Import necessary libraries

In [155…

```
import numpy as np
import tensorflow as tf
import tensorflow_datasets as tfds
from easydict import EasyDict
from tensorflow.keras import Model
from tensorflow.keras.layers import Dense, Dropout, Flatten, Conv2D
import sklearn as sk
from cleverhans.tf2.attacks.basic_iterative_method import basic_iterative_method
from cleverhans.tf2.attacks.projected_gradient_descent import projected_gradient_descent
from cleverhans.tf2.attacks.fast_gradient_method import fast_gradient_method
from cleverhans.tf2.attacks.carlini_wagner_l2 import carlini_wagner_l2
import warnings
warnings.filterwarnings("ignore")
```

Create CNN Architecture

In [156…

```
class CNN(Model):
    def __init__(self):
        super(CNN, self).__init__()
        self.conv1 = Conv2D(64, 8, strides=(2, 2), activation="relu", padding="same")
        self.conv2 = Conv2D(128, 6, strides=(2, 2), activation="relu", padding="valid")
        self.conv3 = Conv2D(128, 5, strides=(1, 1), activation="relu", padding="valid")
        self.dropout = Dropout(0.2)
        self.flatten = Flatten()
        self.dense1 = Dense(128, activation="relu")
        self.dense2 = Dense(10)

    def call(self, x):
        x = self.conv1(x)
        x = self.conv2(x)
        x = self.conv3(x)
        x = self.dropout(x)
        x = self.flatten(x)
        x = self.dense1(x)

        return self.dense2(x)

model = CNN()
model.build(input_shape=(None, 28, 28, 3))
model.summary()
```

```
Model: "cnn_28"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_147 (Conv2D)         multiple                  12352

 conv2d_148 (Conv2D)         multiple                  295040

 conv2d_149 (Conv2D)         multiple                  409728

 dropout_49 (Dropout)        multiple                  0

 flatten_49 (Flatten)        multiple                  0

 dense_98 (Dense)            multiple                  16512

 dense_99 (Dense)            multiple                  1290

=================================================================
Total params: 734,922
Trainable params: 734,922
Non-trainable params: 0
_________________________________________________________________
```

Load and Preprocess MNIST Dataset

In [158…

```
def load_mnist():
    """Load training and testing mnist data."""

    def convert_types(image, label):
        image = tf.cast(image, tf.float32)
        image /= 255
        return image, label

    dataset, info = tfds.load(
        "mnist",
        with_info=True,
        as_supervised=True
    )

    mnist_train, mnist_test = dataset["train"], dataset["test"]
    mnist_train = mnist_train.map(convert_types).shuffle(10000).batch(128)
    mnist_test = mnist_test.map(convert_types).batch(128)

    return EasyDict(train=mnist_train, test=mnist_test)
```

