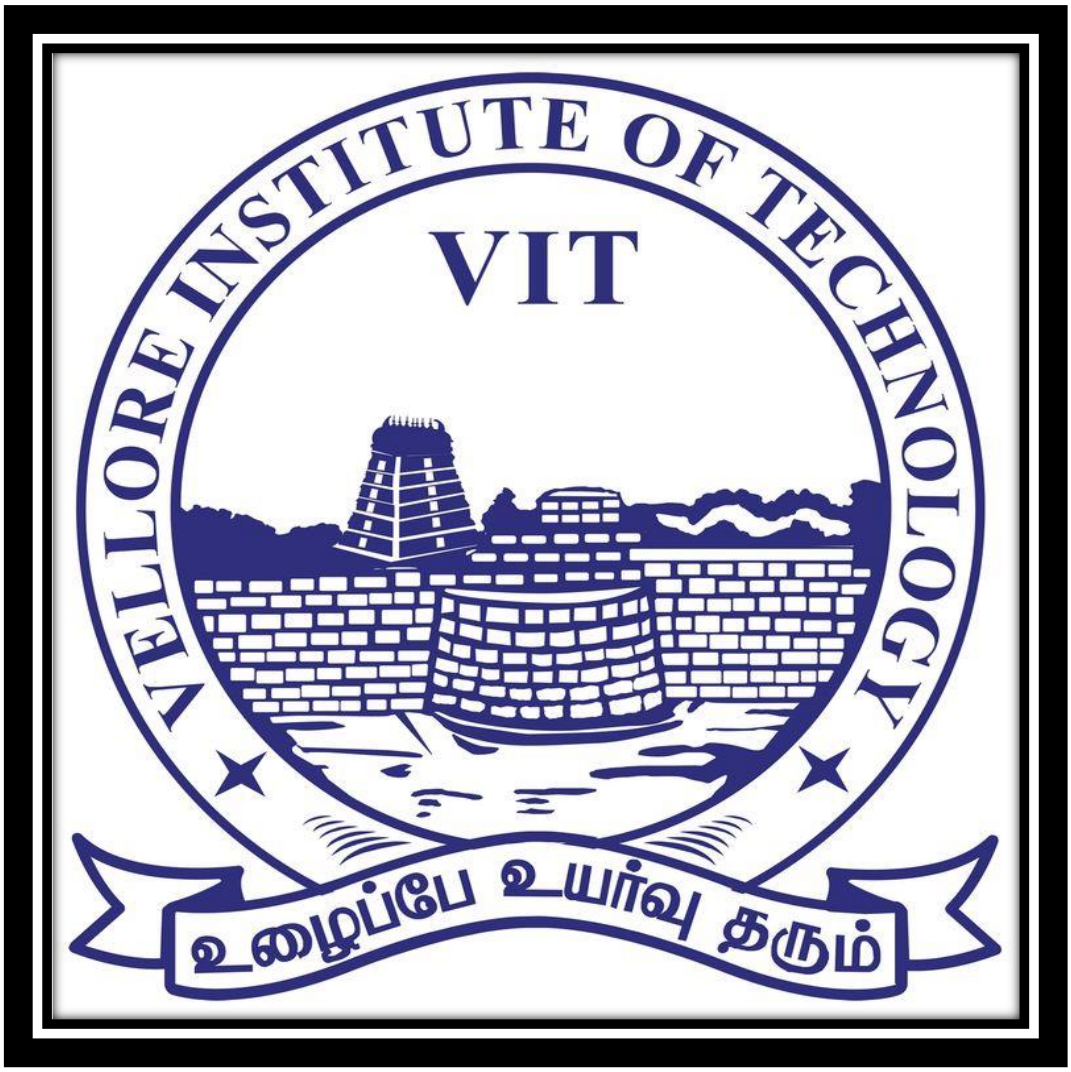
VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

CHENNAI



VELLORE INSTITUTE OF TECHNOLOGY

VIT – CHENNAI CAMPUS

DEEP LEARNING LAB

LAB ASSIGNMENT – 3

FACULTY : DR. HARINI. S

VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

CHENNAI

BCSE332P – DL Lab

Lab 3 – DL Lab Assignment

1. AUTO-Implement 3 different neural networks as listed below
    - 2 layers. 1st layer: ReLU (5 perceptrons),
    - 2nd Layer: Sigmoid (1 perceptron)
    - In the above layers, both sigmoid
    - 4 layers with the first three layers: Sigmoid (3) and the last layer: sigmoid (1)

CODE:

```
import numpy as np
import pandas as pd
import tensorflow as tf
from tensorflow.keras.callbacks import Callback
from tensorflow.keras.layers import Dense, Input
from tensorflow.keras import Sequential
from tensorflow.keras.initializers import RandomNormal
from tensorflow.keras.optimizers import Adam
import matplotlib.pyplot as plt
from sklearn.datasets import make_circles
from sklearn.metrics import accuracy_score

tf.random.set_seed(42)
np.random.seed(42)

X, Y = make_circles(n_samples=1000, factor = 0.5, noise = 0.1)
plt.figure(figsize=(8,6))
plt.scatter(X[:, 0], X[:, 1], c=Y)
plt.show()

# First Model
model = Sequential()
```

```
model.add(Dense(5, activation='relu', input_dim=X.shape[1]))
model.add(Dense(1, activation ='sigmoid'))

model.compile(optimizer=Adam(learning_rate=0.01),
loss='binary_crossentropy',metrics='accuracy')
model.fit(X, Y, batch_size=32, epochs = 100, verbose = 1)
# Second Model

model2 = Sequential()
model2.add(Dense(5, activation='sigmoid', input_dim=X.shape[1]))
model2.add(Dense(1, activation ='sigmoid'))

model2.compile(optimizer=Adam(learning_rate=0.01),
loss='binary_crossentropy',metrics='accuracy')
model2.fit(X, Y, batch_size=32, epochs = 100, verbose = 1)
model2.evaluate(X,Y)
model3 = Sequential()
model3.add(Dense(3, activation='sigmoid', input_dim=X.shape[1]))
model3.add(Dense(3, activation ='sigmoid'))
model3.add(Dense(3, activation ='sigmoid'))
model3.add(Dense(1, activation ='sigmoid'))

model3.compile(optimizer=Adam(learning_rate=0.01),
loss='binary_crossentropy',metrics='accuracy')
model3.fit(X, Y, batch_size=32, epochs = 100, verbose = 1)
model3.evaluate(X, Y)
dataset = "/content/creditcard.csv"

df = pd.read_csv(dataset)

features = df.drop(columns='Class')
target=df['Class']
features.head()
target.head()
from sklearn.model_selection import train_test_split

X_train, X_test, Y_train, Y_test = train_test_split(features,
target,test_size=0.2, random_state=42)
X_train.shape


X_test.shape
```

```
from sklearn.preprocessing import MinMaxScaler
scaler=MinMaxScaler()

scaled_features_train = scaler.fit_transform(X_train)
scaled_features_test = scaler.fit_transform(X_test)
```

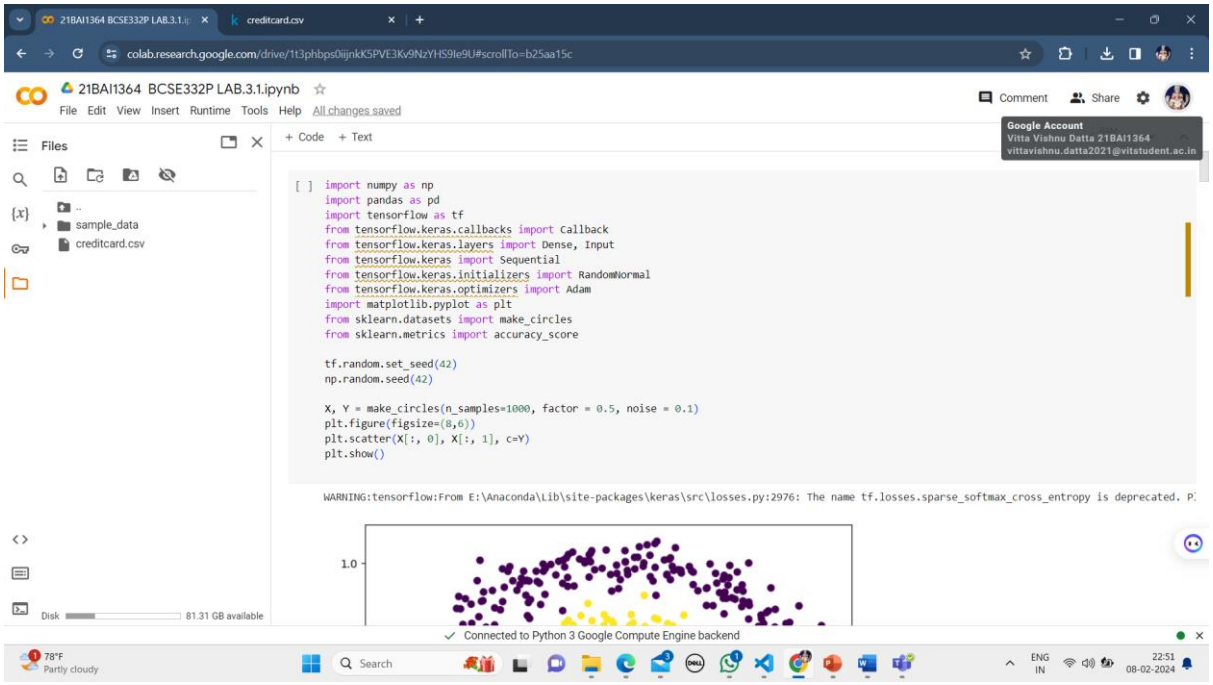
```
scaled_feature_train_array = np.array(scaled_features_train)
scaled_feature_test_array = np.array(scaled_features_test)
target_train_array= np.array(Y_train)
target_test_array= np.array(Y_test)
scaled_feature_train_array.shape
# First Model
model_p = Sequential()
model_p.add(Dense(5, activation='relu',
input_dim=scaled_feature_train_array.shape[1]))
model_p.add(Dense(1, activation ='sigmoid'))

model_p.compile(optimizer=Adam(learning_rate=0.01),
loss='binary_crossentropy',metrics='accuracy')
model_p.fit(scaled_feature_train_array, target_train_array,
batch_size=32, epochs = 5, verbose = 1)
model_p.evaluate(scaled_feature_train_array,target_train_array)
predictions = model_p.predict(scaled_features_test_array)
binary_predictions = (predictions > 0.5).astype(int)

predicted_labels = np.squeeze(binary_predictions)
true_labels = target_test_array

result_comparison = pd.DataFrame({'True Labels': true_labels,
'Predicted Labels': predicted_labels})
print(result_comparison)
# Second Model

model2_p = Sequential()
model2_p.add(Dense(5, activation='sigmoid',
input_dim=scaled_feature_train_array.shape[1]))
model2_p.add(Dense(1, activation ='sigmoid'))

model2_p.compile(optimizer=Adam(learning_rate=0.01),
loss='binary_crossentropy',metrics='accuracy')
```

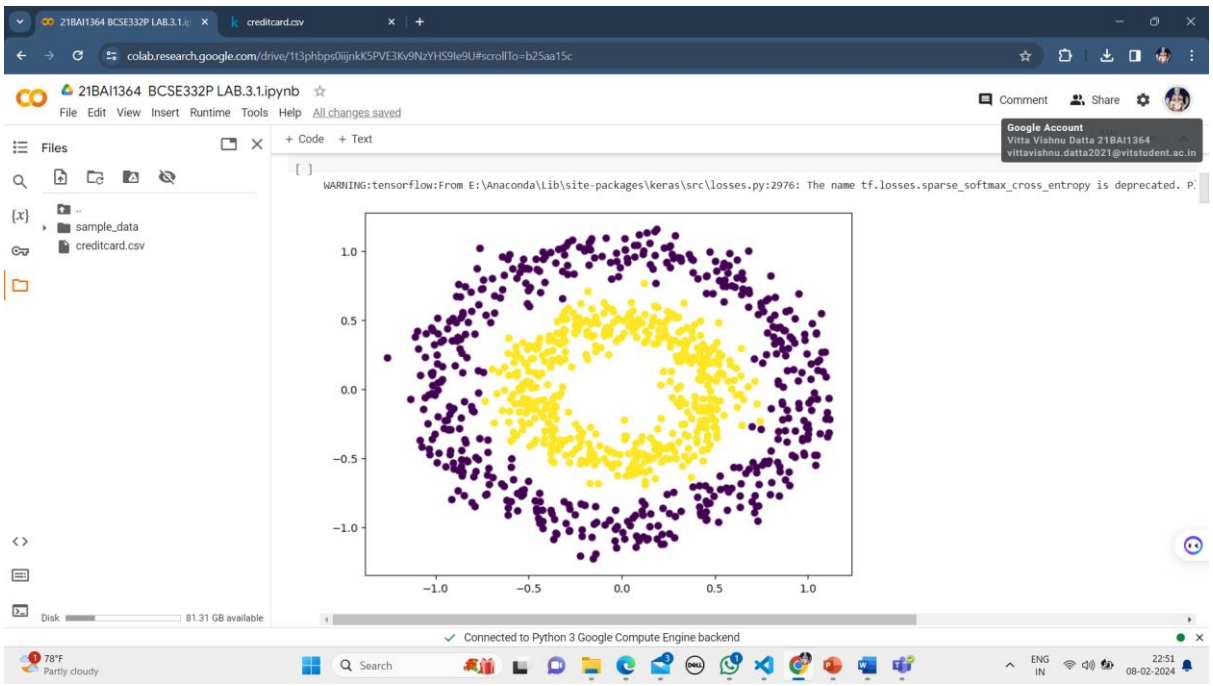
```
model2_p.fit(scaled_feature_train_array, target_train_array,
batch_size=32, epochs = 5, verbose = 1)
model2_p.evaluate(scaled_feature_train_array,target_train_array)
predictions_2 = model2_p.predict(scaled_feature_test_array)
binary_predictions_2 = (predictions_2 > 0.5).astype(int)

predicted_labels_2 = np.squeeze(binary_predictions_2)
true_labels = target_test_array

result_comparison_2 = pd.DataFrame({'True Labels': true_labels,
'Predicted Labels': predicted_labels})
print(result_comparison_2)
model3_p = Sequential()
model3_p.add(Dense(3, activation='sigmoid',
input_dim=scaled_feature_train_array.shape[1]))
model3_p.add(Dense(3, activation ='sigmoid'))
model3_p.add(Dense(3, activation ='sigmoid'))
model3_p.add(Dense(1, activation ='sigmoid'))

model3_p.compile(optimizer=Adam(learning_rate=0.01),
loss='binary_crossentropy',metrics='accuracy')
model3_p.fit(scaled_feature_train_array, target_train_array,
batch_size=32, epochs = 5, verbose = 1)
model3_p.evaluate(scaled_feature_train_array, target_train_array)
predictions_3 = model3_p.predict(scaled_feature_test_array)
binary_predictions_3 = (predictions_3 > 0.5).astype(int)

predicted_labels_3 = np.squeeze(binary_predictions_3)
true_labels = target_test_array

result_comparison_3 = pd.DataFrame({'True Labels': true_labels,
'Predicted Labels': predicted_labels})
print(result_comparison_3)
```

