Q1) Classify Handwritten Digits from the MNIST Dataset

```
import numpy as npimport tensorflow as tfimport
matplotlib.pyplot as pltfrom tensorflow import keras
(x_train, y_train), (x_test, y_test) =
keras.datasets.mnist.load_data()
print(len(x_train))print(len(x_test))
6000010000
x_train[0].shape
(28, 28)
plt.matshow(x_train[2])
y_train[2]
x_train_flattened = x_train.reshape(len(x_train),
28*28)x_test_flattened = x_test.reshape(len(x_test),
28*28)
x_test_flattened.shape
model = keras.Sequential([keras.layers.Dense(10,
input_shape=(784,),
activation='sigmoid')])model.compile(
optimizer='adam',
loss='sparse_categorical_crossentropy',
metrics=['accuracy'])model.fit(x_train_flattened,
y_train, epochs=5)
model.evaluate(x_test_flattened, y_test)
model.predict(x_test_flattened)
y_predicted =
model.predict(x_test_flattened)y_predicted[0]
np.argmax(y_predicted[0])
y_predicted_labels=[np.argmax(i) for i in
y_predicted]y_predicted_labels[:5]
#implementing confusion matrixcm =
tf.math.confusion_matrix(labels=y_test,
predictions=y_predicted_labels)cm
import seaborn as snplt.figure(figsize=(10,
7))sn.heatmap(cm, annot=True,
fmt='d')plt.xlabel("predicted")plt.ylabel("True
values")plt.show()
x_train=x_train/255x_test=x_test/255
x_train_flattened = x_train.reshape(len(x_train),
28*28)x_test_flattened = x_test.reshape(len(x_test),
28*28)
model = keras.Sequential([keras.layers.Dense(10,
input_shape=(784,),
activation='sigmoid')])model.compile(
optimizer='adam',
loss='sparse_categorical_crossentropy',
metrics=['accuracy']
)model.fit(x_train_flattened, y_train, epochs=5)
model.evaluate(x_test_flattened, y_test)
model.predict(x_test_flattened)
y_predicted =
model.predict(x_test_flattened)y_predicted[0]
np.argmax(y_predicted[0])
y_predicted_labels=[np.argmax(i) for i in
y_predicted]y_predicted_labels[:5]
#implementing confusion matrixcm =
tf.math.confusion_matrix(labels=y_test,
predictions=y_predicted_labels)cm
import seaborn as snplt.figure(figsize=(10,
7))sn.heatmap(cm, annot=True,
fmt='d')plt.xlabel("predicted")plt.ylabel("True
values")plt.show()
model = keras.Sequential([
keras.layers.Flatten(input_shape=(28,28)),
keras.layers.Dense(100, activation='relu'),
keras.layers.Dense(10,
activation='sigmoid')])model.compile(
optimizer='adam',
loss='sparse_categorical_crossentropy',
metrics=['accuracy'])model.fit(x_train, y_train,
epochs=5)
model.evaluate(x_test, y_test)
```

Q2) CNN) for image classification

```
import pandas as pd
import numpy as np
import tensorflow as tf
from tensorflow.keras import layers, models
from sklearn.preprocessing import LabelBinarizer
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix,
ConfusionMatrixDisplay
# Load the CSV files
train_df = pd.read_csv('train.csv')
test_df = pd.read_csv('test.csv')
# Separate features and labels
X_train = train_df.drop(columns=['label']).values
y_train = train_df['label'].values
X_test = test_df.drop(columns=['label']).values
y_test = test_df['label'].values
# Reshape the data to fit the model (e.g., 28x28
images with 1 channel)
X_train = X_train.reshape(-1, 28, 28, 1)
X_test = X_test.reshape(-1, 28, 28, 1)
# Normalize the pixel values to be between 0 and 1
X_train = X_train / 255.0
X_test = X_test / 255.0
# Convert labels to one-hot encoding
lb = LabelBinarizer()
y_train = lb.fit_transform(y_train)
y_test = lb.transform(y_test)
# Build the CNN model
model = models.Sequential([
layers.Conv2D(32, (3, 3), activation='relu',
input_shape=(28, 28, 1)),
```

```
layers.MaxPooling2D((2, 2)),
layers.Conv2D(64, (3, 3), activation='relu'),
layers.MaxPooling2D((2, 2)),
layers.Conv2D(128, (3, 3), activation='relu'),
layers.Flatten(),
layers.Dense(128, activation='relu'),
layers.Dense(10, activation='softmax') # 10 classes for
Fashion-MNIST
])
# Compile the model
model.compile(optimizer='adam',
loss='categorical_crossentropy',
metrics=['accuracy'])
# Train the model
history = model.fit(X_train, y_train, epochs=10,
batch_size=64, validation_data=(X_test, y_test))
