Student Name: Mohammad Mohammad Beigi - Pantea Amoie
Student ID: 99102189 - 400101656
Subject: ML Privacy

Introduction to Machine Learning - Dr. R. Amiri
Final Project

1 Machine Unlearning

2 Private Training Attack

2.1 Base Model Learning

```
Train: X=(60000, 28, 28), y=(60000,)
Test: X=(10000, 28, 28), y=(10000,)
Model: "sequential_2"
```

Layer (type)	Output Shape	Param #
flatten_2 (Flatten)	(None, 784)	0
dense_4 (Dense)	(None, 128)	100,480
dense_5 (Dense)	(None, 10)	1,290

```
 Total params: 101,770 (397.54 KB)
 Trainable params: 101,770 (397.54 KB)
 Non-trainable params: 0 (0.00 B)
Epoch 1/5
94/94 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.5073 - loss: 82.7176
Epoch 2/5
94/94 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.7224 - loss: 13.4081
Epoch 3/5
94/94 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7444 - loss: 8.7707
Epoch 4/5
94/94 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.7929 - loss: 5.0982
Epoch 5/5
94/94 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.7845 - loss: 1.9695
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6379 - loss: 3.7948

 Model  0  test accuracy: 0.6439999938011169
```

Figure 1:

2.2 Base Model Shadow Learning and attack

```
 Model  0  test accuracy: 0.6439999938011169
Epoch 1/5
94/94 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4784 - loss: 90.4529
Epoch 2/5
94/94 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.7028 - loss: 16.0662
Epoch 3/5
94/94 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.7385 - loss: 9.1251
Epoch 4/5
94/94 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7479 - loss: 7.1349
Epoch 5/5
94/94 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7733 - loss: 5.2279
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6955 - loss: 7.1460

 Model  0  test accuracy: 0.6790000200271606
Epoch 1/5
94/94 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4726 - loss: 82.6930
Epoch 2/5
94/94 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.6843 - loss: 16.4903
Epoch 3/5
94/94 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7010 - loss: 5.9780
Epoch 4/5
94/94 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.7113 - loss: 1.6879
Epoch 5/5
94/94 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.7634 - loss: 1.2455
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7023 - loss: 2.9090

 Model  1  test accuracy: 0.7080000042915344
Training svm model :  0
[LibSVM]SVM model  0 score :  0.6102150537634409
```

Figure 2: Number of shadow models = 2

2.3 Private Model Learning

```
Train: X=(60000, 28, 28), y=(60000,)
Test: X=(10000, 28, 28), y=(10000,)
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten_4 (Flatten)         (None, 784)               0

 dense_8 (Dense)             (None, 128)               100480

 dropout_6 (Dropout)         (None, 128)               0

 dense_9 (Dense)             (None, 10)                1290

=================================================================
Total params: 101770 (397.54 KB)
Trainable params: 101770 (397.54 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Epoch 1/5
94/94 [==============================] - 1s 4ms/step - loss: 24.4152 - accuracy: 0.3330
Epoch 2/5
94/94 [==============================] - 0s 3ms/step - loss: 2.3659 - accuracy: 0.4293
Epoch 3/5
94/94 [==============================] - 0s 3ms/step - loss: 1.9294 - accuracy: 0.4753
Epoch 4/5
94/94 [==============================] - 0s 4ms/step - loss: 1.7721 - accuracy: 0.5003
Epoch 5/5
94/94 [==============================] - 0s 3ms/step - loss: 1.7717 - accuracy: 0.4990
63/63 [==============================] - 0s 2ms/step - loss: 1.8605 - accuracy: 0.5755

 Model  0  test accuracy: 0.5755000114440918
```

