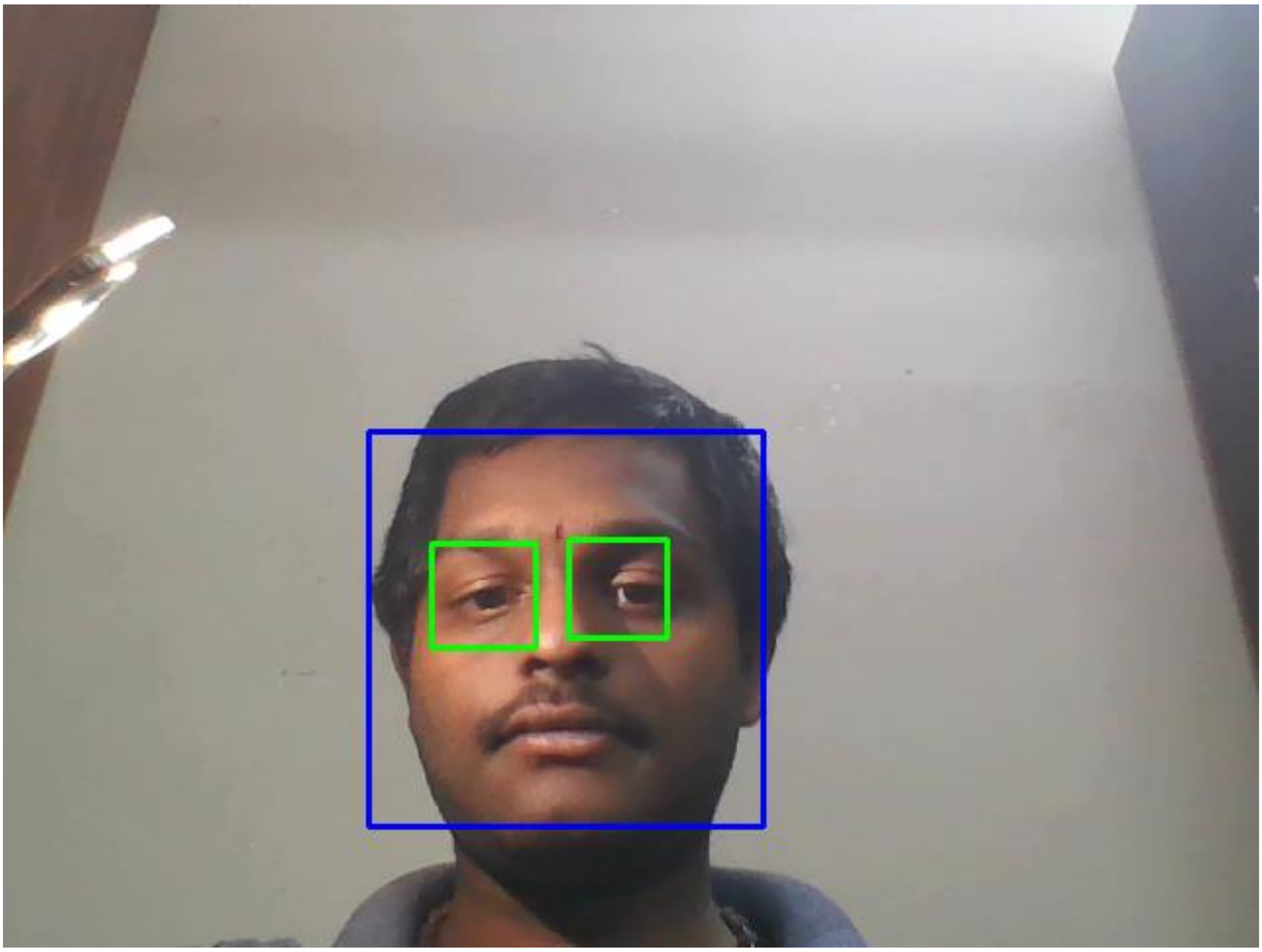

6