# Evaluate the model
test_loss, test_acc = model.evaluate(X_test, y_test)
print(f'Test accuracy: {test_acc:.4f}')
# Plot training & validation accuracy values
plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], 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('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()
# Predict the labels for the test data
y_pred = model.predict(X_test)
y_pred_classes = np.argmax(y_pred, axis=1)
y_true = np.argmax(y_test, axis=1)
# Compute the confusion matrix
conf_matrix = confusion_matrix(y_true,
y_pred_classes)
# Plot the confusion matrix
plt.figure(figsize=(10, 8))
cm_display =
ConfusionMatrixDisplay(confusion_matrix=conf_matri
x, display_labels=lb.classes_)
cm_display.plot(cmap=plt.cm.Blues,
values_format='d')
plt.title('Confusion Matrix')
plt.show()
# Display 10 random test images with their predicted
and actual labels
plt.figure(figsize=(15, 15))
for i in range(10):
plt.subplot(5, 2, i + 1)
plt.matshow(X_test[i].reshape(28, 28), fignum=False)
plt.title(f"True: {lb.classes_[y_true[i]]}, Pred:
{lb.classes_[y_pred_classes[i]]}")
plt.axis('off')
plt.show()
```

Q3) Hyper parameter tuning and regularization practice : Multilayer Perceptron (BPN), Mini-batch gradient descent.

```
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from keras.datasets import mnist
from keras.models import Sequential
from keras.utils import to_categorical
(X_train,y_train), (X_test,y_test) = mnist.load_data()
plt.imshow(X_train[7])
plt.show()
X_train.shape, y_train.shape, X_test.shape,
y_test.shape
X_train[0].shape
X_train = X_train.reshape((X_train.shape[0],-1))
X_test = X_test.reshape((X_test.shape[0],-1))
X_train, _, y_train,_ = train_test_split(X_train, y_train,
test_size = 0.67,random_state=7)
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)
print(X_train.shape, X_test.shape,y_train.shape,
y_test.shape)
from keras.models import Sequential
from keras.layers import Activation, Dense
from keras import optimizers
model = Sequential()
model.add(Dense(50,input_shape=(784,)))
model.add(Activation('sigmoid'))
model.add(Dense(50))
model.add(Activation('sigmoid'))
model.add(Dense(50))
model.add(Activation('sigmoid'))
model.add(Dense(50))
model.add(Activation('sigmoid'))
model.add(Dense(10))
model.add(Activation('softmax'))
sgd = optimizers.SGD(learning_rate = 0.001)
model.compile(optimizer = sgd, loss =
'categorical_crossentropy', metrics = ['accuracy'])
```

```
history = model.fit(X_train, y_train, batch_size = 256,
validation_split = 0.3, epochs = 100, verbose = 0)
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.legend(['training', 'validation'], loc='upper left')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.title('Model Accuracy')
plt.show()
results = model.evaluate(X_test,y_test)
print("Test accuracy : ", results[1])
"""## Using He_Normal Initializer"""
def mlp_model():
model = Sequential()
model.add(Dense(50, input_shape=(784,),
kernel_initializer = 'he_normal'))
model.add(Activation('sigmoid'))
model.add(Dense(50,kernel_initializer = 'he_normal'))
model.add(Activation('sigmoid'))
model.add(Dense(50,kernel_initializer = 'he_normal'))
model.add(Activation('sigmoid'))
model.add(Dense(50,kernel_initializer = 'he_normal'))
model.add(Activation('sigmoid'))
model.add(Dense(10,kernel_initializer = 'he_normal'))
model.add(Activation('softmax'))
sgd = optimizers.SGD(learning_rate = 0.001)
model.compile(optimizer = sgd, loss =
'categorical_crossentropy', metrics = ['accuracy'])
return model
model = mlp_model()
history = model.fit(X_train, y_train, validation_split =
0.3, epochs = 100, verbose = 0)
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.legend(['training', 'validation'], loc='upper left')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.title('Model Accuracy')
plt.show()
results = model.evaluate(X_test,y_test)
print("Test accuracy: ", results[1])
def mlp_model():
model = Sequential()
model.add(Dense(50, input_shape=(784,)))