Figure 3: Private Trained Model by adding DropOut

2.4 Private Model Shadow Learning and attack

```
 Model  0  test accuracy: 0.5755000114440918
Epoch 1/5
94/94 [==============================] - 1s 3ms/step - loss: 28.6290 - accuracy: 0.3900
Epoch 2/5
94/94 [==============================] - 0s 3ms/step - loss: 2.2916 - accuracy: 0.3817
Epoch 3/5
94/94 [==============================] - 0s 3ms/step - loss: 1.9308 - accuracy: 0.4120
Epoch 4/5
94/94 [==============================] - 0s 3ms/step - loss: 1.7087 - accuracy: 0.4530
Epoch 5/5
94/94 [==============================] - 0s 4ms/step - loss: 1.6937 - accuracy: 0.4740
63/63 [==============================] - 0s 2ms/step - loss: 1.3593 - accuracy: 0.6210

 Model  0  test accuracy: 0.6209999918937683
Epoch 1/5
94/94 [==============================] - 1s 3ms/step - loss: 24.4149 - accuracy: 0.3767
Epoch 2/5
94/94 [==============================] - 0s 3ms/step - loss: 2.3103 - accuracy: 0.3637
Epoch 3/5
94/94 [==============================] - 0s 4ms/step - loss: 1.9929 - accuracy: 0.4233
Epoch 4/5
94/94 [==============================] - 1s 6ms/step - loss: 1.8375 - accuracy: 0.4540
Epoch 5/5
94/94 [==============================] - 0s 5ms/step - loss: 1.7843 - accuracy: 0.4907
63/63 [==============================] - 0s 3ms/step - loss: 1.3897 - accuracy: 0.5945

 Model  1  test accuracy: 0.5945000052452087
Training svm model :  0
[LibSVM]SVM model  0 score :  0.48118279569892475
```

Figure 4: Number of shadow models = 2

2.5 Base Model Increasing Shadow Models

Model: "sequential_5"