Train and Evaluate CNN Architecture on Original Data and Adversarial Data

In [161…

```
nb_epochs = 8
eps = 0.3
adv_train = False #Use adversarial training (on PGD adversarial examples).

# Load training and test data
data = load_mnist()
model = CNN()
loss_object = tf.losses.SparseCategoricalCrossentropy(from_logits=True)
optimizer = tf.optimizers.Adam(learning_rate=0.001)

# Metrics to track the different accuracies.
train_loss = tf.metrics.Mean(name="train_loss")
test_acc_clean = tf.metrics.SparseCategoricalAccuracy()
test_acc_fgsm = tf.metrics.SparseCategoricalAccuracy()
test_acc_pgd = tf.metrics.SparseCategoricalAccuracy()
test_acc_bim = tf.metrics.SparseCategoricalAccuracy()
test_acc_cw = tf.metrics.SparseCategoricalAccuracy()


@tf.function
def train_step(x, y):
    with tf.GradientTape() as tape:
        predictions = model(x)
        loss = loss_object(y, predictions)
    gradients = tape.gradient(loss, model.trainable_variables)
    optimizer.apply_gradients(zip(gradients, model.trainable_variables))
    train_loss(loss)

# Train model with adversarial training
for epoch in range(nb_epochs):
    # keras like display of progress
    progress_bar_train = tf.keras.utils.Progbar(60000)
    for (x, y) in data.train:
        if adv_train:
            # Replace clean example with adversarial example for adversarial training
            x = projected_gradient_descent(model, x, eps, 0.01, 40, np.inf)
        train_step(x, y)
        progress_bar_train.add(x.shape[0], values=[("loss", train_loss.result())])

# Evaluate on clean and adversarial data
progress_bar_test = tf.keras.utils.Progbar(10000)
for x, y in data.test:
    y_pred = model(x)
    test_acc_clean(y, y_pred)

    x_fgm = fast_gradient_method(model, x, eps, np.inf)
    y_pred_fgm = model(x_fgm)
    test_acc_fgsm(y, y_pred_fgm)

    x_cw = carlini_wagner_l2(model, x, max_iterations=100,
                                       binary_search_steps=2,
                                       learning_rate=1e-2,
                                       initial_const=1,)
    y_pred_cw = model(x_cw)
    test_acc_cw(y, y_pred_cw)

    bim = basic_iterative_method(model, x, eps, 0.05, 10, np.inf)
    y_pred_bim = model(bim)
    test_acc_bim(y, y_pred_bim)



    progress_bar_test.add(x.shape[0])

# Displaying various metrics for evaluation
print("test acc on clean examples (%): {:.3f}".format(test_acc_clean.result() * 100))
y_pred = np.argmax(y_pred,1)
print("Precision", sk.metrics.precision_score(y, y_pred, average="macro"))
print("Recall", sk.metrics.recall_score(y, y_pred, average="macro"))
print("f1_score", sk.metrics.f1_score(y, y_pred, average="macro"))

print("test acc on FGM adversarial examples (%): {:.3f}".format(test_acc_fgsm.result() * 100))
y_pred_fgm = np.argmax(y_pred_fgm,1)
print("Precision", sk.metrics.precision_score(y, y_pred_fgm, average="macro"))
print("Recall", sk.metrics.recall_score(y, y_pred_fgm, average="macro"))
print("f1_score", sk.metrics.f1_score(y, y_pred_fgm, average="macro"))

print("test acc on CW adversarial examples (%): {:.3f}".format(test_acc_cw.result() * 100))
y_pred_cw = np.argmax(y_pred_cw,1)
print("Precision", sk.metrics.precision_score(y, y_pred_cw, average="macro"))
print("Recall", sk.metrics.recall_score(y, y_pred_cw, average="macro"))
print("f1_score", sk.metrics.f1_score(y, y_pred_cw, average="macro"))

print("test acc on BIM adversarial examples (%): {:.3f}".format(test_acc_bim.result() * 100))
y_pred_bim = np.argmax(y_pred_bim,1)
print("Precision", sk.metrics.precision_score(y, y_pred_bim, average="macro"))
print("Recall", sk.metrics.recall_score(y, y_pred_bim, average="macro"))
print("f1_score", sk.metrics.f1_score(y, y_pred_bim, average="macro"))
```

```
60000/60000 [==============================] - 5s 89us/step - loss: 0.4347
60000/60000 [==============================] - 4s 66us/step - loss: 0.1502
60000/60000 [==============================] - 4s 69us/step - loss: 0.1069
60000/60000 [==============================] - 4s 59us/step - loss: 0.0846
60000/60000 [==============================] - 3s 56us/step - loss: 0.0708
60000/60000 [==============================] - 3s 57us/step - loss: 0.0612
60000/60000 [==============================] - 3s 58us/step - loss: 0.0540
60000/60000 [==============================] - 3s 57us/step - loss: 0.0484
10000/10000 [==============================] - 249s 25ms/step
test acc on clean examples (%): 98.980
Precision 1.0
Recall 1.0
f1_score 1.0
test acc on FGM adversarial examples (%): 9.700
Precision 0.2333333333333333
Recall 0.275
f1_score 0.23333333333333334
test acc on CW adversarial examples (%): 66.240
Precision 0.8518518518518517
Recall 0.8611111111111112
f1_score 0.8322751322751323
test acc on BIM adversarial examples (%): 0.850
Precision 0.0
Recall 0.0
f1_score 0.0
```

In [ ]: