GENERATIVE ARTIFICIAL INTELLIGENCE

AIML303

Practical file

AMITY SCHOOL OF ENGINEERING AND TECHNOLOGY

AMITY UNIVERSITY, UTTAR PRADESH

Submitted To: Dr. Ritesh Maurya

Submitted By: Abha Ghildiyal

Enrollment no. A2305221533

INDEX

S.no.	Program	Date	Sign
1.	Implement perceptron using Pytorch		
2.	Implement MLP using Pytorch		
3.	Implement MLP for XOR gate using Pytorch		
4.	Implement simple NN for MNIST digit classification using Pytorch.		
5.	Implement simple NN for MNIST digit classification using Keras and TF		
6.	Implement CNN using Keras for CIFAR10 classification		
7.	Transfer Learning for sugarcane disease classification		
8.	Image denoising using autoencoder		
9.	Image generation using convolutional GAN		
10.	Simple audio recoginition		
11.	Text generation using RNN		

EXPERIMENT - 1

Aim:

Implement perceptron using pytorch.

Experiment Conducted:

```
import numpy as np
```

```
#define perceptron parameters
input_size = 2
weights = np.random.rand(input_size)
bias = np.random.rand(1)
# learning_rate = 0.01
```

```
#define the step function
def step_function(x): return 1 if x > 0 else 0
```

```
# define the perceptron function
def perceptron(inputs,weights,bias):
  linear_combination = np.dot(inputs, weights) + bias
  output = step_function(linear_combination)
  return output
```

```
#generate random inputs
inputs = np.random.rand(input_size)
```

```
#apply the perceptron
output = perceptron(inputs,weights,bias)
```

```
#display results
print("Inputs:",inputs)
print("Weights:",weights)
print("Bias:",bias)
print("output",output)
```

Output:

```
Inputs: [0.90863384 0.6993004 ]
Weights: [0.18947875 0.87561118]
Bias: [0.03618943]
output 1
```

Conclusion: Hence, the experiment is conducted successfully and the results have been verified.

EXPERIMENT - 2

Aim:

Implement MLP using Pytorch.

Experiment Conducted:

```python
import torch
import torch.nn as nn
import torch.optim as optim
```

```python
#Define the neural network architecture
class MLP(nn.Module):
  def __init__(self, input_size, hidden_size, output_size):
    super(MLP, self).__init__()
    self.fc1 = nn.Linear(input_size,hidden_size)
    self.relu = nn.ReLU()
    self.fc2 = nn.Linear(hidden_size, output_size)

  def forward(self,x):
    x=self.fc1(x)
    x=self.relu(x)
    x=self.fc2(x)
    return x
```

```python
# set random seed for reproducibility
torch.manual_seed(42)
```

```
<torch._C.Generator at 0x7959681fd6b0>
```

```python
# Instantiate the model
input_size = 10
hidden_size = 20
output_size = 1
model = MLP(input_size,hidden_size,output_size)
```

```python
# Define the loss function and optimizer
criterion = nn.MSELoss()
optimizer = optim.SGD(model.parameters(), lr=0.5)
```

```python
#dummy input data and arget
input_data=torch.randn(100,input_size)
print("input_data_shape",input_data.shape)
target=torch.randn(100,output_size)
#print(target)
```

```
input_data_shape torch.Size([100, 10])
```

```python
#Training loop
num_epochs=4000
for epoch in range(num_epochs):
  #Forward pass
  output= model(input_data)
  loss= criterion(output,target)
```

```python
#Backward pass and optimization
optimizer.zero_grad()
#Zero the gradients to avoid accumulation
loss.backward()
#Backpropagation
optimizer.step()
#Update weights
#Print the loss every 100 epochs
if (epoch +1)%100==0:
  print(f'Epoch[{epoch +1}/{num_epochs}],Loss:{loss.item():.4f}')
```

```
Epoch[4000/4000],Loss:1.1767
```

```python
#Test the trained model
test_input=torch.randn(1,input_size)
print("test_input",test_input.shape)
with torch.no_grad():
  predicted_output=model(test_input)
  print(f'Test Input:{test_input.numpy()}')
  print(f'Predicted Output:{predicted_output.numpy()}')
```