LINK TO THE GOOGLE COLAB NOTEBOOK:

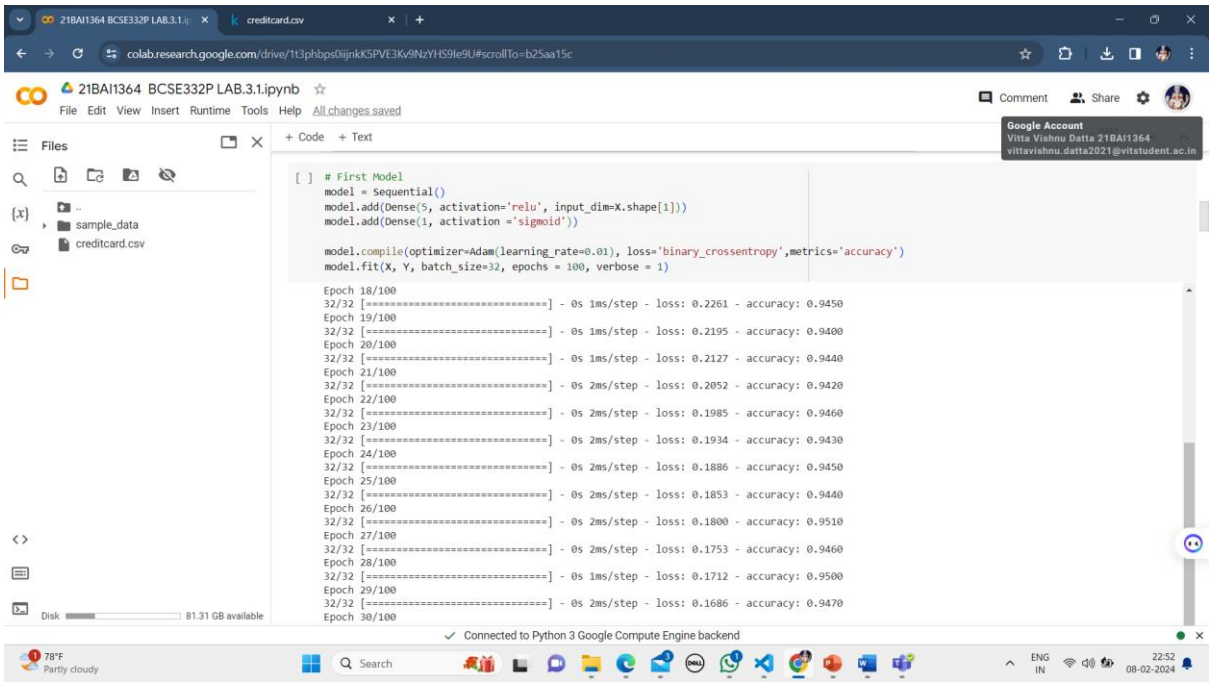
https://colab.research.google.com/drive/1t3phbps0iijnkK5PVE3Kv9NzYHS9Ie9U

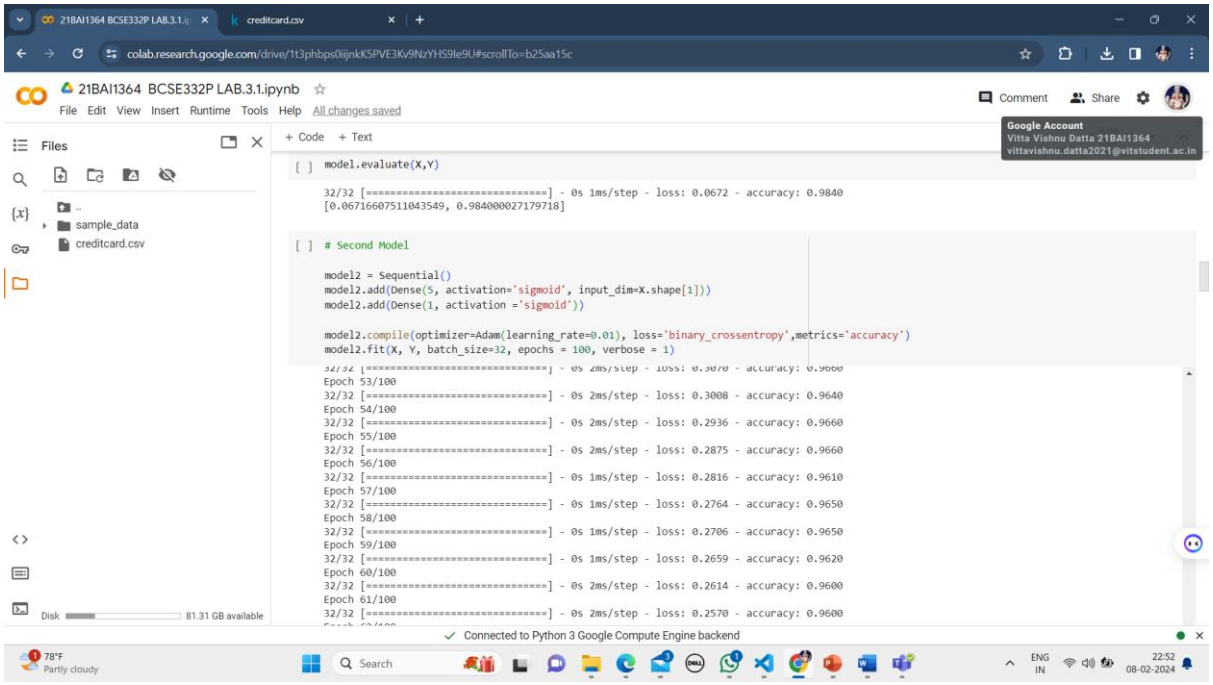
OUTPUT OF THE CODE SNIPPET:





VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)
CHENNAI



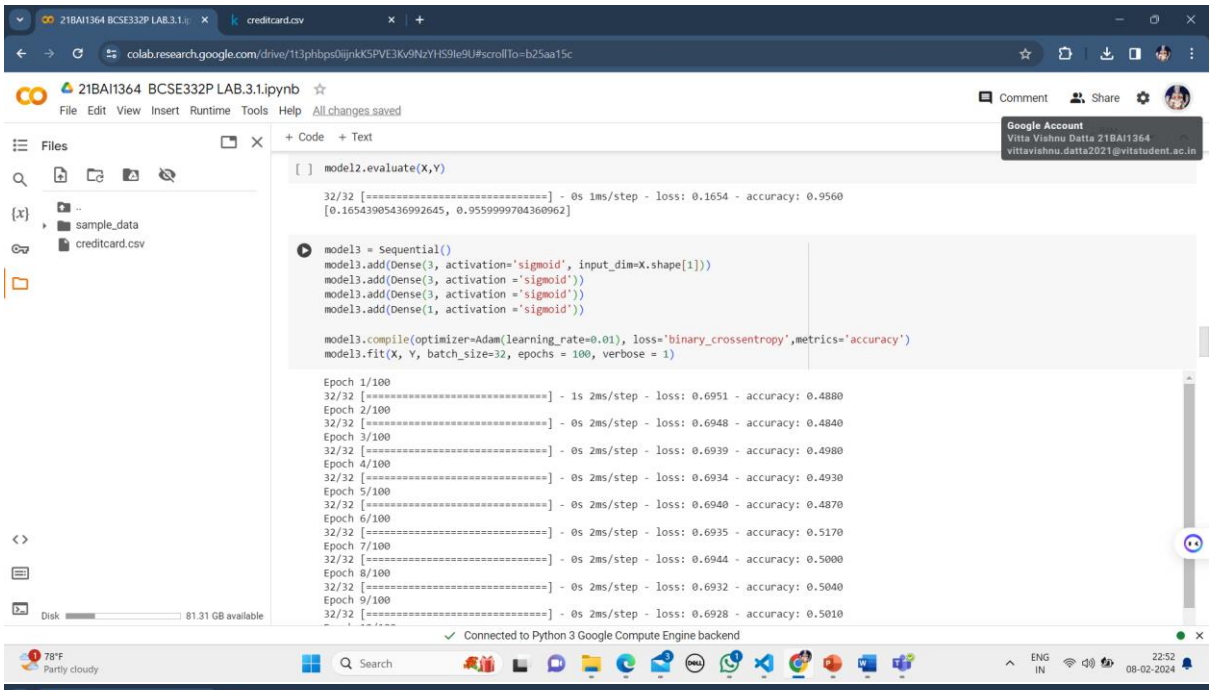


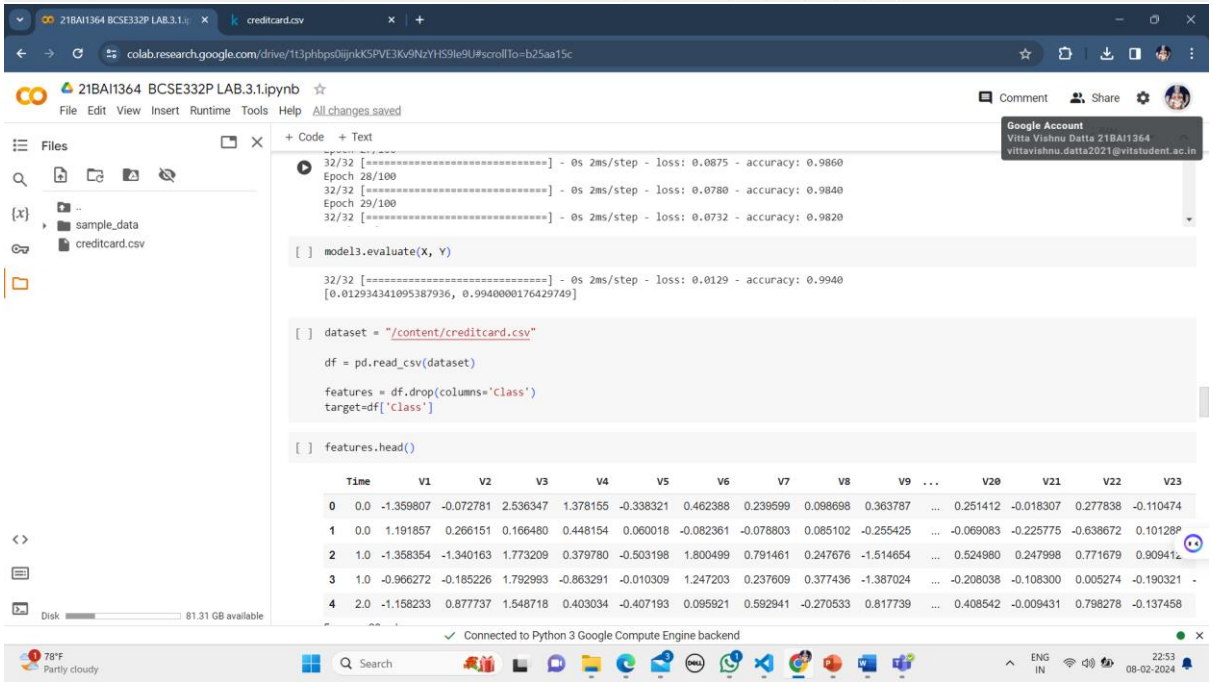
VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