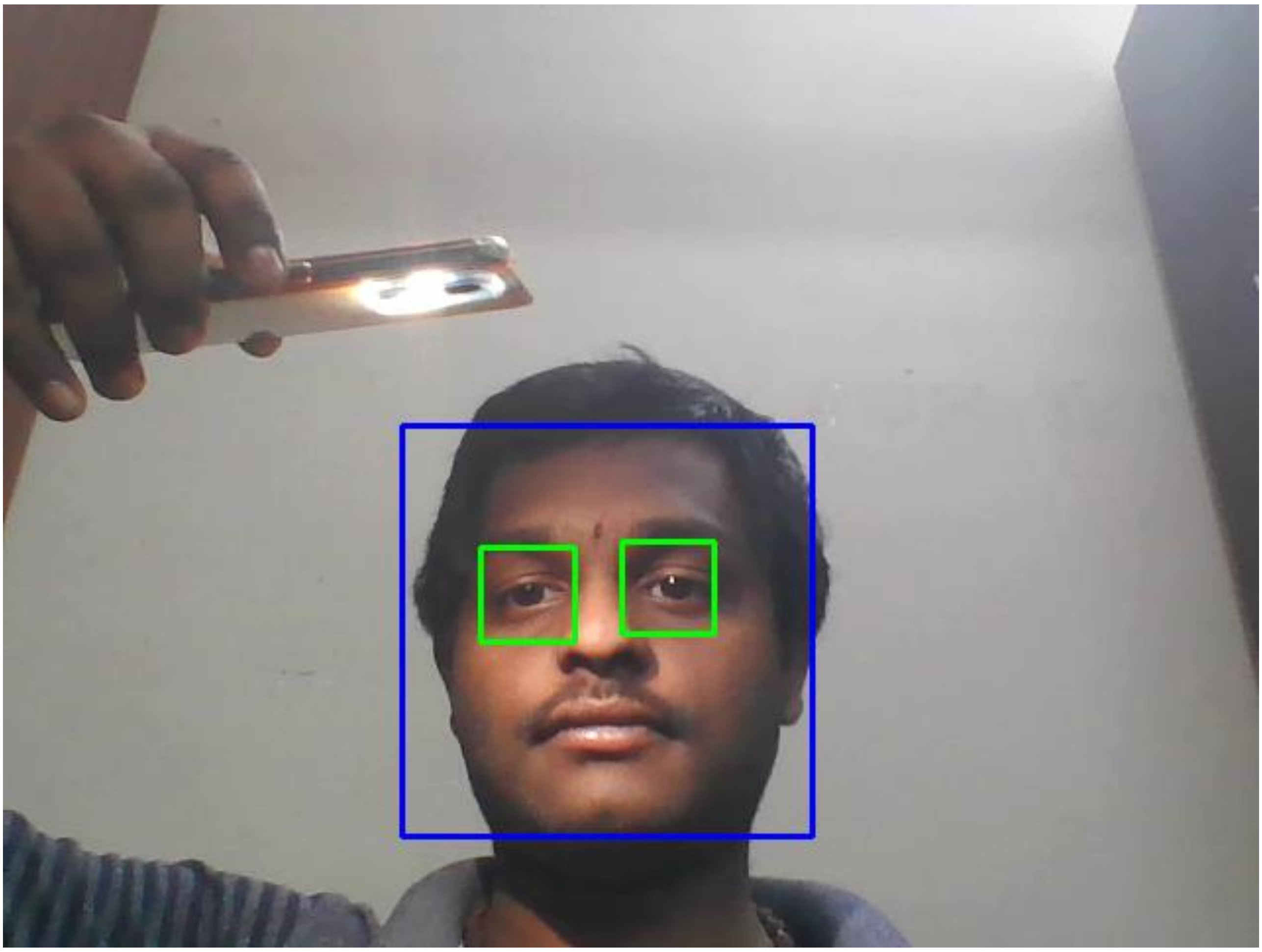

2.