Output:

```
test_input torch.Size([1, 10])
Test Input:[[ 0.77188647  0.69540155  1.0965257  -0.00797612  1.3109813   0.25582278
   0.24653137 -0.421456   -0.86095834  0.56766146]]
Predicted Output:[[0.2269126]]
```

Conclusion: Hence, the experiment is conducted successfully and the results have been verified.

EXPERIMENT - 3

Aim:

Implement MLP for XOR gate using PyTorch.

Experiment Conducted:

```
import torch
import torch.nn as nn
import torch.optim as optim
from torchvision import datasets, transforms
from torch.utils.data import DataLoader
```

```
[ ] # Define a simple neural network
class SimpleNN(nn.Module):
  def __init__(self):
    super(SimpleNN, self).__init__()
    self.fc1 = nn.Linear(28*28,128)
    self.relu=nn.ReLU()
    self.fc2=nn.Linear(128,10)

  def forward(self,x):
    x=x.view(-1,28*28)
      #print("shape",x.shape)
    x=self.fc1(x)
    x=self.relu(x)
    x=self.fc2(x)
    return x
```

```
# Download and load the MNIST dataset
transform=transforms.Compose([transforms.ToTensor(),transforms.Normalize((0.5,),(0.5,))])

train_dataset=datasets.MNIST(root='./data',train=True,transform=transform,download=True)
test_dataset=datasets.MNIST(root='./data',train=False,transform=transform,download=True)

# Print the sizes of the train and test datasets
print(f"Train dataset size:{len(train_dataset)}")
print(f"Test dataset size:{len(test_dataset)}")
```

```
Train dataset size:60000
Test dataset size:10000
```

```
[ ] train_loader = DataLoader(dataset=train_dataset,batch_size=64,shuffle=True)
    test_loader = DataLoader(dataset=test_dataset,batch_size=64,shuffle=False)
```

```
[ ] # Initialize the model,loss function, and optimizer
model=SimpleNN()
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(),lr=0.001)
```

```
# Training loop
num_epochs=10

for epoch in range(num_epochs):
  for i,(images,labels) in enumerate(train_loader):
    optimizer.zero_grad()
    outputs=model(images)
    loss=criterion(outputs,labels)
    loss.backward()
    optimizer.step()

    if(i+1)%100==0:
      print(f'Epoch[{epoch+1}/{num_epochs}],Step[{i+1}/{len(train_loader)}],Loss:{loss.item():.4f}')
```

```
Epoch[10/10],Step[800/938],Loss:0.0824
Epoch[10/10],Step[900/938],Loss:0.0109
```

```
# Testing the model
model.eval()
correct,total=0,0

with torch.no_grad():
  for images,labels in test_loader:
    outputs=model(images)
    _, predicted = torch.max(outputs.data,1)
    total+=labels.size(0)
    correct+=(predicted==labels).sum().item()

accuracy = correct/total
print(f'Test Accuracy:{accuracy*100:.2f}%')
```

Output:

```
Test Accuracy:97.30%
```

Conclusion: Hence, the experiment is conducted successfully and the results have been verified.

EXPERIMENT - 4

Aim:

Implement simple NN for MNIST digit classification using Pytorch.