CHENNAI

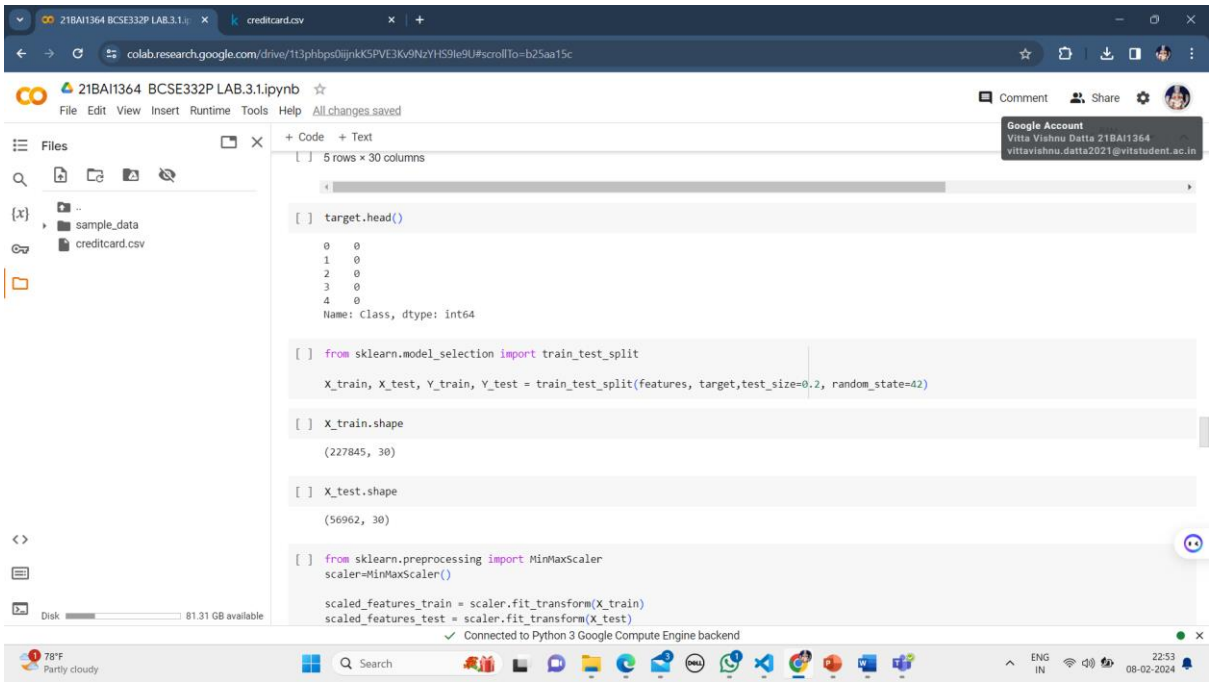


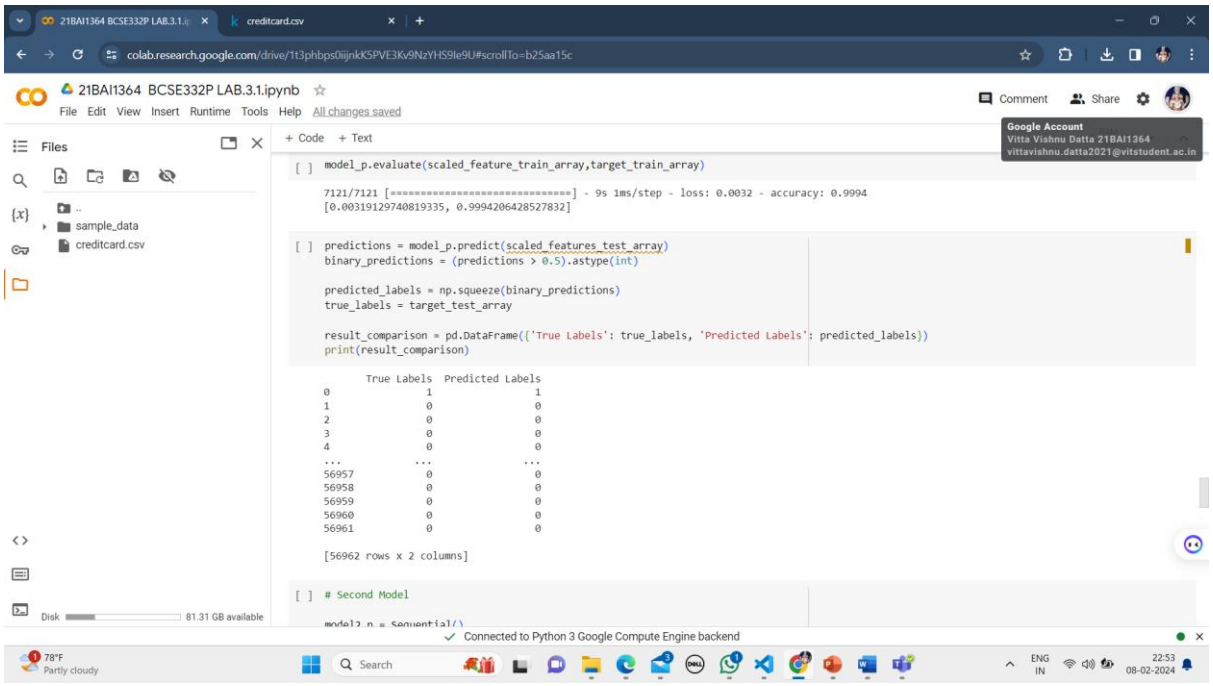


Vellore Institute of Technology

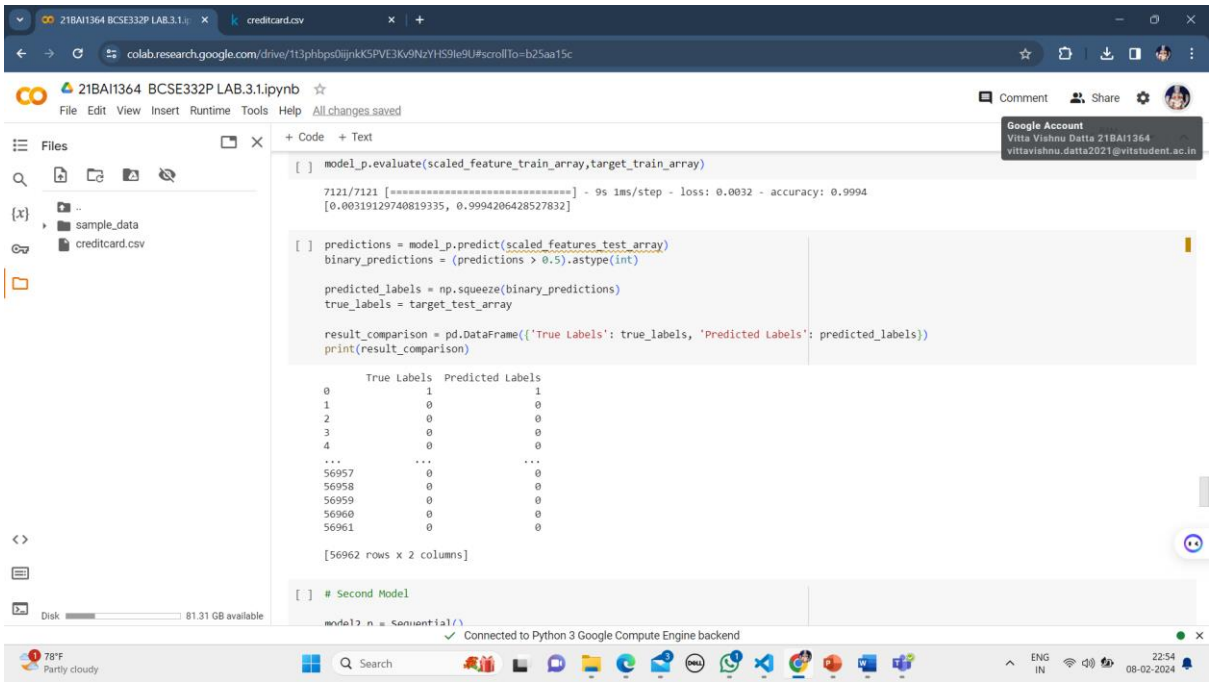
(Deemed to be University under section 3 of UGC Act, 1956)

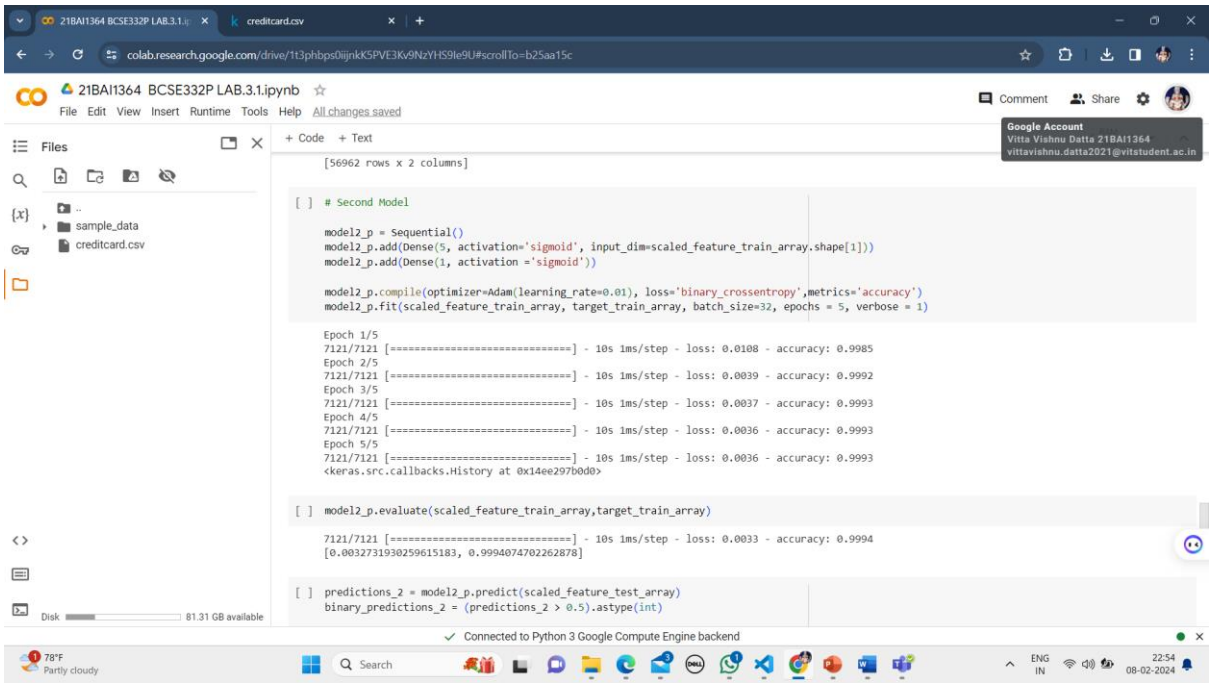
CHENNAI

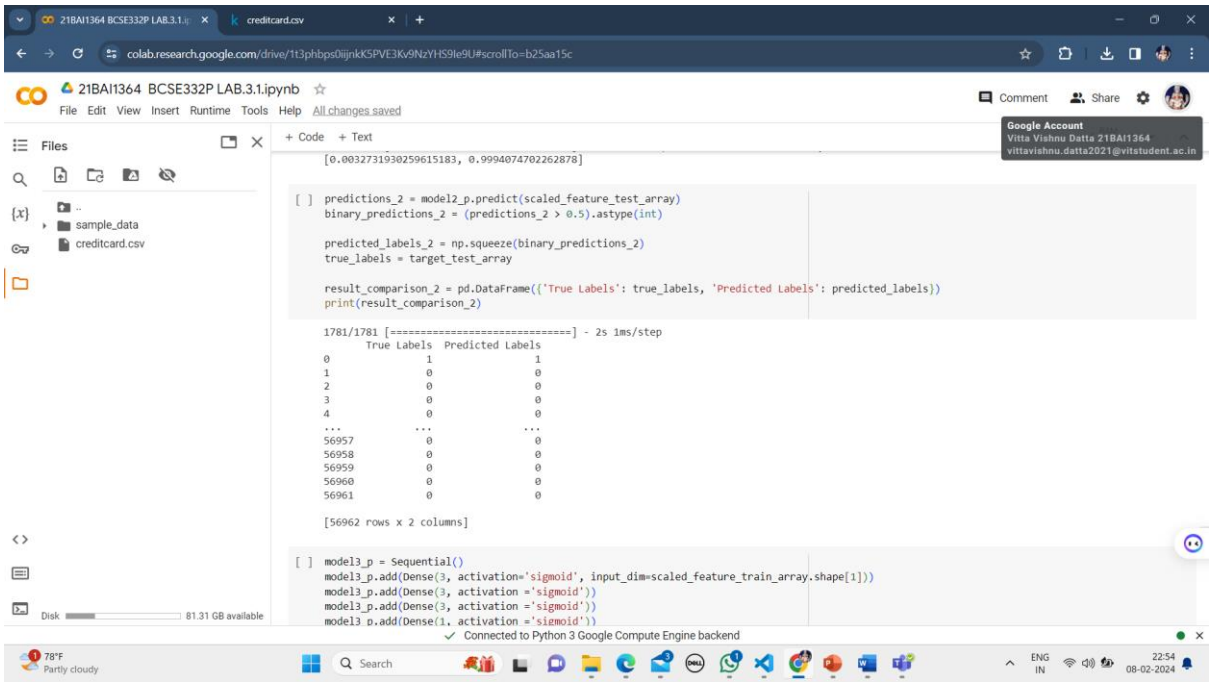


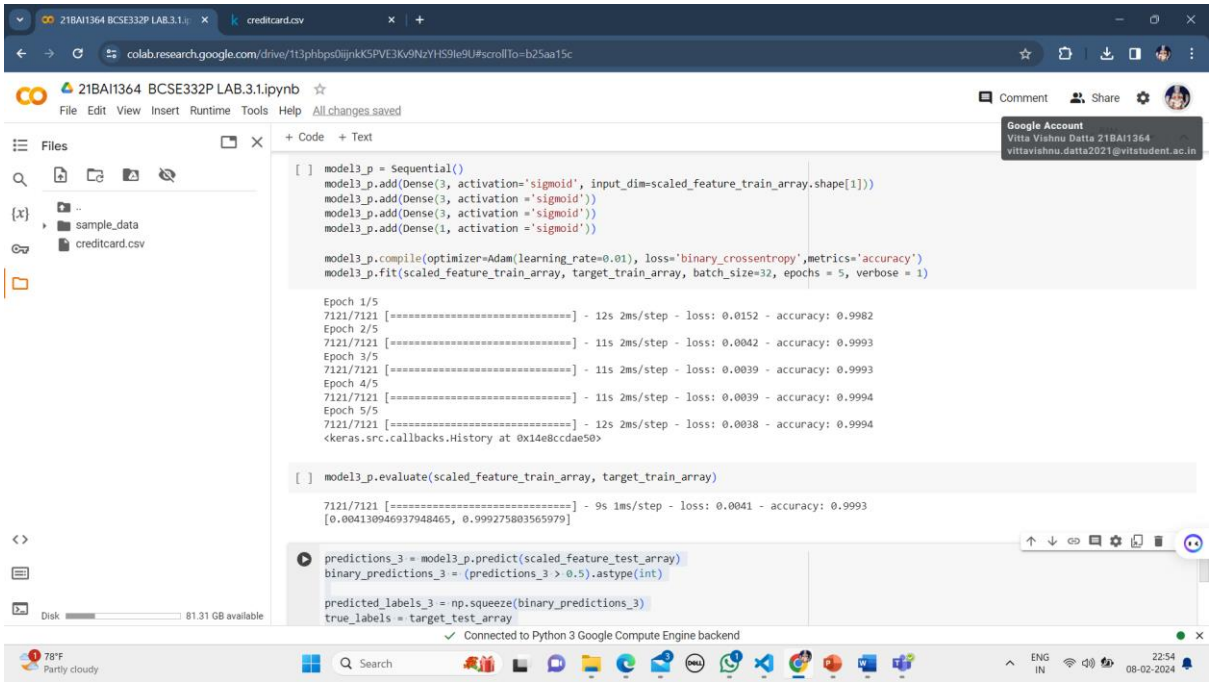


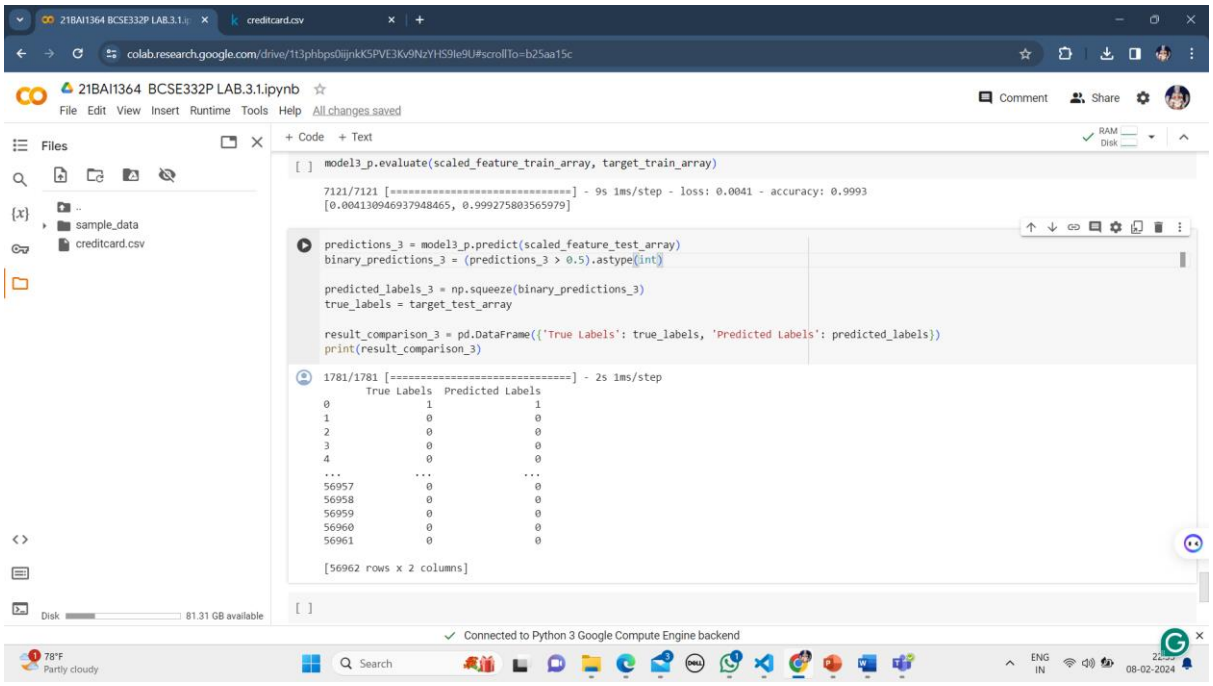
VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)
CHENNAI