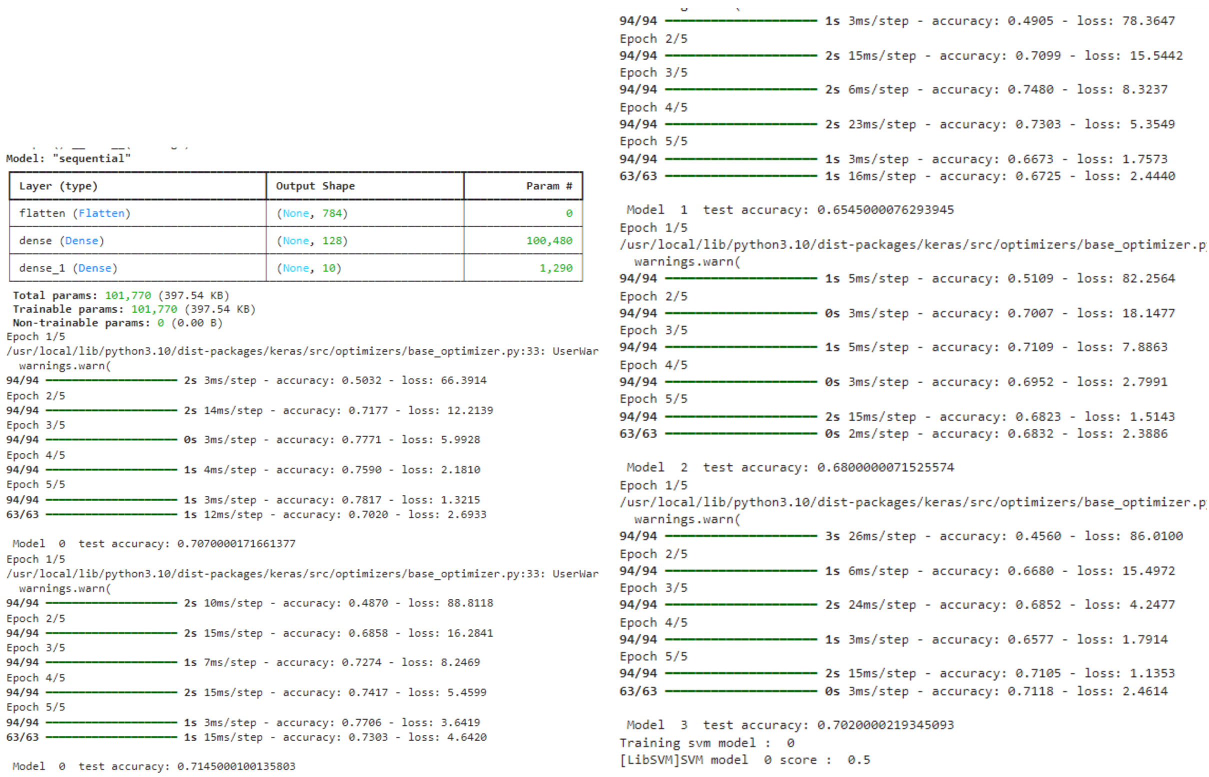
Layer (type)	Output Shape	Param #
flatten_5 (Flatten)	(None, 784)	0
dense_10 (Dense)	(None, 128)	100,480
dropout_1 (Dropout)	(None, 128)	0
dense_11 (Dense)	(None, 10)	1,290

```
 Total params: 101,770 (397.54 KB)
 Trainable params: 101,770 (397.54 KB)
 Non-trainable params: 0 (0.00 B)
Epoch 1/5
94/94 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - accuracy: 0.4059 - loss: 85.0269
Epoch 2/5
94/94 ━━━━━━━━━━━━━━━━━━━━ 4s 35ms/step - accuracy: 0.4181 - loss: 2.2720
Epoch 3/5
94/94 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.4721 - loss: 1.8457
Epoch 4/5
94/94 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4717 - loss: 2.0073
Epoch 5/5
94/94 ━━━━━━━━━━━━━━━━━━━━ 1s 14ms/step - accuracy: 0.5214 - loss: 1.6442
63/63 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - accuracy: 0.6274 - loss: 1.5189

 Model  0  test accuracy: 0.6119999885559082
Epoch 1/5
94/94 ━━━━━━━━━━━━━━━━━━━━ 2s 6ms/step - accuracy: 0.3803 - loss: 74.3437
Epoch 2/5
94/94 ━━━━━━━━━━━━━━━━━━━━ 2s 23ms/step - accuracy: 0.3819 - loss: 2.2994
Epoch 3/5
94/94 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.4140 - loss: 1.8617
Epoch 4/5
94/94 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - accuracy: 0.4580 - loss: 1.7917
Epoch 5/5
94/94 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4706 - loss: 1.5689
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6694 - loss: 1.3016

 Model  0  test accuracy: 0.6539999842643738
```

```
 Model  0  test accuracy: 0.6539999842643738
Epoch 1/5
94/94 ━━━━━━━━━━━━━━━━━━━━ 2s 10ms/step - accuracy: 0.3444 - loss: 78.2884
Epoch 2/5
94/94 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.3500 - loss: 2.5628
Epoch 3/5
94/94 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.3818 - loss: 2.1204
Epoch 4/5
94/94 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.4212 - loss: 1.8821
Epoch 5/5
94/94 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step - accuracy: 0.4528 - loss: 1.8724
63/63 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.5746 - loss: 1.5327

 Model  1  test accuracy: 0.5590000152587891
Epoch 1/5
94/94 ━━━━━━━━━━━━━━━━━━━━ 3s 16ms/step - accuracy: 0.3453 - loss: 68.9072
Epoch 2/5
94/94 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.4100 - loss: 2.5006
Epoch 3/5
94/94 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.4345 - loss: 2.1591
Epoch 4/5
94/94 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.4741 - loss: 1.7842
Epoch 5/5
94/94 ━━━━━━━━━━━━━━━━━━━━ 2s 21ms/step - accuracy: 0.4934 - loss: 1.6147
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6377 - loss: 1.9835

 Model  2  test accuracy: 0.6305000185966492
Epoch 1/5
94/94 ━━━━━━━━━━━━━━━━━━━━ 3s 20ms/step - accuracy: 0.3645 - loss: 72.3915
Epoch 2/5
94/94 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.4231 - loss: 2.5470
Epoch 3/5
94/94 ━━━━━━━━━━━━━━━━━━━━ 2s 12ms/step - accuracy: 0.4441 - loss: 2.0397
Epoch 4/5
94/94 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.4795 - loss: 1.7557
Epoch 5/5
94/94 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4982 - loss: 1.8641
63/63 ━━━━━━━━━━━━━━━━━━━━ 1s 15ms/step - accuracy: 0.6661 - loss: 1.3550

 Model  3  test accuracy: 0.6614999771118164
Training svm model :  0
[LibSVM]SVM model  0 score :  0.667632665465435
```