Experiment Conducted:

```python
import torch
import torch.nn as nn
import torch.optim as optim
from torchvision import datasets,transforms
from torch.utils.data import DataLoader
from sklearn.metrics import confusion_matrix, classification_report
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

```python
# set device(use GPU if available)
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
```

```python
# Define a simple neural network
class SimpleNN(nn.Module):
    def __init__(self):
        super(SimpleNN, self).__init__()
        self.fc1 = nn.Linear(28*28, 128)
        self.relu = nn.ReLU()
        self.fc2 = nn.Linear(128, 10)

    def forward(self, x):
        x = x.view(-1, 28*28)
        x = self.fc1(x)
        x = self.relu(x)
        x = self.fc2(x)
        return x
```

```python
# Download and load the MNIST dataset
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5,), (0.5,))
])

train_dataset = datasets.MNIST(root='./data', train=True, transform=transform, download=True)
test_dataset = datasets.MNIST(root='./data', train=False, transform=transform, download=True)

train_loader = DataLoader(dataset=train_dataset, batch_size=64, shuffle=True)
test_loader = DataLoader(dataset=test_dataset, batch_size=64, shuffle=False)
```

```python
# Initialize the model, loss function, and optimizer
model = SimpleNN().to(device)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)
```

```python
# Training loop
num_epochs = 5

for epoch in range(num_epochs):
    for i, (images, labels) in enumerate(train_loader):
        images, labels = images.to(device), labels.to(device)
        optimizer.zero_grad()
        outputs = model(images)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        if (i+1) % 100 == 0:
            print(f'Epoch [{epoch+1}/{num_epochs}], Step [{i+1}/{len(train_loader)}], Loss: {loss.item():.4f}')
```

```
Epoch [5/5], Step [800/938], Loss: 0.0853
Epoch [5/5], Step [900/938], Loss: 0.1098
```

```python
# Testing the model
model.eval()
y_true, y_pred = [], []

with torch.no_grad():
    for images, labels in test_loader:
        images, labels = images.to(device), labels.to(device)
        outputs = model(images)
        _, predicted = torch.max(outputs.data, 1)
        y_true.extend(labels.cpu().numpy())
        y_pred.extend(predicted.cpu().numpy())
```

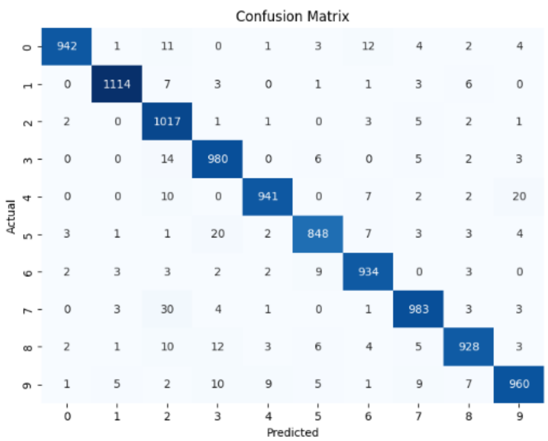
```python
# Convert lists to NumPy arrays
y_true = np.array(y_true)
y_pred = np.array(y_pred)
```

```python
# Create a confusion matrix
conf_matrix = confusion_matrix(y_true, y_pred)
```

```python
# Plot the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues", cbar=False,
            xticklabels=np.arange(10), yticklabels=np.arange(10))
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix")
plt.show()
```

```python
# Print classification report
print("Classification Report:")
print(classification_report(y_true, y_pred, target_names=[str(i) for i in range(10)]))
```

Output:




```
Classification Report:
              precision    recall  f1-score   support

           0       0.99      0.96      0.98       980
           1       0.99      0.98      0.98      1135
           2       0.92      0.99      0.95      1032
           3       0.95      0.97      0.96      1010
           4       0.98      0.96      0.97       982
           5       0.97      0.95      0.96       892
           6       0.96      0.97      0.97       958
           7       0.96      0.96      0.96      1028
           8       0.97      0.95      0.96       974
           9       0.96      0.95      0.96      1009

    accuracy                           0.96     10000
   macro avg       0.97      0.96      0.96     10000
weighted avg       0.97      0.96      0.96     10000
```

Conclusion: Hence, the experiment is conducted successfully and the results have been verified.

EXPERIMENT - 5

Aim:

Implement simple NN for MNIST digit classification using Keras and TF.

Experiment Conducted:

```
# Importing necessary libraries
import numpy as np
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.utils import to_categorical
```

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

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 [==============================] - 0s 0us/step
```

```
# Normalize pixel values to be between 0 and 1
x_train = x_train / 255.0
x_test = x_test / 255.0
```

```
# Flatten the 28x28 images into 1D arrays
x_train = x_train.reshape((x_train.shape[0], -1))
x_test = x_test.reshape((x_test.shape[0], -1))
```

```
# One-hot encode the target labels
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)
```

```
# Build the MLP model
model = Sequential()
model.add(Dense(128, input_shape=(784,), activation='relu'))
model.add(Dense(64, activation='relu'))
model.add(Dense(10, activation='softmax'))
```

```
# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

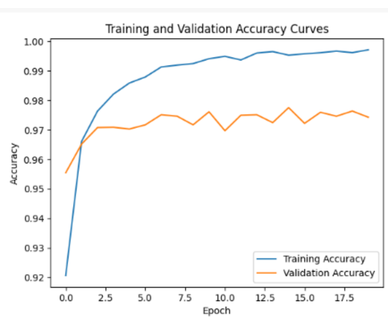
```
# Train the model
history=model.fit(x_train, y_train, epochs=20, batch_size=32, validation_split=0.2)
```

```
# Evaluate the model on the test set
loss, accuracy = model.evaluate(x_test, y_test)
print(f'Test Accuracy: {accuracy * 100:.2f}%')
```

```
# Plot training and validation accuracy curves
import matplotlib.pyplot as plt

plt.plot(history.history['accuracy'],label='Training Accuracy')
plt.plot(history.history['val_accuracy'],label='Validation Accuracy')
plt.title('Training and Validation Accuracy Curves')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```

Output:

```
313/313 [==============================] - 1s 2ms/step - loss: 0.1188 - accuracy: 0.9778
Test Accuracy: 97.78%
```



Conclusion: Hence, the experiment is conducted successfully and the results have been verified.

EXPERIMENT - 6

Aim:

Implement CNN using Keras for CIFAR10 classification.

Experiment Conducted:

```python
import tensorflow as tf
from tensorflow.keras import datasets, layers, models, optimizers
from tensorflow.keras.applications import VGG16
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Load CIFAR-10 dataset
(train_images, train_labels), (test_images, test_labels) = datasets.cifar10.load_data()

# Normalize pixel values to be between 0 and 1
train_images, test_images = train_images / 255.0, test_images / 255.0

# Define data augmentation
train_datagen = ImageDataGenerator(
    rotation_range=15,
    width_shift_range=0.1,
    height_shift_range=0.1,
    horizontal_flip=True,
    )
train_datagen.fit(train_images)

# Load pre-trained VGG16 model without including top dense layers
base_model = VGG16(weights='imagenet', include_top=False, input_shape=(32, 32, 3))

# Freeze all layers in base model
for layer in base_model.layers:
    layer.trainable = False
```

```python
# Add custom dense layers for CIFAR-10 classification
model = models.Sequential()
model.add(base_model)
model.add(layers.Flatten())
model.add(layers.Dense(256, activation='relu'))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(10, activation='softmax'))

# Compile the model
model.compile(optimizer=optimizers.Adam(),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

# Train the model
history = model.fit(train_datagen.flow(train_images, train_labels, batch_size=64),
                    steps_per_epoch=len(train_images) / 64, epochs=10,
                    validation_data=(test_images, test_labels))

# Evaluate the model
test_loss, test_acc = model.evaluate(test_images, test_labels)
print(f'Test accuracy: {test_acc}')
```

Output:

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071 [==============================] - 95s 1us/step
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5
58889256/58889256 [==============================] - 4s 0us/step
Epoch 1/10
781/781 [==============================] - 44s 50ms/step - loss: 1.6040 - accuracy: 0.1001 - val_loss: 1.3276 - val_accuracy: 0.1136
Epoch 2/10
781/781 [==============================] - 35s 45ms/step - loss: 1.4076 - accuracy: 0.0984 - val_loss: 1.2594 - val_accuracy: 0.1031
Epoch 3/10
781/781 [==============================] - 35s 44ms/step - loss: 1.3603 - accuracy: 0.0968 - val_loss: 1.2182 - val_accuracy: 0.1246
Epoch 4/10
781/781 [==============================] - 34s 44ms/step - loss: 1.3256 - accuracy: 0.0981 - val_loss: 1.2021 - val_accuracy: 0.0943
Epoch 5/10
781/781 [==============================] - 35s 44ms/step - loss: 1.3104 - accuracy: 0.0970 - val_loss: 1.2014 - val_accuracy: 0.1000
Epoch 6/10
781/781 [==============================] - 35s 45ms/step - loss: 1.3005 - accuracy: 0.0978 - val_loss: 1.1776 - val_accuracy: 0.0984
Epoch 7/10
781/781 [==============================] - 35s 45ms/step - loss: 1.2816 - accuracy: 0.0977 - val_loss: 1.1735 - val_accuracy: 0.1187
Epoch 8/10
781/781 [==============================] - 35s 45ms/step - loss: 1.2731 - accuracy: 0.0963 - val_loss: 1.1796 - val_accuracy: 0.1122
Epoch 9/10
781/781 [==============================] - 36s 46ms/step - loss: 1.2657 - accuracy: 0.0994 - val_loss: 1.1634 - val_accuracy: 0.0895
Epoch 10/10
781/781 [==============================] - 36s 45ms/step - loss: 1.2594 - accuracy: 0.0977 - val_loss: 1.1484 - val_accuracy: 0.1020
313/313 [==============================] - 3s 9ms/step - loss: 1.1484 - accuracy: 0.1020
Test accuracy: 0.10199999809265137
```

Conclusion: Hence, the experiment is conducted successfully and the results have been verified.

EXPERIMENT - 7

Aim:

Transfer Learning for sugarcane disease classification.

Experiment Conducted:

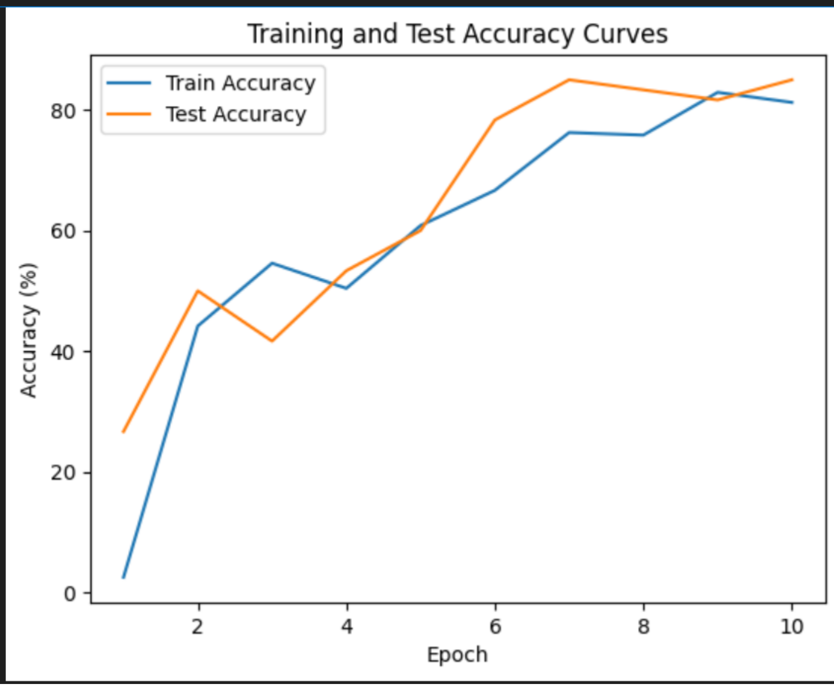
```
import os
import torch
import torchvision.transforms as transforms
import torchvision.datasets as datasets
import torchvision.models as models
import torch.optim as optim
import torch.nn as nn
import matplotlib.pyplot as plt

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

transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
])

data_folder = '/kaggle/input/sugarcane-disease-dataset/sugarcane RA/'

dataset = datasets.ImageFolder(root=data_folder, transform=transform)

train_size = int(0.8 * len(dataset))
test_size = len(dataset) - train_size
train_dataset, test_dataset = torch.utils.data.random_split(dataset, [train_size,
test_size])

num_classes = len(dataset.classes)
model = models.vgg16(pretrained=True)
model.fc = nn.Linear(4096, num_classes)  # VGG16 has 4096 output features
model = model.to(device)

criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(), lr=0.001, momentum=0.9)

batch_size = 64
trainloader = torch.utils.data.DataLoader(train_dataset, batch_size=batch_size,
shuffle=True, num_workers=4)
testloader = torch.utils.data.DataLoader(test_dataset, batch_size=batch_size,
shuffle=False, num_workers=4)

num_epochs = 10
train_acc_history = []
test_acc_history = []
```

```python
for epoch in range(num_epochs):
    model.train()
    running_loss = 0.0
    total_train = 0
    correct_train = 0

    for i, (inputs, labels) in enumerate(trainloader):
        inputs, labels = inputs.to(device), labels.to(device)

        optimizer.zero_grad()

        outputs = model(inputs)
        loss = criterion(outputs, labels)

        loss.backward()
        optimizer.step()

        running_loss += loss.item()
        _, predicted_train = torch.max(outputs.data, 1)
        total_train += labels.size(0)
        correct_train += (predicted_train == labels).sum().item()

    train_accuracy = 100 * correct_train / total_train
    train_acc_history.append(train_accuracy)

    model.eval()
    correct_test = 0
    total_test = 0

    with torch.no_grad():
        for inputs, labels in testloader:
            inputs, labels = inputs.to(device), labels.to(device)
            outputs = model(inputs)
            _, predicted_test = torch.max(outputs.data, 1)
            total_test += labels.size(0)
            correct_test += (predicted_test == labels).sum().item()

    test_accuracy = 100 * correct_test / total_test
    test_acc_history.append(test_accuracy)

    print(f'Epoch [{epoch+1}/{num_epochs}], Train Accuracy: {train_accuracy:.2f}%, Test 
Accuracy: {test_accuracy:.2f}%')

# Plotting accuracy curves
epochs = range(1, num_epochs + 1)
plt.plot(epochs, train_acc_history, label='Train Accuracy')
plt.plot(epochs, test_acc_history, label='Test Accuracy')
plt.title('Training and Test Accuracy Curves')
plt.xlabel('Epoch')
```

```python
plt.ylabel('Accuracy (%)')
plt.legend()
plt.show()
```

Output:

```
Epoch [10/10], Train Accuracy: 81.25%, Test Accuracy: 85.00%
```



Conclusion: Hence, the experiment is conducted successfully and the results have been verified.

EXPERIMENT - 8

Aim:

Image denoising using autoencoder.

Experiment Conducted:

```
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
```

```
[ ] # Load MNIST dataset
(x_train, _), (x_test, _) = tf.keras.datasets.mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 [==============================] - 0s 0us/step
```

```
[ ] # Normalize and reshape the data
x_train = x_train.astype('float32') / 255.
x_test = x_test.astype('float32') / 255.
x_train = np.reshape(x_train, (len(x_train), 28, 28, 1))
x_test = np.reshape(x_test, (len(x_test), 28, 28, 1))
```

```
[ ] # Add Gaussian noise to the images
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.)
```

```
[ ] # Define the autoencoder model
input_img = tf.keras.layers.Input(shape=(28, 28, 1))
```