21BAI1364 BCSE332P LAB.3.1.ipynb
creditcard.csv
colab.research.google.com/drive/1t3phbps0iijnkK5PVE3Kv9NzYHS9Ie9U#scrollTo=b25aa15c
21BAI1364 BCSE332P LAB.3.1.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment
Share
Google Account
Vitta Vishnu Datta 21BAI1364
vittavishnu.datta2021@vitstudent.ac.in
Files
+ Code + Text
sample_data
creditcard.csv
Disk 81.31 GB available
[0.0032731930259615183, 0.9994074702262878]
[ ] predictions_2 = model2_p.predict(scaled_feature_test_array)
binary_predictions_2 = (predictions_2 > 0.5).astype(int)
predicted_labels_2 = np.squeeze(binary_predictions_2)
true_labels = target_test_array
result_comparison_2 = pd.DataFrame({'True Labels': true_labels, 'Predicted Labels': predicted_labels})
print(result_comparison_2)
1781/1781 [==============================] - 2s 1ms/step
True Labels Predicted Labels
0 1 1
1 0 0
2 0 0
3 0 0
4 0 0
... ...
56957 0 0
56958 0 0
56959 0 0
56960 0 0
56961 0 0
[56962 rows x 2 columns]
[ ] model3_p = Sequential()
model3_p.add(Dense(3, activation='sigmoid', input_dim=scaled_feature_train_array.shape[1]))
model3_p.add(Dense(3, activation ='sigmoid'))
model3_p.add(Dense(3, activation ='sigmoid'))
model3_p.add(Dense(1, activation ='sigmoid'))
Connected to Python 3 Google Compute Engine backend
78°F
Partly cloudy
Search
ENG
IN
22:54
08-02-2024


21BAI1364 BCSE332P LAB.3.1.ipynb
creditcard.csv
colab.research.google.com/drive/1t3phbps0iijnkK5PVE3Kv9NzYHS9Ie9U#scrollTo=b25aa15c
21BAI1364 BCSE332P LAB.3.1.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment
Share
Google Account
Vitta Vishnu Datta 21BAI1364
vittavishnu.datta2021@vitstudent.ac.in
Files
+ Code + Text
sample_data
creditcard.csv
Disk 81.31 GB available
[ ] model3_p = Sequential()
model3_p.add(Dense(3, activation='sigmoid', input_dim=scaled_feature_train_array.shape[1]))
model3_p.add(Dense(3, activation ='sigmoid'))
model3_p.add(Dense(3, activation ='sigmoid'))
model3_p.add(Dense(1, activation ='sigmoid'))
model3_p.compile(optimizer=Adam(learning_rate=0.01), loss='binary_crossentropy',metrics='accuracy')
model3_p.fit(scaled_feature_train_array, target_train_array, batch_size=32, epochs = 5, verbose = 1)
Epoch 1/5
7121/7121 [==============================] - 12s 2ms/step - loss: 0.0152 - accuracy: 0.9982
Epoch 2/5
7121/7121 [==============================] - 11s 2ms/step - loss: 0.0042 - accuracy: 0.9993
Epoch 3/5
7121/7121 [==============================] - 11s 2ms/step - loss: 0.0039 - accuracy: 0.9993
Epoch 4/5
7121/7121 [==============================] - 11s 2ms/step - loss: 0.0039 - accuracy: 0.9994
Epoch 5/5
7121/7121 [==============================] - 12s 2ms/step - loss: 0.0038 - accuracy: 0.9994
<keras.src.callbacks.History at 0x14e8ccdae50>
[ ] model3_p.evaluate(scaled_feature_train_array, target_train_array)
7121/7121 [==============================] - 9s 1ms/step - loss: 0.0041 - accuracy: 0.9993
[0.004130946937948465, 0.999275803565979]
predictions_3 = model3_p.predict(scaled_feature_test_array)
binary_predictions_3 = (predictions_3 > 0.5).astype(int)
predicted_labels_3 = np.squeeze(binary_predictions_3)
true_labels = target_test_array
Connected to Python 3 Google Compute Engine backend
78°F
Partly cloudy
Search
ENG
IN
22:54
08-02-2024



2. Change the activation functions and implement the code

CODE:

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.callbacks import Callback
from tensorflow.keras.layers import Dense, Input
from tensorflow.keras import Sequential
from tensorflow.keras.initializers import RandomNormal
import matplotlib.pyplot as plt
from sklearn.datasets import make_circles
from sklearn.metrics import accuracy_score

tf.random.set_seed(42)
np.random.seed(42)

# Make data: Two circles on x-y plane as a classification problem
X, y = make_circles(n_samples=1000, factor=0.5, noise=0.1)
plt.figure(figsize=(8,6))
```

```
plt.scatter(X[:,0], X[:,1], c=y)
plt.show()
# Test performance with 3-layer binary classification network
model = Sequential([
    Input(shape=(2,)),
    Dense(5, "relu"),
    Dense(1, "sigmoid")
])
model.compile(optimizer="adam", loss="binary_crossentropy",
metrics=["acc"])
model.fit(X, y, batch_size=32, epochs=100, verbose=0)
print(model.evaluate(X,y))


# Test performance with 3-layer network with sigmoid activation
model = Sequential([
    Input(shape=(2,)),
    Dense(5, "tanh"),
    Dense(1, "sigmoid")
])
model.compile(optimizer="adam", loss="binary_crossentropy",
metrics=["acc"])
model.fit(X, y, batch_size=32, epochs=100, verbose=0)
print(model.evaluate(X,y))


# Test performance with 5-layer network with sigmoid activation
model = Sequential([
    Input(shape=(2,)),
    Dense(5, "tanh"),
    Dense(5, "tanh"),
    Dense(5, "tanh"),
    Dense(1, "tanh")
])
model.compile(optimizer="adam", loss="binary_crossentropy",
metrics=["acc"])
model.fit(X, y, batch_size=32, epochs=100, verbose=0)
print(model.evaluate(X,y))

# Test performance with 5-layer network with sigmoid activation
model = Sequential([
    Input(shape=(2,)),
    Dense(5, "relu"),
```

```
    Dense(5, "relu"),
    Dense(5, "relu"),
    Dense(1, "relu")
])
model.compile(optimizer="adam", loss="binary_crossentropy",
metrics=["acc"])
model.fit(X, y, batch_size=32, epochs=100, verbose=0)
print(model.evaluate(X,y))
# Test performance with 5-layer network with sigmoid activation
model = Sequential([
    Input(shape=(2,)),
    Dense(5, "sigmoid"),
    Dense(5, "sigmoid"),
    Dense(5, "sigmoid"),
    Dense(1, "sigmoid")
])
model.compile(optimizer="adam", loss="binary_crossentropy",
metrics=["acc"])
model.fit(X, y, batch_size=32, epochs=100, verbose=0)
print(model.evaluate(X,y))
# Illustrate weights across epochs
class WeightCapture(Callback):
    "Capture the weights of each layer of the model"
    def __init__(self, model):
        super().__init__()
        self.model = model
        self.weights = []
        self.epochs = []

    def on_epoch_end(self, epoch, logs=None):
        self.epochs.append(epoch) # remember the epoch axis
        weight = {}
        for layer in model.layers:
            if not layer.weights:
                continue
            name = layer.weights[0].name.split("/")[0]
            weight[name] = layer.weights[0].numpy()
        self.weights.append(weight)
def make_mlp(activation, initializer, name):
    "Create a model with specified activation and initalizer"
    model = Sequential([
        Input(shape=(2,), name=name+"0"),
```