Figure 5: Number of shadow models = 4

2.6 Safe Model Increasing Shadow Models



```
Model: "sequential"
```

Layer (type)	Output Shape	Param #
flatten (Flatten)	(None, 784)	0
dense (Dense)	(None, 128)	100,480
dense_1 (Dense)	(None, 10)	1,290

```
 Total params: 101,770 (397.54 KB)
 Trainable params: 101,770 (397.54 KB)
 Non-trainable params: 0 (0.00 B)
Epoch 1/5
/usr/local/lib/python3.10/dist-packages/keras/src/optimizers/base_optimizer.py:33: UserWar
  warnings.warn(
94/94 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.5032 - loss: 66.3914
Epoch 2/5
94/94 ━━━━━━━━━━━━━━━━━━━━ 2s 14ms/step - accuracy: 0.7177 - loss: 12.2139
Epoch 3/5
94/94 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7771 - loss: 5.9928
Epoch 4/5
94/94 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.7590 - loss: 2.1810
Epoch 5/5
94/94 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.7817 - loss: 1.3215
63/63 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - accuracy: 0.7020 - loss: 2.6933

 Model  0  test accuracy: 0.7070000171661377
Epoch 1/5
/usr/local/lib/python3.10/dist-packages/keras/src/optimizers/base_optimizer.py:33: UserWar
  warnings.warn(
94/94 ━━━━━━━━━━━━━━━━━━━━ 2s 10ms/step - accuracy: 0.4870 - loss: 88.8118
Epoch 2/5
94/94 ━━━━━━━━━━━━━━━━━━━━ 2s 15ms/step - accuracy: 0.6858 - loss: 16.2841
Epoch 3/5
94/94 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - accuracy: 0.7274 - loss: 8.2469
Epoch 4/5
94/94 ━━━━━━━━━━━━━━━━━━━━ 2s 15ms/step - accuracy: 0.7417 - loss: 5.4599
Epoch 5/5
94/94 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.7706 - loss: 3.6419
63/63 ━━━━━━━━━━━━━━━━━━━━ 1s 15ms/step - accuracy: 0.7303 - loss: 4.6420

 Model  0  test accuracy: 0.7145000100135803
```

```
94/94 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4905 - loss: 78.3647
Epoch 2/5
94/94 ━━━━━━━━━━━━━━━━━━━━ 2s 15ms/step - accuracy: 0.7099 - loss: 15.5442
Epoch 3/5
94/94 ━━━━━━━━━━━━━━━━━━━━ 2s 6ms/step - accuracy: 0.7480 - loss: 8.3237
Epoch 4/5
94/94 ━━━━━━━━━━━━━━━━━━━━ 2s 23ms/step - accuracy: 0.7303 - loss: 5.3549
Epoch 5/5
94/94 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.6673 - loss: 1.7573
63/63 ━━━━━━━━━━━━━━━━━━━━ 1s 16ms/step - accuracy: 0.6725 - loss: 2.4440

 Model  1  test accuracy: 0.6545000076293945
Epoch 1/5
/usr/local/lib/python3.10/dist-packages/keras/src/optimizers/base_optimizer.p
  warnings.warn(
94/94 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.5109 - loss: 82.2564
Epoch 2/5
94/94 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7007 - loss: 18.1477
Epoch 3/5
94/94 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.7109 - loss: 7.8863
Epoch 4/5
94/94 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.6952 - loss: 2.7991
Epoch 5/5
94/94 ━━━━━━━━━━━━━━━━━━━━ 2s 15ms/step - accuracy: 0.6823 - loss: 1.5143
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6832 - loss: 2.3886

 Model  2  test accuracy: 0.6800000071525574
Epoch 1/5
/usr/local/lib/python3.10/dist-packages/keras/src/optimizers/base_optimizer.p
  warnings.warn(
94/94 ━━━━━━━━━━━━━━━━━━━━ 3s 26ms/step - accuracy: 0.4560 - loss: 86.0100
Epoch 2/5
94/94 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.6680 - loss: 15.4972
Epoch 3/5
94/94 ━━━━━━━━━━━━━━━━━━━━ 2s 24ms/step - accuracy: 0.6852 - loss: 4.2477
Epoch 4/5
94/94 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.6577 - loss: 1.7914
Epoch 5/5
94/94 ━━━━━━━━━━━━━━━━━━━━ 2s 15ms/step - accuracy: 0.7105 - loss: 1.1353
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7118 - loss: 2.4614

 Model  3  test accuracy: 0.7020000219345093
Training svm model :  0
[LibSVM]SVM model  0 score :  0.5
```