2.Implement YOLO to recognize your Face and eyes from live camera

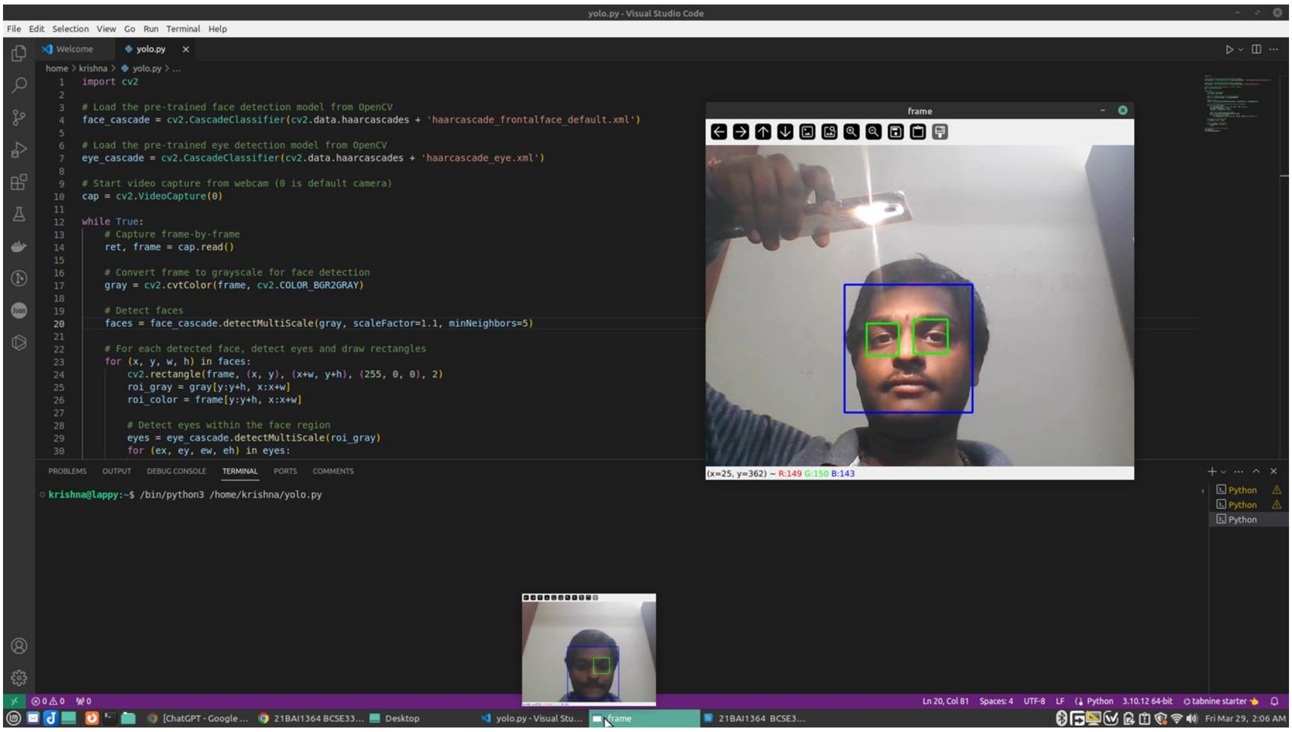
```
import cv2

# Load the pre-trained face detection model from OpenCV
face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_frontalface_default.xml')

# Load the pre-trained eye detection model from OpenCV
eye_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_eye.xml')

# Start video capture from webcam (0 is default camera)
cap = cv2.VideoCapture(0)

while True:
# Capture frame-by-frame
ret, frame = cap.read()

# Convert frame to grayscale for face detection
gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

# Detect faces
faces = face_cascade.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=5)

# For each detected face, detect eyes and draw rectangles
for (x, y, w, h) in faces:
cv2.rectangle(frame, (x, y), (x+w, y+h), (255, 0, 0), 2)
roi_gray = gray[y:y+h, x:x+w]
roi_color = frame[y:y+h, x:x+w]

# Detect eyes within the face region
eyes = eye_cascade.detectMultiScale(roi_gray)
for (ex, ey, ew, eh) in eyes:
cv2.rectangle(roi_color, (ex, ey), (ex+ew, ey+eh), (0, 255, 0), 2)

# Display the resulting frame
cv2.imshow('frame', frame)

# Exit loop on 'q' key press
if cv2.waitKey(1) == ord('q'):
break

# Release capture and close window
cap.release()
cv2.destroyAllWindows()
```



2.



3.


yolo.py - Visual Studio Code
File Edit Selection View Go Run Terminal Help
Welcome
yolo.py
home > krishna > yolo.py > ...
import cv2
# Load the pre-trained face detection model from OpenCV
face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_frontalface_default.xml')
# Load the pre-trained eye detection model from OpenCV
eye_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_eye.xml')
# Start video capture from webcam (0 is default camera)
cap = cv2.VideoCapture(0)
while True:
# Capture frame-by-frame
ret, frame = cap.read()
# Convert frame to grayscale for face detection
gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
# Detect faces
faces = face_cascade.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=5)
# For each detected face, detect eyes and draw rectangles
for (x, y, w, h) in faces:
cv2.rectangle(frame, (x, y), (x+w, y+h), (255, 0, 0), 2)
roi_gray = gray[y:y+h, x:x+w]
roi_color = frame[y:y+h, x:x+w]
# Detect eyes within the face region
eyes = eye_cascade.detectMultiScale(roi_gray)
for (ex, ey, ew, eh) in eyes:
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS COMMENTS
krishna@lappy:~$ /bin/python3 /home/krishna/yolo.py
frame

************THANK YOU**************