```
        Dense(5, activation=activation, kernel_initializer=initializer,
name=name+"1"),
        Dense(5, activation=activation, kernel_initializer=initializer,
name=name+"2"),
        Dense(5, activation=activation, kernel_initializer=initializer,
name=name+"3"),
        Dense(5, activation=activation, kernel_initializer=initializer,
name=name+"4"),
        Dense(1, activation=activation, kernel_initializer=initializer,
name=name+"5")
    ])
    return model
def plotweight(capture_cb):
    "Plot the weights' mean and s.d. across epochs"
    fig, ax = plt.subplots(2, 1, sharex=True, constrained_layout=True,
figsize=(8, 10))
    ax[0].set_title("Mean weight")
    for key in capture_cb.weights[0]:
        ax[0].plot(capture_cb.epochs, [w[key].mean() for w in
capture_cb.weights], label=key)
    ax[0].legend()
    ax[1].set_title("S.D.")
    for key in capture_cb.weights[0]:
        ax[1].plot(capture_cb.epochs, [w[key].std() for w in
capture_cb.weights], label=key)
    ax[1].legend()
    plt.show()
initializer = RandomNormal(mean=0, stddev=1)
batch_size = 32
n_epochs = 100
# sigmoid activation
model = make_mlp("sigmoid", initializer, "sigmoid")
capture_cb = WeightCapture(model)
capture_cb.on_epoch_end(-1)
model.compile(optimizer="rmsprop", loss="binary_crossentropy",
metrics=["acc"])
print("Before training: Accuracy", accuracy_score(y, (model(X).numpy()
> 0.5).astype(int)))
model.fit(X, y, batch_size=batch_size, epochs=n_epochs,
callbacks=[capture_cb], verbose=0)
print("After training: Accuracy", accuracy_score(y, (model(X).numpy() >
0.5).astype(int)))
```

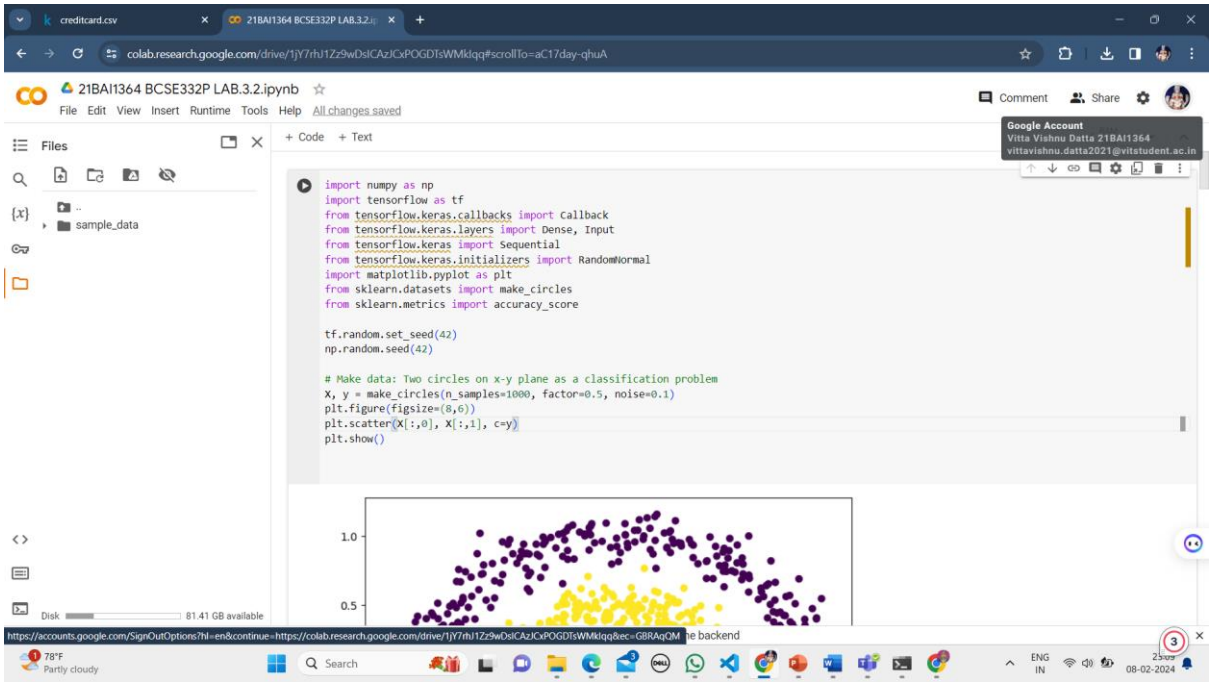
```
print(model.evaluate(X,y))
plotweight(capture_cb)
# tanh activation
model = make_mlp("tanh", initializer, "tanh")
capture_cb = WeightCapture(model)
capture_cb.on_epoch_end(-1)
model.compile(optimizer="rmsprop", loss="binary_crossentropy",
metrics=["acc"])
print("Before training: Accuracy", accuracy_score(y, (model(X).numpy()