Figure 6: Number of shadow models = 4

3 Membership Inference Attack

We trained an attacker with 2 shadow models similar to Question 4, but using the new model.

```
from torchvision import models
import torch
import torchvision
import torchvision.transforms as transforms
from torch.utils.data import DataLoader
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import numpy as np
import matplotlib.pyplot as plt


class CIFAR10Classifier(nn.Module):
def __init__(self):
super(CIFAR10Classifier, self).__init__()
self.conv1 = nn.Conv2d(3, 16, 3, 1)
self.conv2 = nn.Conv2d(16, 32, 3, 1)
self.dropout1 = nn.Dropout2d(0.25)
self.dropout2 = nn.Dropout2d(0.5)
self.fc1 = nn.Linear(6272, 64)
self.fc2 = nn.Linear(64, 10)

def forward(self, x):
x = self.conv1(x)
x = F.relu(x)
```

```
x = self.conv2(x)
x = F.relu(x)
x = F.max_pool2d(x, 2)
x = self.dropout1(x)
x = torch.flatten(x, 1)
x = self.fc1(x)
x = F.relu(x)
x = self.dropout2(x)
x = self.fc2(x)
return x

import tensorflow as tf
import tensorflow.keras
from tensorflow.keras.utils import to_categorical
import numpy as np
from sklearn.utils import resample
from sklearn.metrics import ConfusionMatrixDisplay
import matplotlib.pyplot as plt
from matplotlib import pyplot



def load_fashion_mnist_dataset():
from keras.datasets import fashion_mnist

# load dataset
(trainX, trainY), (testX, testY) = fashion_mnist.load_data()
# summarize loaded dataset
print('Train: X=%s, y=%s' % (trainX.shape, trainY.shape))
print('Test: X=%s, y=%s' % (testX.shape, testY.shape))

# reshape dataset to have a single channel
trainX = trainX.reshape((trainX.shape[0], 28, 28, 1))
testX = testX.reshape((testX.shape[0], 28, 28, 1))

# one hot encode target values
trainY = to_categorical(trainY)
testY = to_categorical(testY)

return (trainX, trainY), (testX, testY)

# Set hyper-parameters for our training of neural networks
LEARNING_RATE = 0.001
EPOCH = 5
VERBOSE = 1

TRAINING_SIZE = 3000
TEST_SIZE = 2000

# No of target models
NUM_TARGET = 1
# No of shadow models
NUM_SHADOW = 2

# Label value "in" for records present in training data of shadow models
IN = 1
# Label value "out" for records not present in training data of shadow models
OUT = 0

def sample_data(train_data, test_data, num_sets):
(x_train, y_train), (x_test, y_test) = train_data, test_data
```

```
new_x_train, new_y_train = [], []
new_x_test, new_y_test = [], []
for i in range(num_sets):
x_temp, y_temp = resample(x_train, y_train, n_samples=TRAINING_SIZE, random_state=0)
new_x_train.append(x_temp)
new_y_train.append(y_temp)
x_temp, y_temp = resample(x_test, y_test, n_samples=TEST_SIZE, random_state=0)
new_x_test.append(x_temp)
new_y_test.append(y_temp)
return (new_x_train, new_y_train), (new_x_test, new_y_test)

def get_attack_dataset(models, train_data, test_data, num_models, data_size):
# generate dataset for the attack model
(x_train, y_train), (x_test, y_test) = train_data, test_data
# set number of classes for the attack model
num_classes = 10
x_data, y_data = [[] for _ in range(num_classes)], [[] for _ in range(num_classes)]
for i in range(num_models):
# IN data
x_temp, y_temp = resample(x_train[i], y_train[i], n_samples=data_size, random_state=0)
for j in range(data_size):
y_idx = np.argmax(y_temp[j])
x_data[y_idx].append(models[i].predict(x_temp[j:j+1], verbose=0)[0])
y_data[y_idx].append(IN)

# OUT data
x_temp, y_temp = resample(x_test[i], y_test[i], n_samples=data_size, random_state=0)
for j in range(data_size):
y_idx = np.argmax(y_temp[j])
x_data[y_idx].append(models[i].predict(x_temp[j:j+1], verbose=0)[0])
y_data[y_idx].append(OUT)

return x_data, y_data

def build_fcnn_model_fashion_mnist():
model = tf.keras.models.Sequential([
tf.keras.layers.Flatten(input_shape=(28, 28)),
tf.keras.layers.Dense(128, activation='relu'),
tf.keras.layers.Dense(10, activation='softmax')
])
model.summary()
return model

def get_trained_keras_models(keras_model, train_data, test_data, num_models):
(x_train, y_train), (x_test, y_test) = train_data, test_data
models = []
for i in range(num_models):
models.append(tf.keras.models.clone_model(keras_model))
rms = tf.keras.optimizers.RMSprop(learning_rate=LEARNING_RATE, decay=1e-7)
models[i].compile(loss='categorical_crossentropy', optimizer=rms, metrics=['accuracy'])
models[i].fit(x_train[i], y_train[i], batch_size=32, epochs=EPOCH, verbose=VERBOSE,
  shuffle=True)
score = models[i].evaluate(x_test[i], y_test[i], verbose=VERBOSE)
print('\n', 'Model ', i, ' test accuracy:', score[1])
return models

def get_trained_svm_models(train_data, test_data, num_models=1):
from sklearn import svm
(x_train, y_train), (x_test, y_test) = train_data, test_data
models = []
for i in range(num_models):
```

```
print('Training svm model : ', i)
models.append(svm.SVC(gamma='scale', kernel='linear', verbose=VERBOSE))
models[i].fit(x_train[i], y_train[i])
score = models[i].score(x_test[i], y_test[i])
print('SVM model ', i, 'score : ', score)
return models

def membership_attack():
# load the pre-shuffled train and test data
(x_train, y_train), (x_test, y_test) = load_fashion_mnist_dataset()

# split the data for each model
target_train = (x_train[:TRAINING_SIZE*NUM_TARGET], y_train[:TRAINING_SIZE*NUM_TARGET])
target_test = (x_test[:TEST_SIZE*NUM_TARGET], y_test[:TEST_SIZE*NUM_TARGET])
target_train_data, target_test_data = sample_data(target_train, target_test, NUM_TARGET)

shadow_train = (x_train[TRAINING_SIZE*NUM_TARGET:], y_train[TRAINING_SIZE*NUM_TARGET:])
shadow_test = (x_test[TEST_SIZE*NUM_TARGET:], y_test[TEST_SIZE*NUM_TARGET:])
shadow_train_data, shadow_test_data = sample_data(shadow_train, shadow_test, NUM_SHADOW)

cnn_model = build_fcnn_model_fashion_mnist()

# compile the target model
target_models = get_trained_keras_models(cnn_model, target_train_data, target_test_data,
  NUM_TARGET)
# compile the shadow models
shadow_models = get_trained_keras_models(cnn_model, shadow_train_data, shadow_test_data,
  NUM_SHADOW)

# get train data for the attack model
attack_train = get_attack_dataset(shadow_models, shadow_train_data, shadow_test_data,
  NUM_SHADOW, TEST_SIZE)
# get test data for the attack model
attack_test = get_attack_dataset(target_models, target_train_data, target_test_data,
  NUM_TARGET, TEST_SIZE)

# training the attack model
#attack_model = get_trained_svm_models(attack_train, attack_test)
return attack_train, attack_test
NUM_SHADOW = 2
membership_attack()


import torch
import torch.nn as nn
import torch.optim as optim
from torchvision.datasets import CIFAR10
from torchvision import transforms
from torch.utils.data import Subset, DataLoader, TensorDataset
from sklearn.metrics import confusion_matrix, precision_score, recall_score ,f1_score
from sklearn.linear_model import LogisticRegression

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

model = CIFAR10Classifier()
state_dict = torch.load("model_state_dict.pth", map_location=device)
new_state_dict = {key.replace('_module.', ''): value for key, value in state_dict.items()}
model.load_state_dict(new_state_dict)
model.to(device)
model.eval()
```

```
transform = transforms.Compose([
transforms.ToTensor(),
transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
])

DATA_ROOT = '../cifar10'
BATCH_SIZE = 64

# Load the indices from list.txt
indices_file = 'list.txt' ############
with open(indices_file, 'r') as f:
indices = [int(line.strip()) for line in f]

full_train_dataset = CIFAR10(root=DATA_ROOT, train=True, download=True, transform=transform)
test_dataset = CIFAR10(root=DATA_ROOT, train=False, download=True, transform=transform)

train_indices_set = set(indices)
all_indices = set(range(len(full_train_dataset)))
other_indices = list(all_indices - train_indices_set)

train_dataset = Subset(full_train_dataset, indices[:len(indices)//2])  ###########
other_dataset = Subset(full_train_dataset, other_indices)

# Create data loaders
train_loader = DataLoader(train_dataset, batch_size=BATCH_SIZE, shuffle=False)
other_loader = DataLoader(other_dataset, batch_size=BATCH_SIZE, shuffle=False)
test_loader = DataLoader(test_dataset, batch_size=BATCH_SIZE, shuffle=False)

# Create labels
train_labels = torch.ones(len(train_dataset)).to(device)
other_labels = torch.zeros(len(other_dataset)).to(device)
test_labels = torch.zeros(len(test_dataset)).to(device)
####################################
#if you have an attacker model for each class, modify the above code.
####################################

def extract_features(model, dataloader):
model.eval()
features = []
with torch.no_grad():
for data in dataloader:
inputs, _ = data
inputs = inputs.to(device)
outputs = model(inputs)
features.append(outputs)
return torch.cat(features).to(device)

train_features = extract_features(model, train_loader)
other_features = extract_features(model, other_loader)
test_features = extract_features(model, test_loader)


combined_features = torch.cat((train_features, other_features, test_features))
combined_labels = torch.cat((train_labels, other_labels, test_labels))


new_dataset = TensorDataset(combined_features, combined_labels)
new_loader = DataLoader(new_dataset, batch_size=BATCH_SIZE, shuffle=True)

#load your attacker model
```

```
##############################################

# Calculate training accuracy, confusion matrix, precision, and recall
binary_classifier.eval()
all_labels = []
all_predicted = []
correct = 0
total = 0
with torch.no_grad():
for features, labels in new_loader:
features, labels = features.to(device), labels.to(device)
outputs = get_trained_svm_models(features, attack_test).squeeze()
predicted = (outputs > 0.5).float()
total += labels.size(0)
correct += (predicted == labels).sum().item()
all_labels.extend(labels.cpu().numpy())
all_predicted.extend(predicted.cpu().numpy())

accuracy = correct / total


cm = confusion_matrix(all_labels, all_predicted)
precision = precision_score(all_labels, all_predicted)
recall = recall_score(all_labels, all_predicted)
f1 = f1_score(all_labels, all_predicted)

print(f'Confusion Matrix:\n{cm}')
print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"F1 Score: {f1:.4f}")
print(f'Training Accuracy: {accuracy:.4f}')
```

```
Confusion Matrix:
[[ 9452  8766]
 [ 8361 13421]]
Precision: 0.6049
Recall: 0.6162
F1 Score: 0.6105
Training Accuracy: 0.5718
```

Figure 7:

PyTorch and CIFAR-10 Classifier

Explanation:

1. **Imports**:
   - `torchvision`, `torch`, `transforms`: For handling data transformations and model-related functionalities.
   - `nn`, `F`: For defining neural network layers and activation functions.
   - `optim`: For optimization algorithms.
   - `numpy`, `matplotlib.pyplot`: For numerical operations and plotting.

2. **CIFAR10Classifier Class**:
   - Inherits from `nn.Module`.
   - **Initialization (__init__)**: Defines the structure of the neural network.
     - Convolutional layers (`conv1`, `conv2`), dropout layers (`dropout1`, `dropout2`), and fully connected layers (`fc1`, `fc2`).
   - **Forward Method**: Defines the forward pass through the network.
     - Applies convolution, activation (ReLU), pooling, dropout, and flattening operations sequentially.

TensorFlow and CIFAR 10 Dataset

Explanation:

1. **Imports**:
   - `tensorflow`, `keras.utils`: For building and managing neural networks.
   - `numpy`, `sklearn.utils`, `sklearn.metrics`, `matplotlib.pyplot`: For data handling, resampling, and plotting.
2. **load_fashion_mnist_dataset Function**:
   - Loads the Fashion-MNIST dataset.
   - Reshapes the dataset to have a single channel.
   - Converts the labels to categorical format (one-hot encoding).
   - Returns the training and testing datasets.

Sampling and Training

Explanation:

1. **sample_data Function**:
   - Resamples training and testing datasets to create multiple sets.
   - Helps in creating diverse training and testing sets for training models.
2. **get_attack_dataset Function**:
   - Generates datasets for training attack models.
   - Uses predictions from multiple models on resampled data to create in-distribution (IN) and out-of-distribution (OUT) samples.

Neural Network and SVM Models

Explanation:

1. **build_fcnn_model_fashion_mnist Function**:
   - Builds a fully connected neural network for Fashion-MNIST.
   - Uses two dense layers with ReLU and softmax activations.

2. **get_trained_keras_models Function**:
   - Trains multiple instances of a given Keras model.
   - Uses RMSprop optimizer and evaluates each model.
3. **get_trained_svm_models Function**:
   - Trains multiple SVM models using the provided training data.
   - Evaluates each model and prints the accuracy.

Membership Attack

Explanation:

1. **membership_attack Function**:
   - Implements a membership inference attack.
   - Loads and preprocesses the Fashion-MNIST dataset.
   - Splits data for training target and shadow models.
   - Builds and trains neural networks (both target and shadow models).
   - Creates datasets for training an attack model using predictions from shadow models.
   - Returns the attack training and testing datasets.

Evaluating CIFAR-10 Classifier with Torch

Explanation:

1. **Device Setup**:
   - Checks for GPU availability and sets the device accordingly.
2. **Model Loading**:
   - Loads the pretrained CIFAR10Classifier model state.
3. **Data Preparation**:
   - Defines transformations for the CIFAR-10 dataset.
   - Loads CIFAR-10 dataset and splits it based on indices from a file.
4. **Feature Extraction**:
   - Extracts features from the trained model for training, other, and test datasets.
5. **Creating New Dataset**:
   - Combines extracted features and labels into a new dataset.
   - Creates a DataLoader for the new dataset.
6. **Evaluation**:
   - Evaluates the binary classifier on the new dataset.
   - Computes accuracy, confusion matrix, precision, recall, and F1 score.