model.add(Activation('elu'))
model.add(Dense(50))
model.add(Activation('elu'))
model.add(Dense(50))
model.add(Activation('elu'))
model.add(Dense(50))
model.add(Activation('elu'))
model.add(Dense(10))
model.add(Activation('softmax'))
adam = optimizers.Adam(learning_rate = 0.001)
model.compile(optimizer = adam, loss =
'categorical_crossentropy', metrics = ['accuracy'])
return model
model = mlp_model()
history = model.fit(X_train, y_train, validation_split =
0.3, epochs = 100, verbose = 0)
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.legend(['training', 'validation'], loc='upper left')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.title('Model Accuracy')
plt.show()
results = model.evaluate(X_test,y_test)
print("Test accuracy: ", results[1])
from keras.layers import BatchNormalization
def mlp_model():
model = Sequential()
model.add(Dense(50, input_shape=(784,)))
model.add(BatchNormalization())
model.add(Activation('elu'))
model.add(Dense(50))
model.add(BatchNormalization())
model.add(Activation('elu'))
model.add(Dense(50))
model.add(BatchNormalization())
model.add(Activation('elu'))
model.add(Dense(50))
model.add(BatchNormalization())
model.add(Activation('elu'))
model.add(Dense(10))
model.add(Activation('softmax'))
adam = optimizers.Adam(learning_rate = 0.001)
model.compile(optimizer = adam, loss =
'categorical_crossentropy', metrics = ['accuracy'])
return model
model = mlp_model()
history = model.fit(X_train, y_train, validation_split =
0.3, epochs = 100, verbose = 0)
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.legend(['training', 'validation'], loc='upper left')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.title('Model Accuracy')
plt.show()
results = model.evaluate(X_test,y_test)
print("Test accuracy: ", results[1])
from keras.layers import Dropout
def mlp_model():
model = Sequential()
model.add(Dense(50, input_shape=(784,)))
```

```
model.add(Activation('elu'))
model.add(Dropout(0.2))
model.add(Dense(50))
model.add(Activation('elu'))
model.add(Dropout(0.2))
model.add(Dense(50))
model.add(Activation('elu'))
model.add(Dropout(0.2))
model.add(Dense(50))
model.add(Activation('elu'))
model.add(Dropout(0.2))
model.add(Dense(10))
model.add(Activation('softmax'))
adam = optimizers.Adam(learning_rate = 0.001)
model.compile(optimizer = adam, loss =
'categorical_crossentropy', metrics = ['accuracy'])
return model
model = mlp_model()
history = model.fit(X_train, y_train, validation_split =
0.3, epochs = 100, verbose = 0)
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.legend(['training', 'validation'], loc='upper left')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.title('Model Accuracy')
plt.show()
results = model.evaluate(X_test,y_test)
print("Test accuracy: ", results[1])
```

Q4) Face Recognition Using CNN

```
import keras
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D,
Dense, Flatten, Dropout
from keras.optimizers import Adam
from keras.callbacks import TensorBoard
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score
from tensorflow.keras.utils import to_categorical #
Updated import
import itertools
# Load dataset
data = np.load('ORL_faces.npz')
# Load the "Train Images"
x_train = data['trainX']
# Normalize every image
x_train = np.array(x_train, dtype='float32') / 255
x_test = data['testX']
x_test = np.array(x_test, dtype='float32') / 255
# Load the Label of Images
y_train = data['trainY']
y_test = data['testY']
# Show the train and test data format
print('x_train: {}'.format(x_train[:]))
print('Y-train shape: {}'.format(y_train.shape))
print('x_test shape: {}'.format(x_test.shape))
x_train, x_valid, y_train, y_valid = train_test_split(
x_train, y_train, test_size=0.05, random_state=1234
)
im_rows = 112
im_cols = 92
batch_size = 512
im_shape = (im_rows, im_cols, 1)
# Change the size of images
x_train = x_train.reshape(x_train.shape[0],
*im_shape)
x_test = x_test.reshape(x_test.shape[0], *im_shape)