> 0.5).astype(int)))
model.fit(X, y, batch_size=batch_size, epochs=n_epochs,
callbacks=[capture_cb], verbose=0)
print("After training: Accuracy", accuracy_score(y, (model(X).numpy() >
0.5).astype(int)))
print(model.evaluate(X,y))
plotweight(capture_cb)
# ReLu activation
model = make_mlp("relu", initializer, "relu")
capture_cb = WeightCapture(model)
capture_cb.on_epoch_end(-1)
model.compile(optimizer="rmsprop", loss="binary_crossentropy",
metrics=["acc"])
print("Before training: Accuracy", accuracy_score(y, (model(X).numpy()
> 0.5).astype(int)))
model.fit(X, y, batch_size=batch_size, epochs=n_epochs,
callbacks=[capture_cb], verbose=0)
print("After training: Accuracy", accuracy_score(y, (model(X).numpy() >
0.5).astype(int)))
print(model.evaluate(X,y))
plotweight(capture_cb)
```

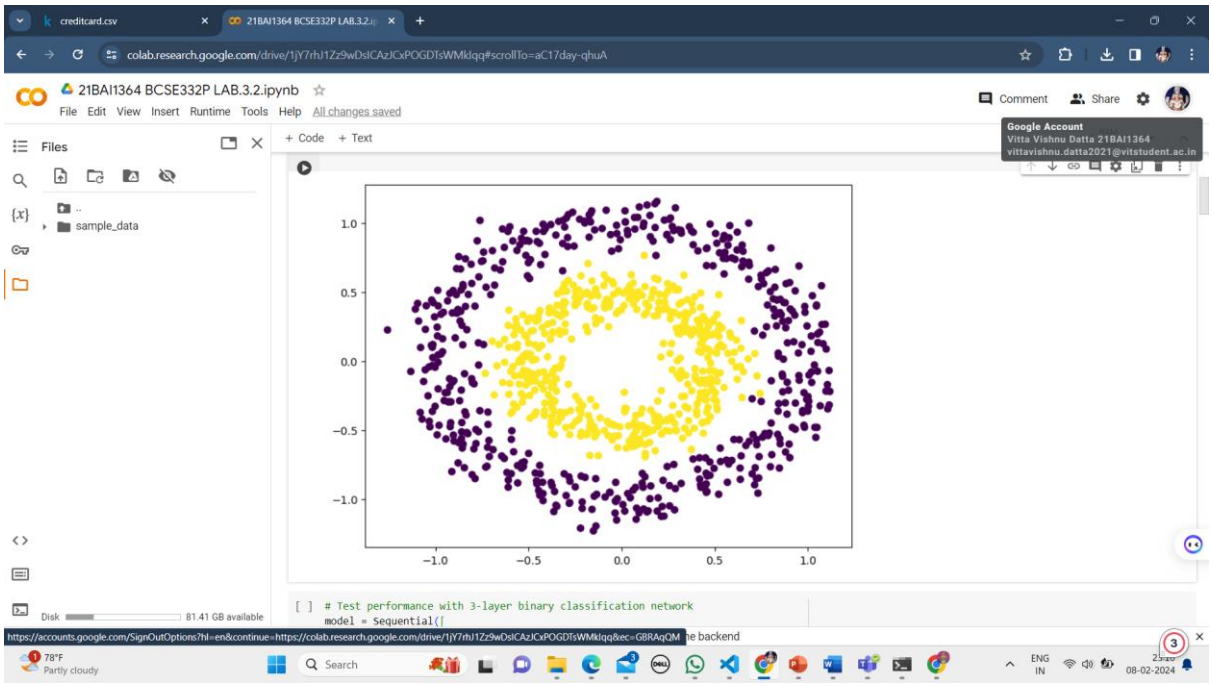
LINK TO THE GOOGLE COLAB NOTEBOOK:

https://colab.research.google.com/drive/1jY7rhJ1Zz9wDsICAzJCxPOGDTsWMkIqq

VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)
CHENNAI

OUTPUT OF THE CODE SNIPPET:

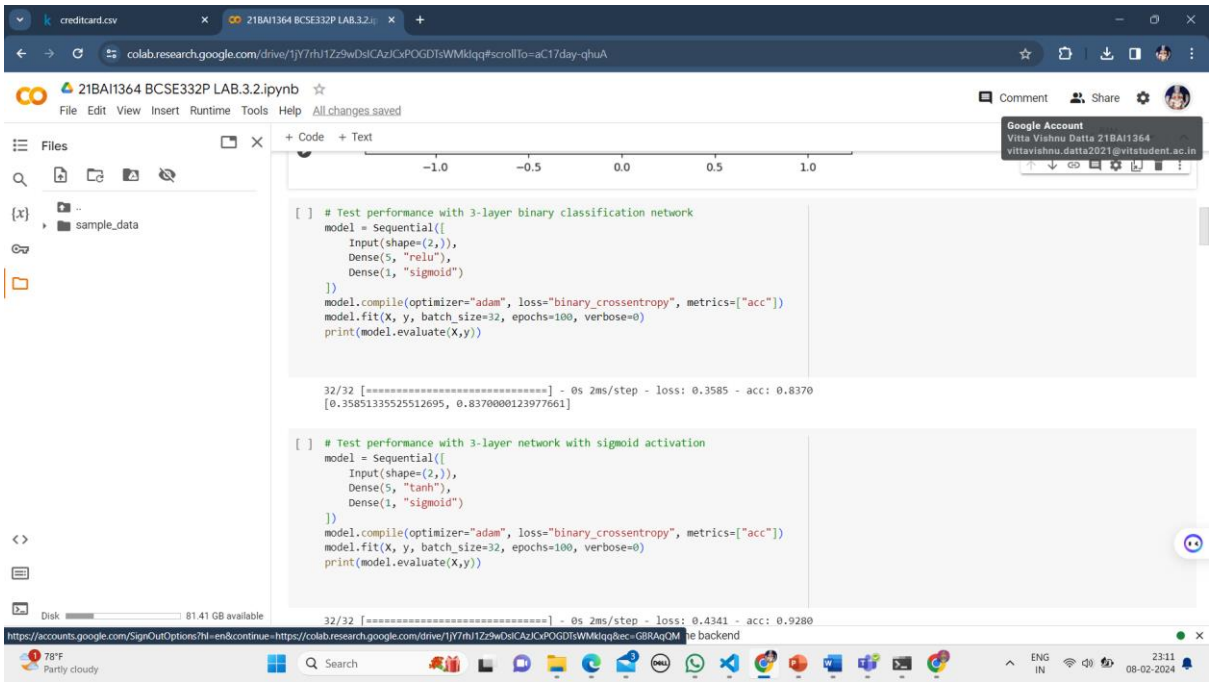


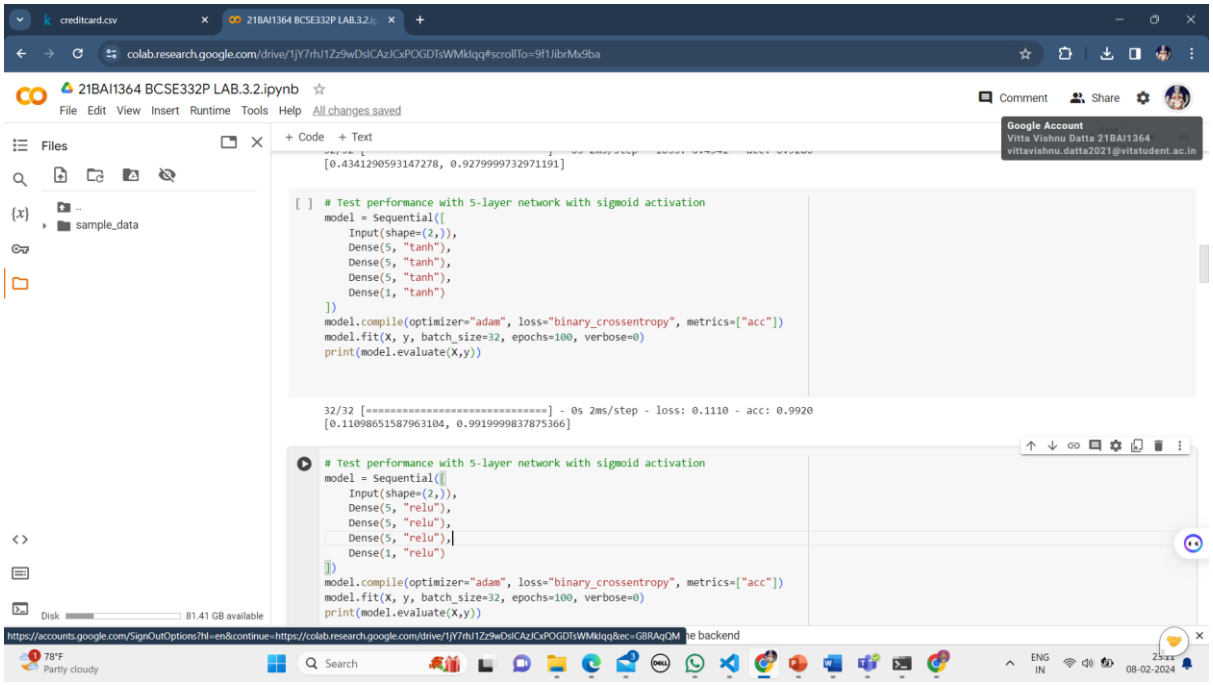


Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

CHENNAI



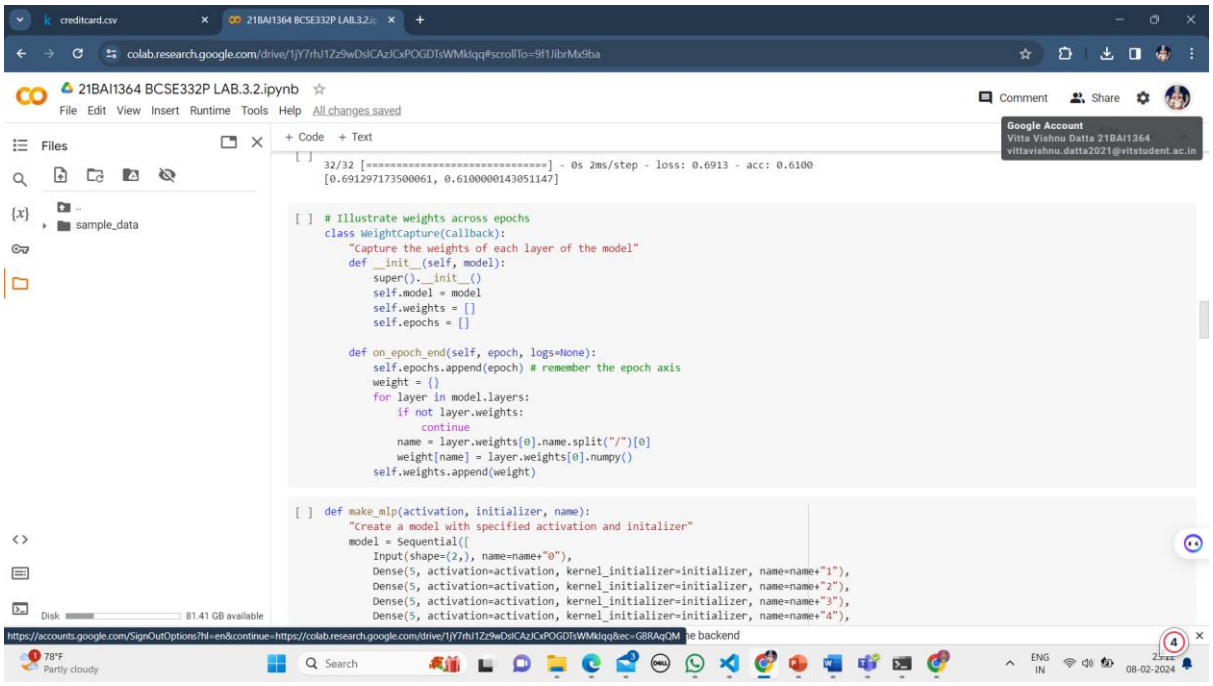


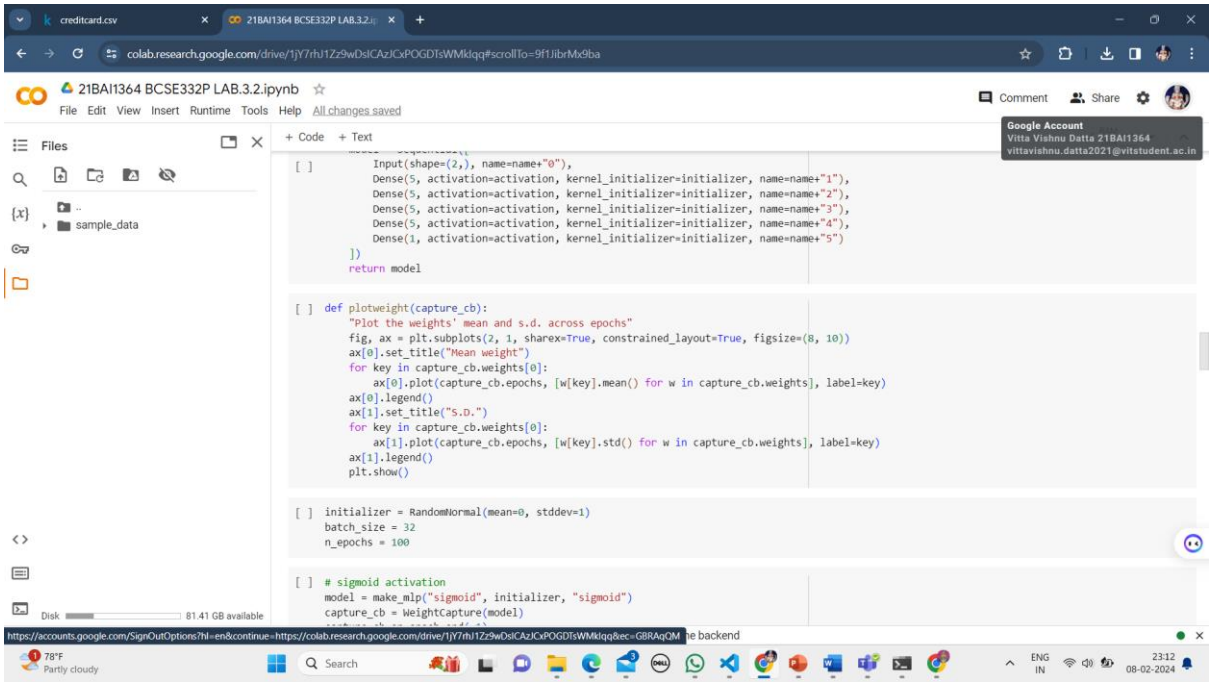
VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

CHENNAI



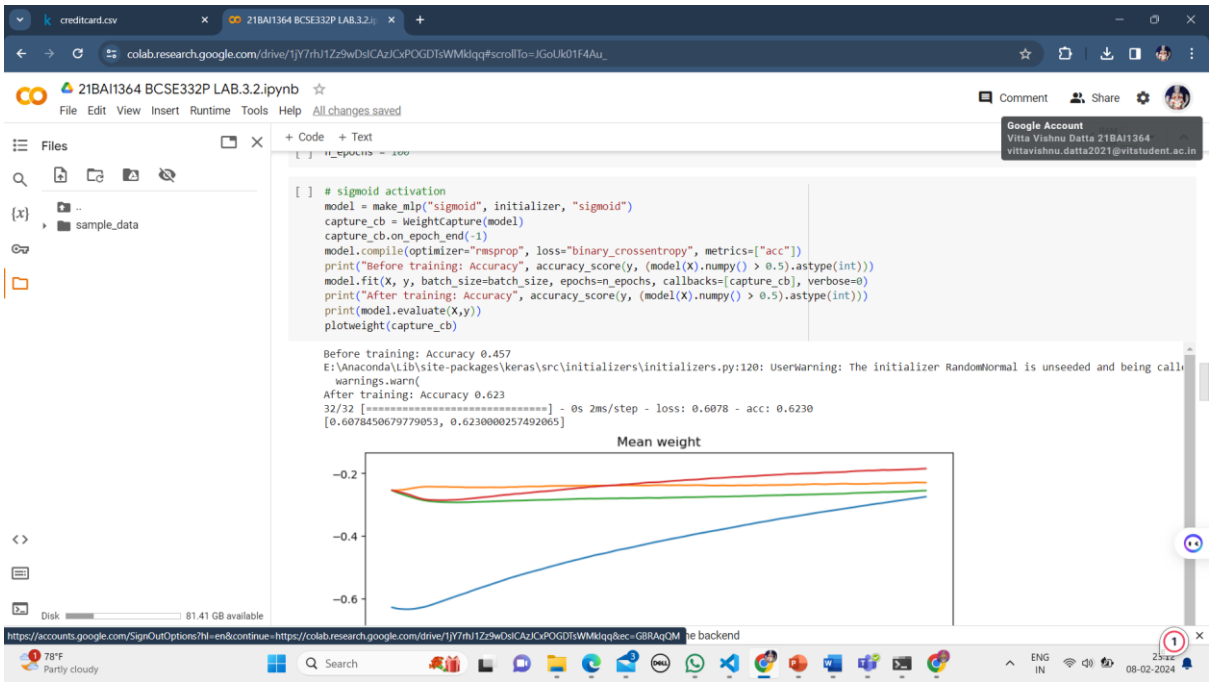


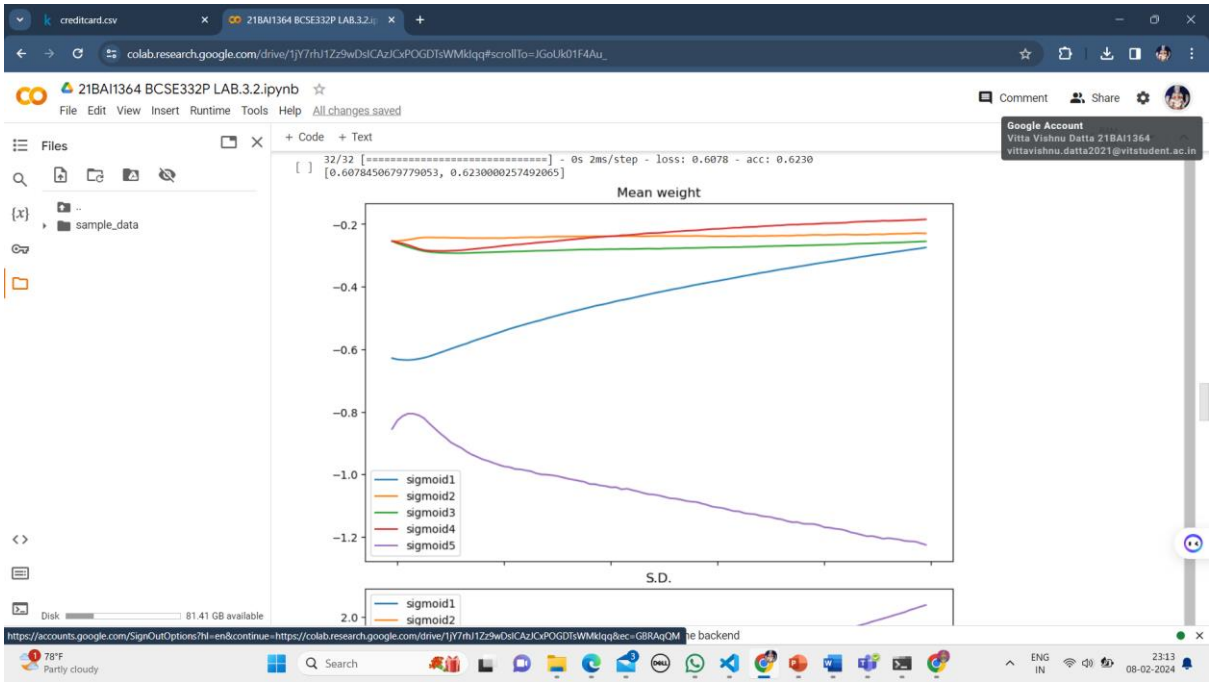
VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

CHENNAI



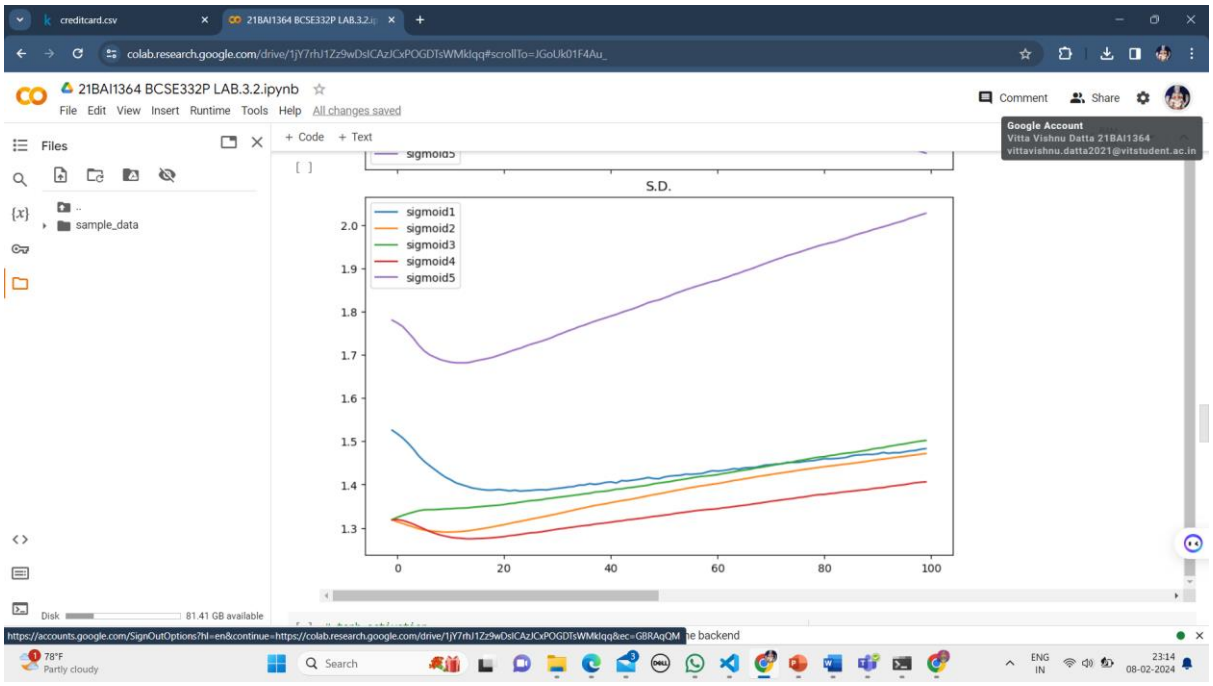


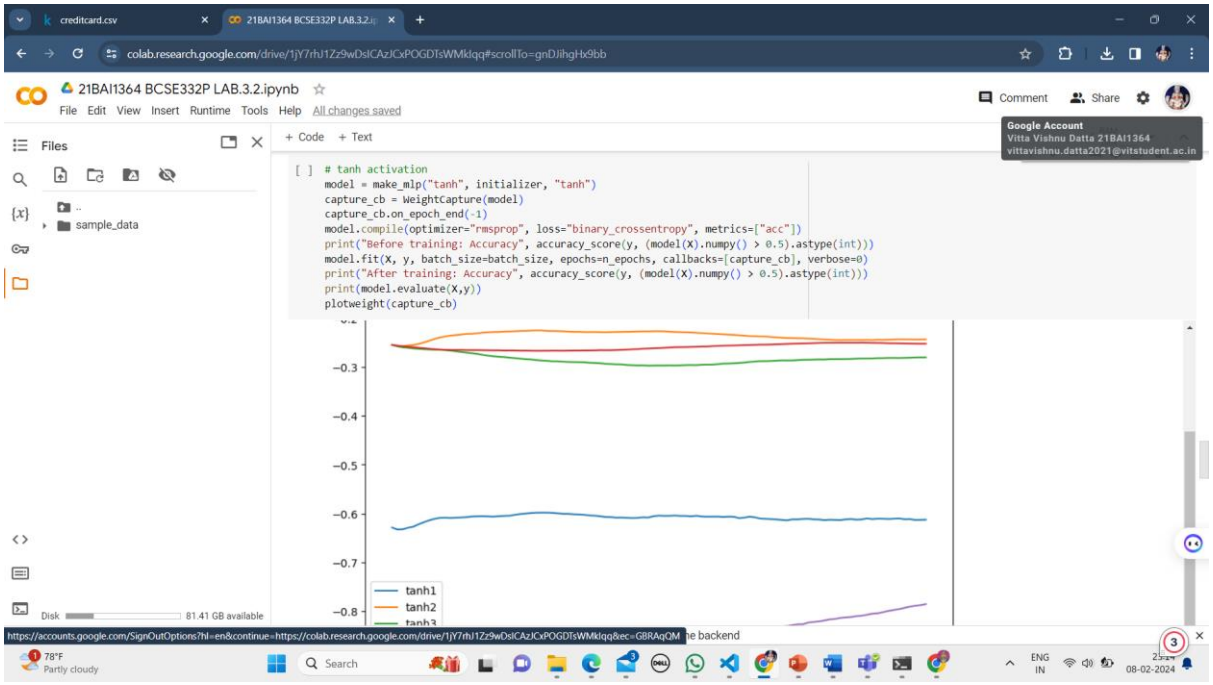
VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

CHENNAI



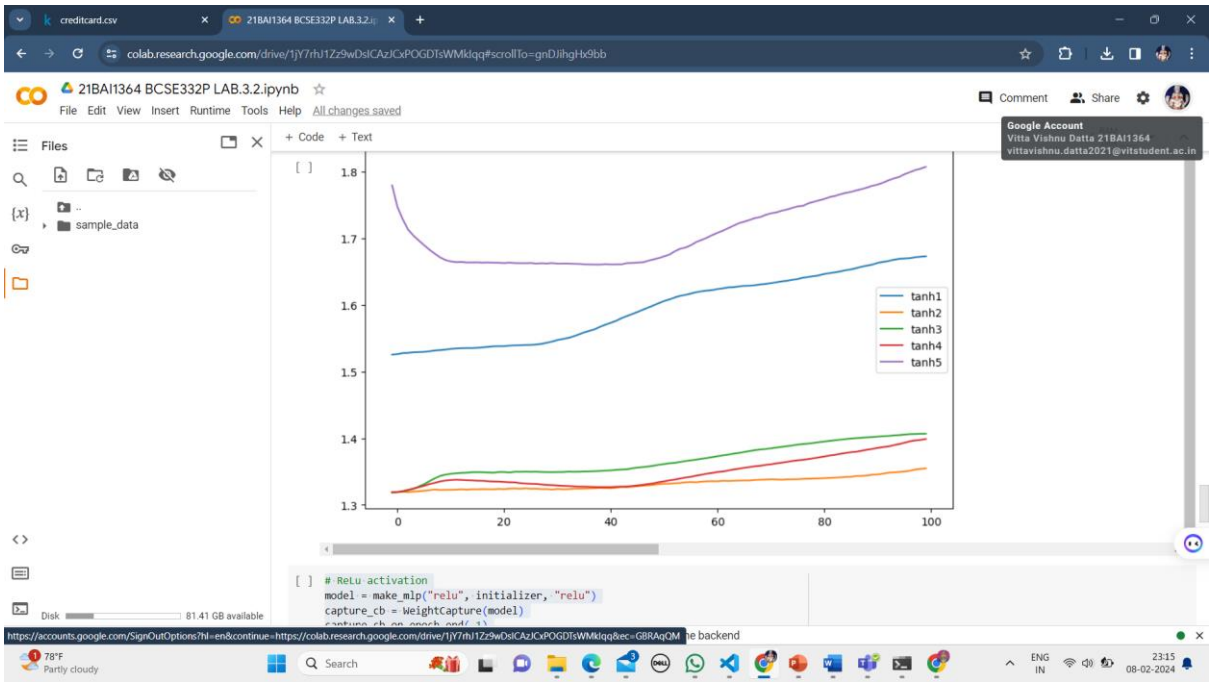


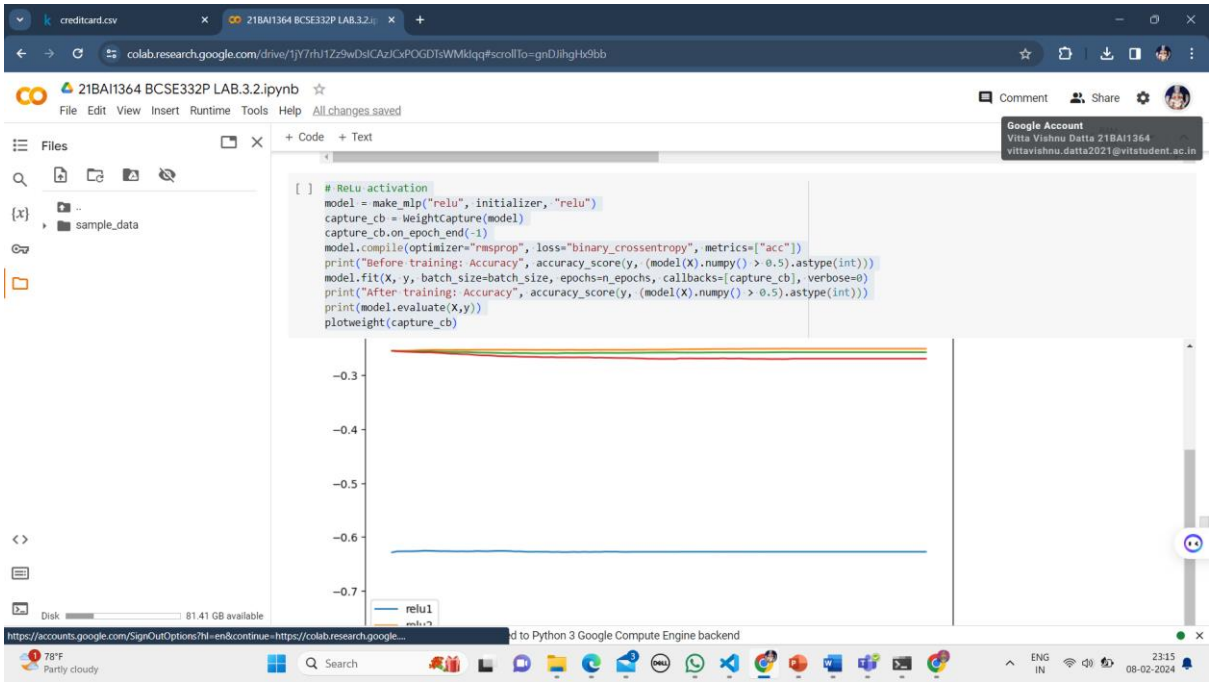
VIT®

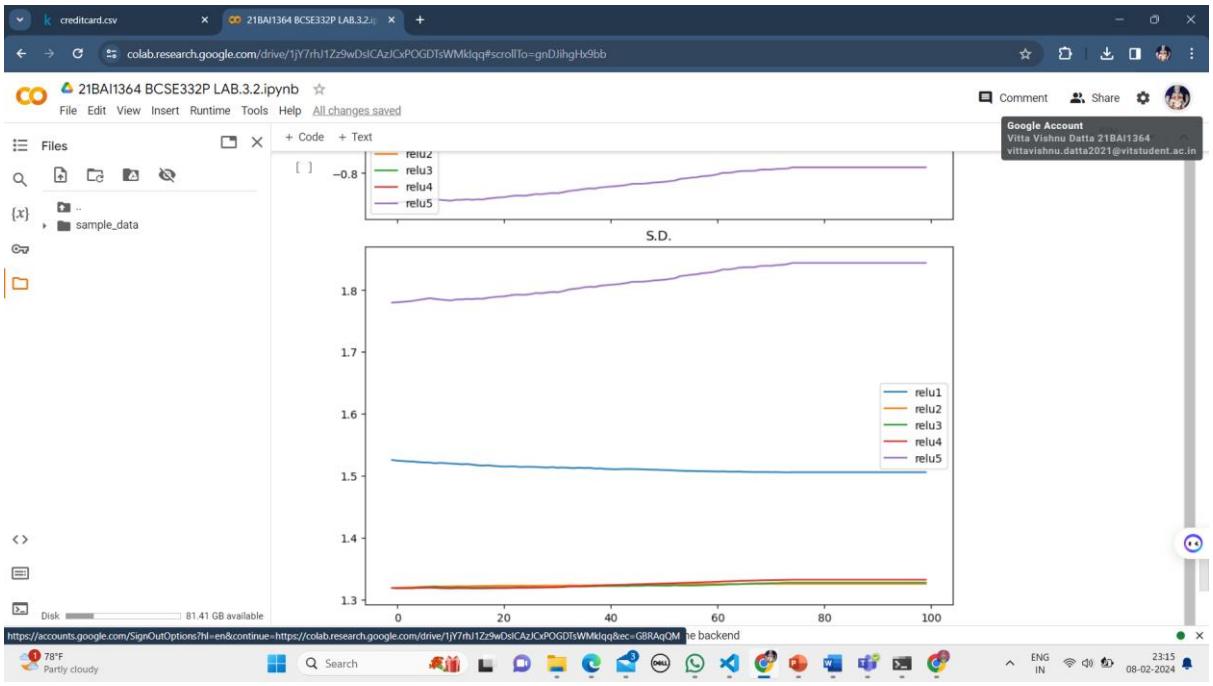
Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

CHENNAI







3. Implement a code to compare GD, SGD, Minibatch SGD for any input/neural model of your choice.

CODE:

```
import numpy as np
import tensorflow as tf
from tensorflow.keras import layers, models, optimizers
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
# Generate synthetic dataset
X, y = make_classification(n_samples=1000, n_features=20, n_classes=2,
random_state=42)
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
# Standardize the data
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
```

```
X_test = scaler.transform(X_test)
# Build a simple neural network model
def build_model():
    model = models.Sequential()
    model.add(layers.Dense(64, activation='relu',
input_shape=(X_train.shape[1],)))
    model.add(layers.Dense(1, activation='sigmoid'))
    return model
# Train the model using different optimizers
def train_model(optimizer, epochs=50, batch_size=None):
    model = build_model()
    model.compile(optimizer=optimizer, loss='binary_crossentropy',
metrics=['accuracy'])

    history = None
    if batch_size:
        history = model.fit(X_train, y_train, epochs=epochs,
batch_size=batch_size, validation_data=(X_test, y_test), verbose=0)
    else:
        history = model.fit(X_train, y_train, epochs=epochs,
validation_data=(X_test, y_test), verbose=0)

    # Evaluate the model on test set
    loss, accuracy = model.evaluate(X_test, y_test)
    print(f'\nOptimizer: {optimizer.get_config()["name"]}')
    print(f'Test Loss: {loss:.4f}, Test Accuracy: {accuracy:.4f}')

    # Make predictions
    predictions = model.predict(X_test)
    print(predictions)

    # Plot training history
    plot_training_history(history, optimizer.get_config()["name"])

    # Plot ROC curve
    plot_roc_curve(y_test, predictions, optimizer.get_config()["name"])


# Plot training history
def plot_training_history(history, optimizer_name):
    plt.figure(figsize=(12, 4))

    # Plot training & validation accuracy values
```

```
    plt.subplot(1, 2, 1)
    plt.plot(history.history['accuracy'])
    plt.plot(history.history['val_accuracy'])
    plt.title(f'Model Accuracy - {optimizer_name}')
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.legend(['Train', 'Validation'], loc='upper left')

    # Plot training & validation loss values
    plt.subplot(1, 2, 2)
    plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.title(f'Model Loss - {optimizer_name}')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.legend(['Train', 'Validation'], loc='upper left')

    plt.tight_layout()
    plt.show()
```

```
# Plot ROC curve
def plot_roc_curve(y_true, y_pred, optimizer_name):
    from sklearn.metrics import roc_curve, auc

    fpr, tpr, thresholds = roc_curve(y_true, y_pred)
    roc_auc = auc(fpr, tpr)

    plt.figure(figsize=(8, 6))
    plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve
(area = {roc_auc:.2f})')
    plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title(f'ROC Curve - {optimizer_name}')
    plt.legend(loc='lower right')
    plt.show()
```

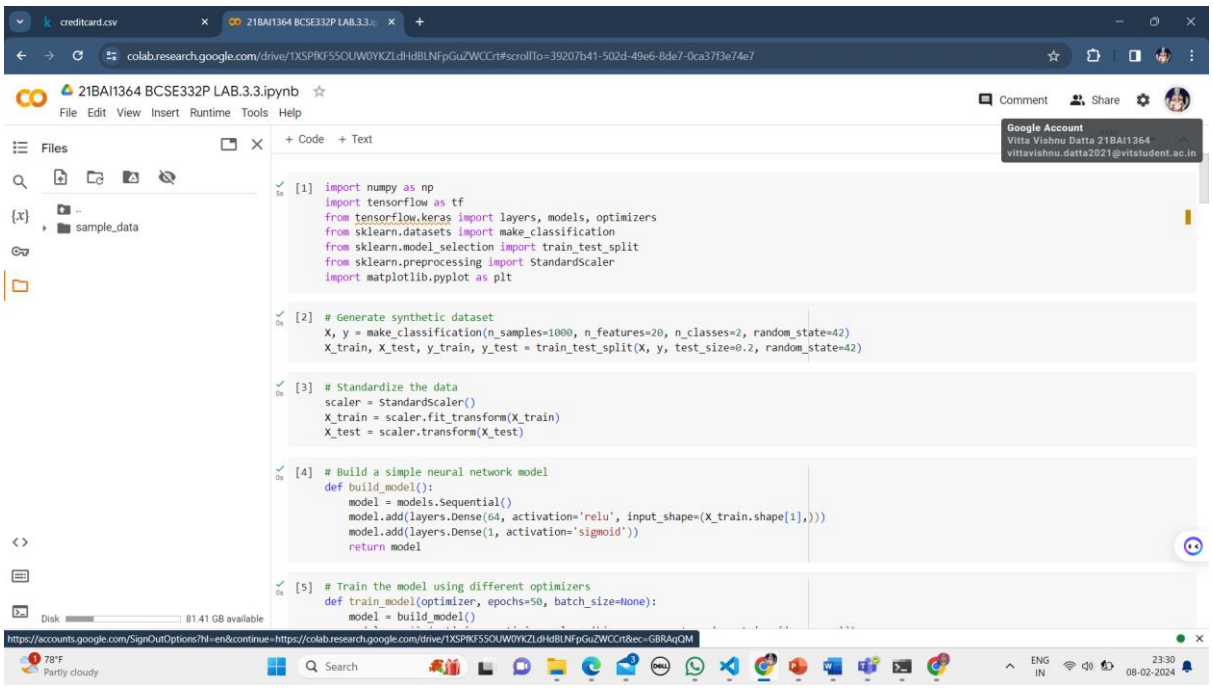
```
# Compare GD, SGD, and Mini-batch SGD
epochs = 50
```

```
# Gradient Descent
gd_optimizer = optimizers.SGD(learning_rate=0.01)
train_model(gd_optimizer, epochs=epochs)
# Stochastic Gradient Descent
sgd_optimizer = optimizers.SGD(learning_rate=0.01)
train_model(sgd_optimizer, epochs=epochs)
# Mini-batch Stochastic Gradient Descent
batch_size = 32
minibatch_sgd_optimizer = optimizers.SGD(learning_rate=0.01)
train_model(minibatch_sgd_optimizer, epochs=epochs,
batch_size=batch_size)
```

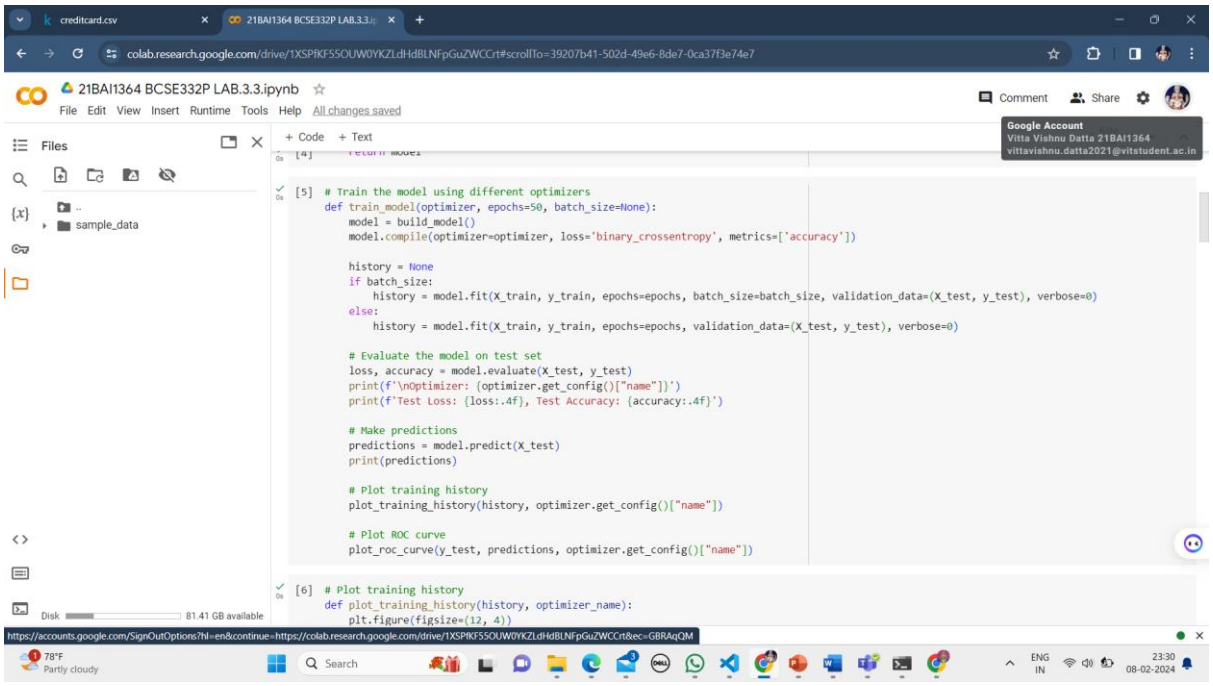
LINK TO THE GOOGLE COLAB NOTEBOOK:

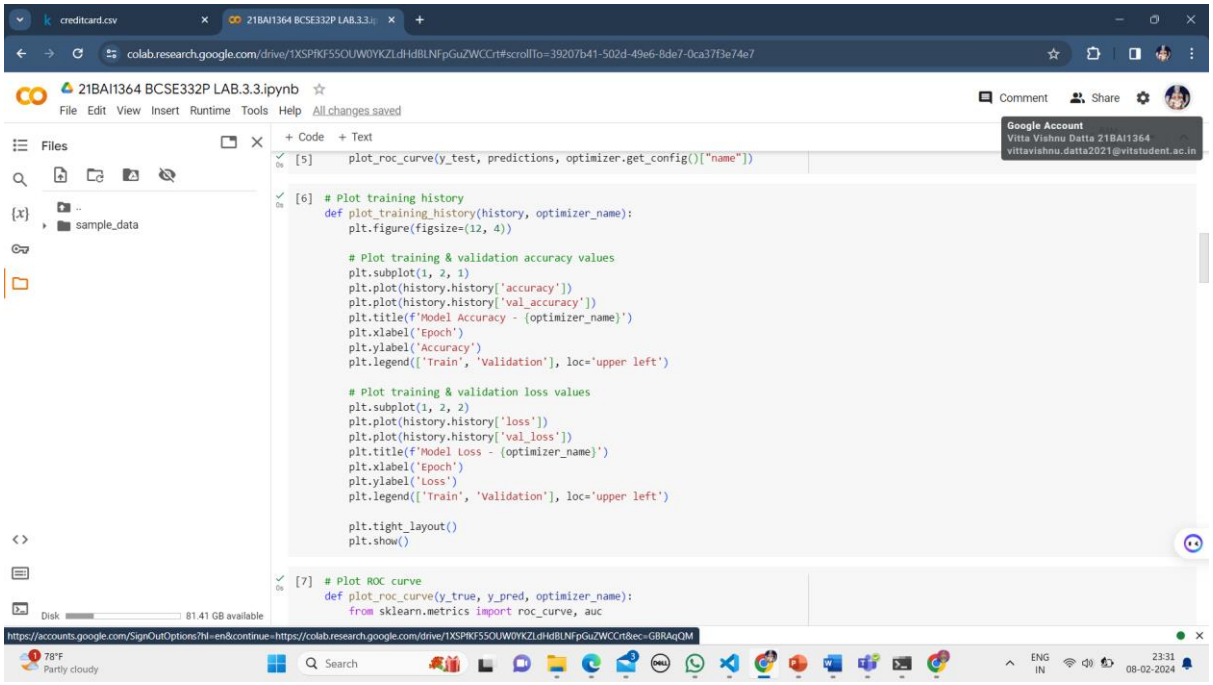
https://colab.research.google.com/drive/1XSPfKF55OUW0YKZLdHdBLNFpGuZWCCrt#scrollTo=39207b41-502d-49e6-8de7-0ca37f3e74e7

OUTPUT OF THE CODE SNIPPET:



VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)
CHENNAI

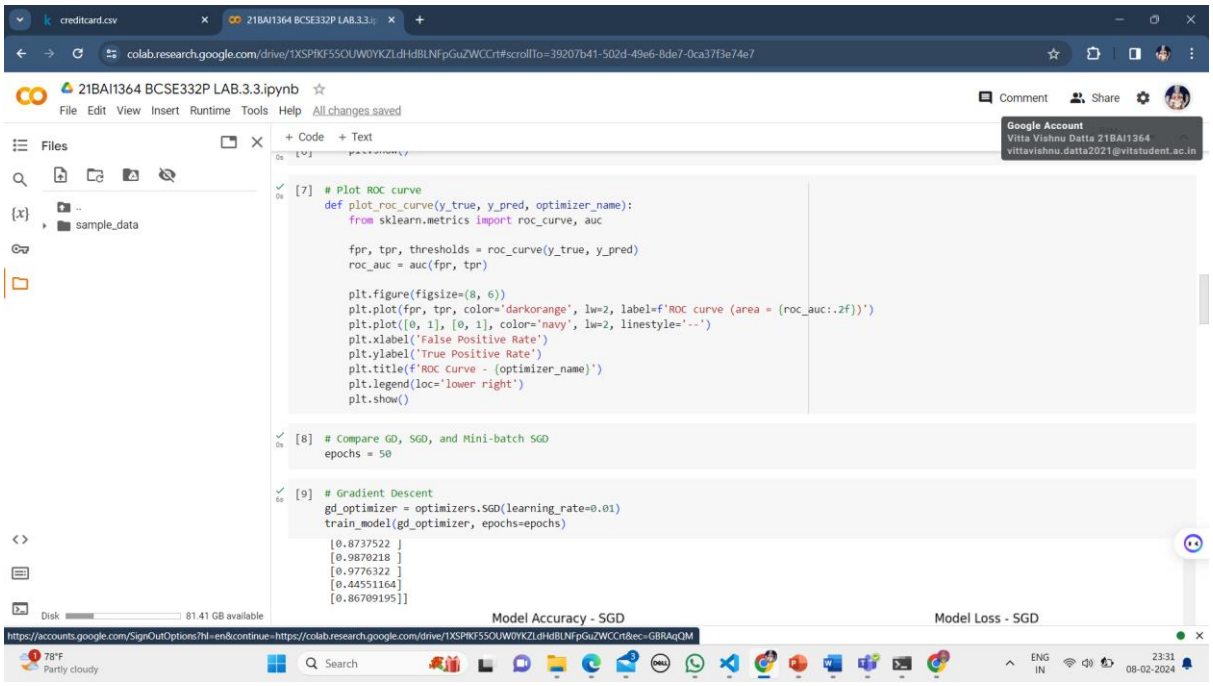


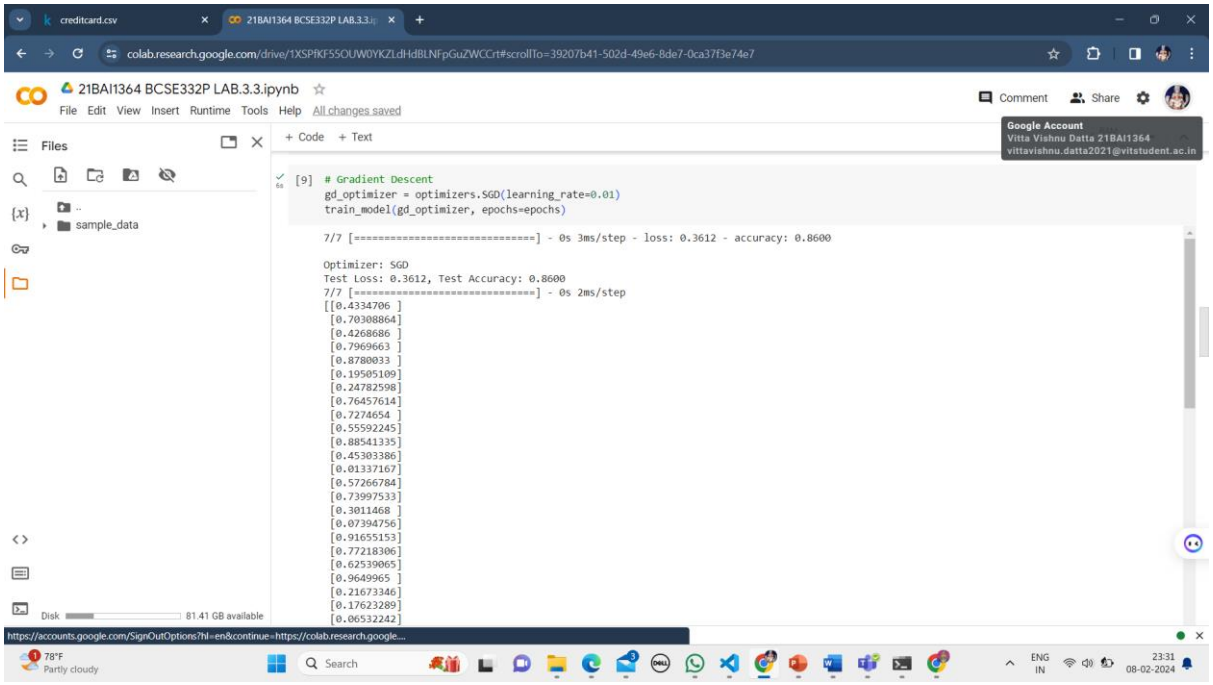


Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

CHENNAI



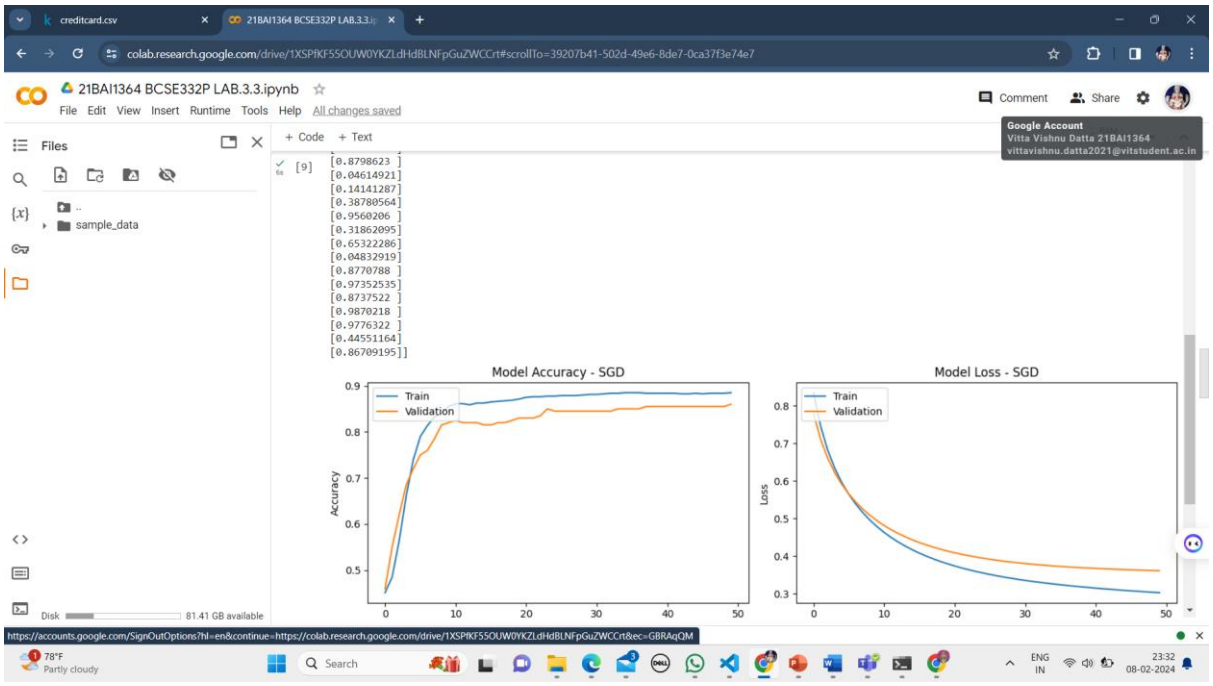


VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

CHENNAI

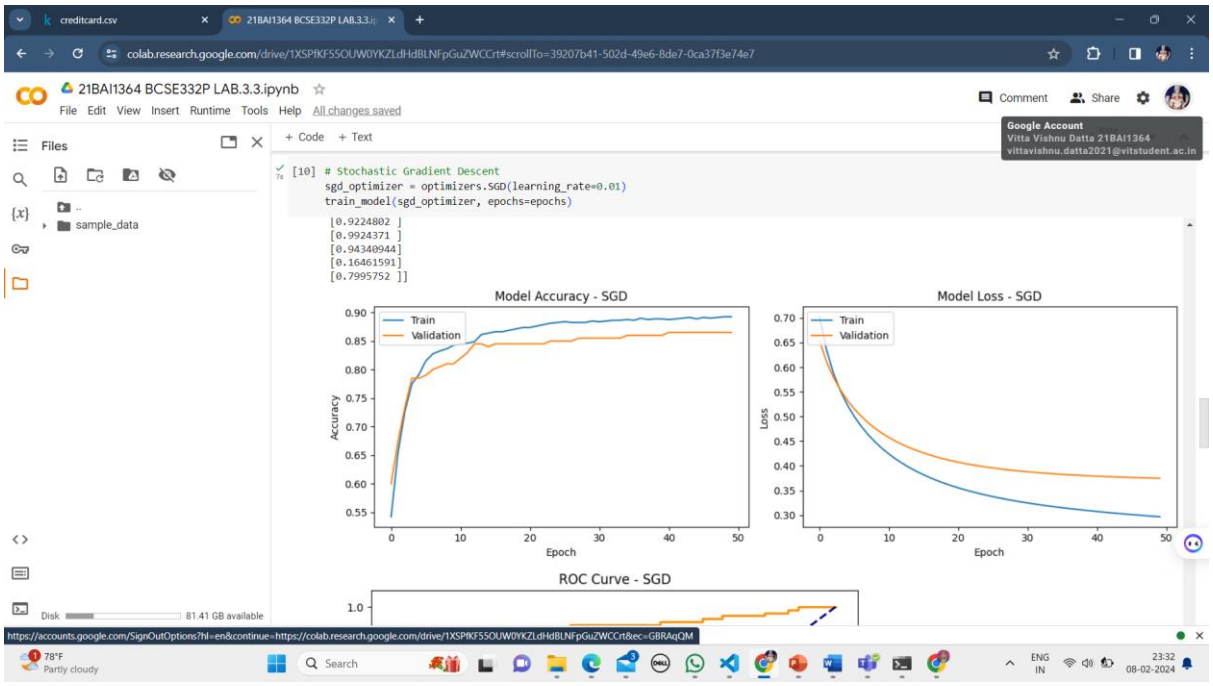


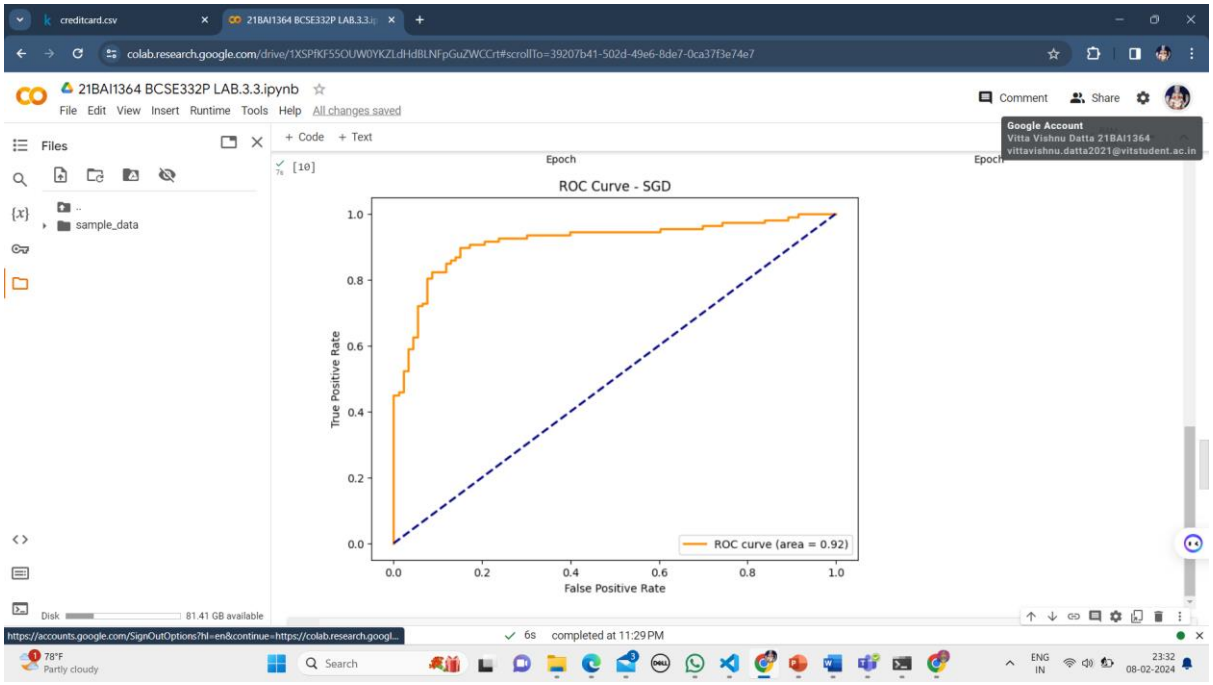
VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

CHENNAI



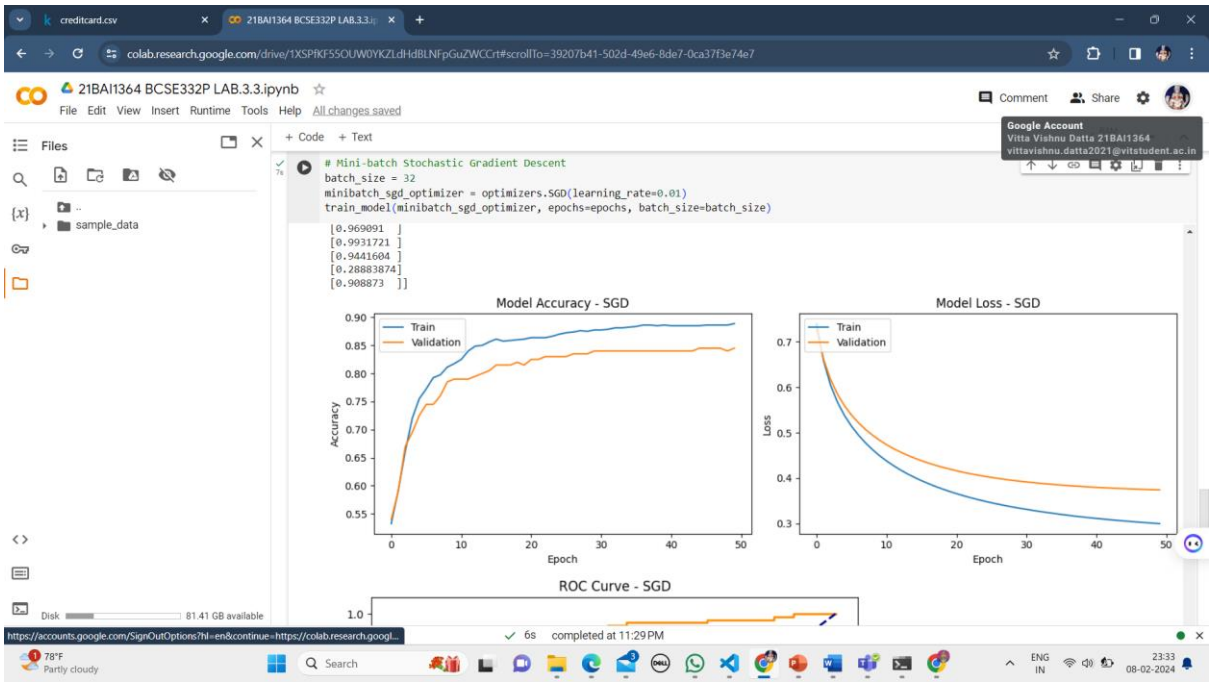


VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

CHENNAI



VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

CHENNAI

PREPARED AND SUBMITTED BY:

VITTA VISHNU DATTA.

21BAI1364 – SCOPE.

VIT – CHENNAI CAMPUS.