x_valid = x_valid.reshape(x_valid.shape[0],
*im_shape)
print('x_train shape: {}'.format(y_train.shape[0]))
print('x_test shape: {}'.format(y_test.shape))
# Create the CNN model
cnn_model = Sequential([
Conv2D(filters=36, kernel_size=7, activation='relu',
input_shape=im_shape),
MaxPooling2D(pool_size=2),
Conv2D(filters=54, kernel_size=5, activation='relu'),
MaxPooling2D(pool_size=2),
Flatten(),
Dense(2024, activation='relu'),
Dropout(0.5),
Dense(1024, activation='relu'),
Dropout(0.5),
Dense(512, activation='relu'),
Dropout(0.5),
Dense(20, activation='softmax')
])
cnn_model.compile(
loss='sparse_categorical_crossentropy',
optimizer=Adam(lr=0.0001),
metrics=['accuracy']
)
history = cnn_model.fit(
np.array(x_train),
np.array(y_train),
batch_size=512,
epochs=100,
verbose=2,
validation_data=(np.array(x_valid), np.array(y_valid))
)
```

```
scor = cnn_model.evaluate(np.array(x_test),
np.array(y_test), verbose=0)
print('Test loss {:.4f}'.format(scor[0]))
print('Test accuracy {:.4f}'.format(scor[1]))
# List all data in history
print(history.history.keys())
# Summarize history for accuracy
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()
# Summarize history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()
# Predicting the classes for the test set
predicted = cnn_model.predict(x_test)
ynew = np.argmax(predicted, axis=-1) # Updated to
replace predict_classes()
# Calculate accuracy
Acc = accuracy_score(y_test, ynew)
print("Accuracy: ")
print(Acc)
# Confusion matrix
cnf_matrix = confusion_matrix(np.array(y_test), ynew)
y_test1 = to_categorical(y_test, 20) # Updated to
replace np_utils.to_categorical
def plot_confusion_matrix(cm, classes,
normalize=False,
title='Confusion matrix',
cmap=plt.cm.Blues):
if normalize:
cm = cm.astype('float') / cm.sum(axis=1)[:,
np.newaxis]
# print("Normalized confusion matrix")
else:
print('Confusion matrix, without normalization')
plt.imshow(cm, interpolation='nearest', cmap=cmap)
plt.title(title)
plt.colorbar()
tick_marks = np.arange(len(classes))
plt.xticks(tick_marks, classes, rotation=45)
plt.yticks(tick_marks, classes)
fmt = '.2f' if normalize else 'd'
thresh = cm.max() / 2.
for i, j in itertools.product(range(cm.shape[0]),
range(cm.shape[1])):
plt.text(j, i, format(cm[i, j], fmt),
horizontalalignment="center",
color="white" if cm[i, j] > thresh else "black")
plt.tight_layout()
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.show()
print('Confusion matrix, without normalization')
print(cnf_matrix)
plt.figure()
plot_confusion_matrix(cnf_matrix[1:10,1:10],
classes=[0,1,2,3,4,5,6,7,8,9],
title='Confusion matrix, without normalization')
plt.figure()
plot_confusion_matrix(cnf_matrix[11:20,11:20],
classes=[10,11,12,13,14,15,16,17,18,19],
title='Confusion matrix, without normalization')
print("Confusion matrix: \n%s" %
confusion_matrix(np.array(y_test), ynew))
print(classification_report(np.array(y_test), ynew))
```

Q5) Text Processing Using-RNN

```
import tensorflow as tf
import tensorflow_datasets as tfds
import numpy as np
import matplotlib.pyplot as plt
# Obtain the imdb review dataset from tensorflow
datasets
dataset = tfds.load('imdb_reviews',
as_supervised=True)
# Separate test and train datasets
train_dataset, test_dataset = dataset['train'],
dataset['test']
# Define batch size and shuffle the training set
batch_size = 32
train_dataset = train_dataset.shuffle(10000)
train_dataset = train_dataset.batch(batch_size)
test_dataset = test_dataset.batch(batch_size)
# Get an example and its label
example, label = next(iter(train_dataset))
print('Text: \n', example.numpy()[0])
print('\nLabel: ', label.numpy()[0])
# Text vectorization layer
encoder =
tf.keras.layers.TextVectorization(max_tokens=10000)
encoder.adapt(train_dataset.map(lambda text, label:
text))
# Extracting the vocabulary from the TextVectorization
layer
vocabulary = np.array(encoder.get_vocabulary())
# Encoding a test example and decoding it back
```

```
original_text = example.numpy()[0]
encoded_text = encoder(original_text).numpy()
decoded_text = ' '.join(vocabulary[encoded_text])
print('Original: ', original_text)
print('Encoded: ', encoded_text)
print('Decoded: ', decoded_text)
# Creating the model
model = tf.keras.Sequential([
encoder,
tf.keras.layers.Embedding(len(encoder.get_vocabulary
()), 64, mask_zero=True),
tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(64,
return_sequences=True)),
tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(32)),
tf.keras.layers.Dense(64, activation='relu'),
tf.keras.layers.Dense(1)
])
# Summary of the model
model.summary()
# Compile the model
model.compile(
loss=tf.keras.losses.BinaryCrossentropy(from_logits=Tr
ue),
optimizer=tf.keras.optimizers.Adam(),
metrics=['accuracy']
)
# Train the model and validate on the test set
history = model.fit(
train_dataset,
epochs=5,
validation_data=test_dataset
)
# Plotting the accuracy and loss over time
history_dict = history.history
acc = history_dict['accuracy']
val_acc = history_dict['val_accuracy']
loss = history_dict['loss']
val_loss = history_dict['val_loss']
# Plotting
plt.figure(figsize=(8, 4))
plt.subplot(1, 2, 1)
plt.plot(acc)
plt.plot(val_acc)
plt.title('Training and Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend(['Accuracy', 'Validation Accuracy'])
plt.subplot(1, 2, 2)
plt.plot(loss)
plt.plot(val_loss)
plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend(['Loss', 'Validation Loss'])
plt.show()
# Making predictions
sample_text = (
'''The movie by GeeksforGeeks was so good and the
animation was amazing.
I would recommend my friends to watch it.'''
)
predictions = model.predict(np.array([sample_text]))
# Print the prediction and determine if it's positive or
negative
print(predictions[0])
if predictions[0] > 0:
print('The review is positive')
else:
print('The review is negative')
```

Q6) Image Generation Using GAN

```
from __future__ import absolute_import, division,
print_function,unicode_literals
import glob
import imageio
import matplotlib.pyplot as plt
import numpy as np
import os
import PIL
import tensorflow as tf
from tensorflow.keras import layers
import time
from IPython import display
# Load the MNIST Dataset
(train_images, train_labels), (_, _) =
tf.keras.datasets.mnist.load_data()
# Preprocess and normalize
train_images =
train_images.reshape(train_images.shape[0], 28, 28,
1).astype('float32')
train_images = (train_images - 127.5) / 127.5 #
Normalize to [-1, 1]
BUFFER_SIZE = 60000
BATCH_SIZE = 256
# Batch and shuffle the data
train_dataset =
tf.data.Dataset.from_tensor_slices(train_images).shuff
le(BUFFER_SIZE).batch(BATCH_SIZE)
def make_generator_model():
model = tf.keras.Sequential()
model.add(layers.Dense(7*7*256, use_bias=False,
input_shape=(100,)))
model.add(layers.BatchNormalization())
model.add(layers.LeakyReLU())
model.add(layers.Reshape((7, 7, 256)))
```

```
model.add(layers.Conv2DTranspose(128, (5, 5),
strides=(1, 1), padding='same', use_bias=False))
model.add(layers.BatchNormalization())
model.add(layers.LeakyReLU())
model.add(layers.Conv2DTranspose(64, (5, 5),
strides=(2, 2), padding='same', use_bias=False))
model.add(layers.BatchNormalization())
model.add(layers.LeakyReLU())
model.add(layers.Conv2DTranspose(1, (5, 5),
strides=(2, 2), padding='same', use_bias=False,
activation='tanh'))
return model
generator = make_generator_model()
generator.summary()
noise = tf.random.normal([1, 100])
print(noise)
generated_image = generator(noise, training=False)
plt.imshow(generated_image[0, :, :, 0], cmap='gray')
def make_discriminator_model():
model = tf.keras.Sequential()
model.add(layers.Conv2D(64, (5, 5), strides=(2, 2),
padding='same',
input_shape=[28, 28, 1]))
model.add(layers.LeakyReLU())
model.add(layers.Dropout(0.3))
model.add(layers.Conv2D(128, (5, 5), strides=(2, 2),
padding='same'))
model.add(layers.LeakyReLU())
model.add(layers.Dropout(0.3))
model.add(layers.Flatten())
model.add(layers.Dense(1))
return model
discriminator = make_discriminator_model()
discriminator.summary()
decision = discriminator(generated_image)
print (decision)
# This method returns a helper function to compute
cross entropy loss
cross_entropy =
tf.keras.losses.BinaryCrossentropy(from_logits=True)
def discriminator_loss(real_output, fake_output):
real_loss = cross_entropy(tf.ones_like(real_output),
real_output)
fake_loss = cross_entropy(tf.zeros_like(fake_output),
fake_output)
total_loss = real_loss + fake_loss
return total_loss
def generator_loss(fake_output):
return cross_entropy(tf.ones_like(fake_output),
fake_output)
generator_optimizer = tf.keras.optimizers.Adam(1e-4)
discriminator_optimizer =
tf.keras.optimizers.Adam(1e-4)
checkpoint_dir = './training_checkpoints'
checkpoint_prefix = os.path.join(checkpoint_dir,
"ckpt")
checkpoint =
tf.train.Checkpoint(generator_optimizer=generator_op
timizer,
discriminator_optimizer=discriminator_optimizer,
generator=generator,
discriminator=discriminator)
EPOCHS = 5
noise_dim = 100
num_examples_to_generate = 16
# We will reuse this seed overtime (so it's easier)
# to visualize progress in the animated GIF)
seed =
tf.random.normal([num_examples_to_generate,
noise_dim])
# Notice the use of `tf.function`
# This annotation causes the function to be
"compiled".
@tf.function
def train_step(images):
noise = tf.random.normal([BATCH_SIZE, noise_dim])
with tf.GradientTape() as gen_tape, tf.GradientTape()
as disc_tape:
generated_images = generator(noise, training=True)
real_output = discriminator(images, training=True)
fake_output = discriminator(generated_images,
training=True)
gen_loss = generator_loss(fake_output)
disc_loss = discriminator_loss(real_output,
fake_output)
gradients_of_generator = gen_tape.gradient(gen_loss,
generator.trainable_variables)
gradients_of_discriminator =
disc_tape.gradient(disc_loss,
discriminator.trainable_variables)
generator_optimizer.apply_gradients(zip(gradients_of
_generator, generator.trainable_variables))
discriminator_optimizer.apply_gradients(zip(gradients
_of_discriminator, discriminator.trainable_variables))
def train(dataset, epochs):
for epoch in range(epochs):
start = time.time()
for image_batch in dataset:
train_step(image_batch)
display.clear_output(wait=True)
generate_and_save_images(generator, epoch + 1,
seed)
if (epoch + 1) % 15 == 0:
checkpoint.save(file_prefix=checkpoint_prefix)
print('Time for epoch {} is {} sec'.format(epoch + 1,
time.time() - start))
```

```
display.clear_output(wait=True)
generate_and_save_images(generator, epochs, seed)
def generate_and_save_images(model, epoch,
test_input):
# Notice `training` is set to False.
# This is so all layers run in inference mode
(batchnorm).
predictions = model(test_input, training=False)
fig = plt.figure(figsize=(4,4))
for i in range(predictions.shape[0]): plt.subplot(4, 4,
i+1)
plt.imshow(predictions[i, :, :, 0] * 127.5 + 127.5,
cmap='gray')
plt.axis('off')
plt.savefig('image_at_epoch_{:04d}.png'.format(epoch
))
plt.show()
train(train_dataset, EPOCHS)
checkpoint.restore(tf.train.latest_checkpoint(checkpoi
nt_dir))
# Display a single image using the epoch number
def display_image(epoch_no):
return
PIL.Image.open('image_at_epoch_{:04d}.png'.format(
epoch_no))
EPOCH_NUM = 5
display_image(EPOCH_NUM)
import imageio
import glob
import IPython
# Create the GIF
anim_file = 'DCGAN_Animation.gif'
with imageio.get_writer(anim_file, mode='I') as
writer:
# Get all filenames matching the pattern
filenames = glob.glob('image*.png')
filenames = sorted(filenames) # Sort filenames to
maintain order
for i, filename in enumerate(filenames):
# Read the image
image = imageio.imread(filename)
writer.append_data(image)
# Display the GIF in Jupyter Notebook if applicable
if IPython.version_info > (6, 2, 0, ''):
display.Image(filename=anim_file)
# Everytime it will generate new data
noise = tf.random.normal([1, 100])
generated_image = generator(noise, training=False)
plt.imshow(generated_image[0, :, :, 0], cmap='gray')
```