```
[ ] # Encoder
x = tf.keras.layers.Conv2D(32, (3, 3), activation='relu', padding='same')(input_img)
x = tf.keras.layers.MaxPooling2D((2, 2), padding='same')(x)
x = tf.keras.layers.Conv2D(32, (3, 3), activation='relu', padding='same')(x)
encoded = tf.keras.layers.MaxPooling2D((2, 2), padding='same')(x)
```

```
[ ] # Decoder
x = tf.keras.layers.Conv2D(32, (3, 3), activation='relu', padding='same')(encoded)
x = tf.keras.layers.UpSampling2D((2, 2))(x)
x = tf.keras.layers.Conv2D(32, (3, 3), activation='relu', padding='same')(x)
x = tf.keras.layers.UpSampling2D((2, 2))(x)
decoded = tf.keras.layers.Conv2D(1, (3, 3), activation='sigmoid', padding='same')(x)

autoencoder = tf.keras.models.Model(input_img, decoded)
autoencoder.compile(optimizer='adam', loss='binary_crossentropy')
```

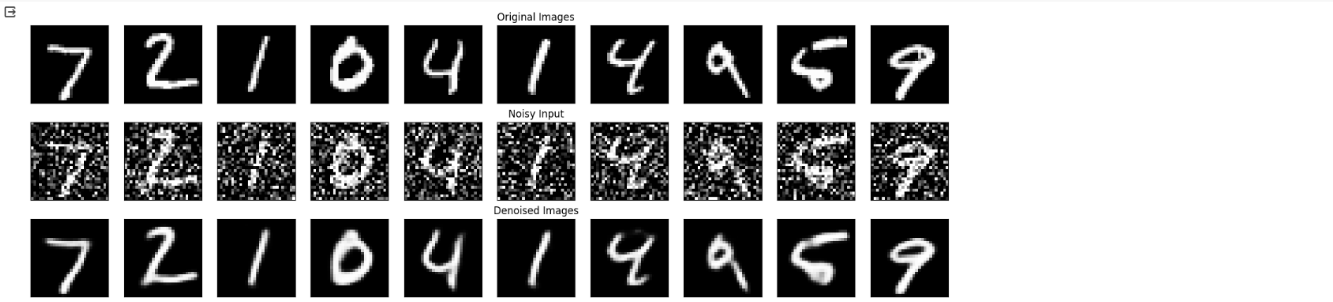
```
[ ] # Train the autoencoder
autoencoder.fit(x_train_noisy, x_train,
                epochs=50,
                batch_size=128,
                shuffle=True,
                validation_data=(x_test_noisy, x_test))
```

```
[ ] # Predict the denoised images
decoded_imgs = autoencoder.predict(x_test_noisy)
```

```
313/313 [==============================] - 1s 2ms/step
```

```
# Plot the original, noisy, and denoised images
n = 10
plt.figure(figsize=(20, 6))
for i in range(n):
    # Original images
    ax = plt.subplot(3, 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)
    if i == n // 2:
        ax.set_title('Original Images')

    # Noisy images
    ax = plt.subplot(3, n, i + 1 + n)
    plt.imshow(x_test_noisy[i].reshape(28, 28))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)
    if i == n // 2:
        ax.set_title('Noisy Input')

    # Denoised images
    ax = plt.subplot(3, n, i + 1 + 2 * n)
    plt.imshow(decoded_imgs[i].reshape(28, 28))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)
    if i == n // 2:
        ax.set_title('Denoised Images')
plt.show()
print(autoencoder.summary())
```

Output:




```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 28, 28, 1)]       0

 conv2d (Conv2D)             (None, 28, 28, 32)        320

 max_pooling2d (MaxPooling2  (None, 14, 14, 32)        0
 D)

 conv2d_1 (Conv2D)           (None, 14, 14, 32)        9248

 max_pooling2d_1 (MaxPoolin  (None, 7, 7, 32)          0
 g2D)

 conv2d_2 (Conv2D)           (None, 7, 7, 32)          9248

 up_sampling2d (UpSampling2  (None, 14, 14, 32)        0
 D)

 conv2d_3 (Conv2D)           (None, 14, 14, 32)        9248

 up_sampling2d_1 (UpSamplin  (None, 28, 28, 32)        0
 g2D)

 conv2d_4 (Conv2D)           (None, 28, 28, 1)         289

=================================================================
Total params: 28353 (110.75 KB)
Trainable params: 28353 (110.75 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
None
```

Conclusion: Hence, the experiment is conducted successfully and the results have been verified.

EXPERIMENT - 9

Aim:

Image generation using convolutional GAN.

Experiment Conducted:

```
import numpy as np
import matplotlib.pyplot as plt
from keras.models import Sequential, Model
from keras.layers import Input, Dense, Reshape, Flatten, Conv2D, Conv2DTranspose, LeakyReLU
from keras.optimizers import Adam
from keras.datasets import mnist
```

```
# Load MNIST data
(X_train, _), (x_test, _) = mnist.load_data()

# Normalize data
X_train = (X_train.astype(np.float32) - 127.5) / 127.5
X_train = np.expand_dims(X_train, axis=3)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 [==============================] - 0s 0us/step
```

```
# Define generator
def build_generator():
    model = Sequential()
    model.add(Dense(7 * 7 * 128, input_dim=100))
    model.add(LeakyReLU(0.2))
    model.add(Reshape((7, 7, 128)))
    model.add(Conv2DTranspose(64, kernel_size=4, strides=2, padding='same'))
    model.add(LeakyReLU(0.2))
    model.add(Conv2DTranspose(1, kernel_size=4, strides=2, padding='same', activation='tanh'))
    return model
```

```
# Define discriminator
def build_discriminator():
    model = Sequential()
    model.add(Conv2D(64, kernel_size=4, strides=2, padding='same', input_shape=(28, 28, 1)))
    model.add(LeakyReLU(0.2))
    model.add(Conv2D(128, kernel_size=4, strides=2, padding='same'))
    model.add(LeakyReLU(0.2))
    model.add(Flatten())
    model.add(Dense(1, activation='sigmoid'))
    return model
```

```
# Compile discriminator
discriminator = build_discriminator()
discriminator.compile(loss='binary_crossentropy', optimizer=Adam(lr=0.0002, beta_1=0.5), metrics=['accuracy'])
```

```
WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learning_rate` or use the legacy optimizer, e.g.,tf.keras.optimizers.legacy.Adam.
```

```
# Combine generator and discriminator into a single model
generator = build_generator()
z = Input(shape=(100,))
img = generator(z)
discriminator.trainable = False
validity = discriminator(img)
gan = Model(z, validity)
gan.compile(loss='binary_crossentropy', optimizer=Adam(lr=0.0002, beta_1=0.5))
```

```
WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learning_rate` or use the legacy optimizer, e.g.,tf.keras.optimizers.legacy.Adam.
```

```
# Train DCGAN
epochs = 1000
batch_size = 64
for epoch in range(epochs):
    # Select a random batch of images
    idx = np.random.randint(0, X_train.shape[0], batch_size)
    real_images = X_train[idx]
    # Generate fake images
    noise = np.random.normal(0, 1, (batch_size, 100))
    fake_images = generator.predict(noise)
    # Train discriminator
    d_loss_real = discriminator.train_on_batch(real_images, np.ones(batch_size))
    d_loss_fake = discriminator.train_on_batch(fake_images, np.zeros(batch_size))
    d_loss = 0.5 * np.add(d_loss_real, d_loss_fake)
    # Train generator
    noise = np.random.normal(0, 1, (batch_size, 100))
    g_loss = gan.train_on_batch(noise, np.ones(batch_size))
    # Print progress
    if epoch % 100 == 0:
        print(f"Epoch: {epoch} \t Discriminator Loss: {d_loss[0]} \t Generator Loss: {g_loss}")
# Generate images
noise = np.random.normal(0, 1, (100, 100))
generated_images = generator.predict(noise)
```

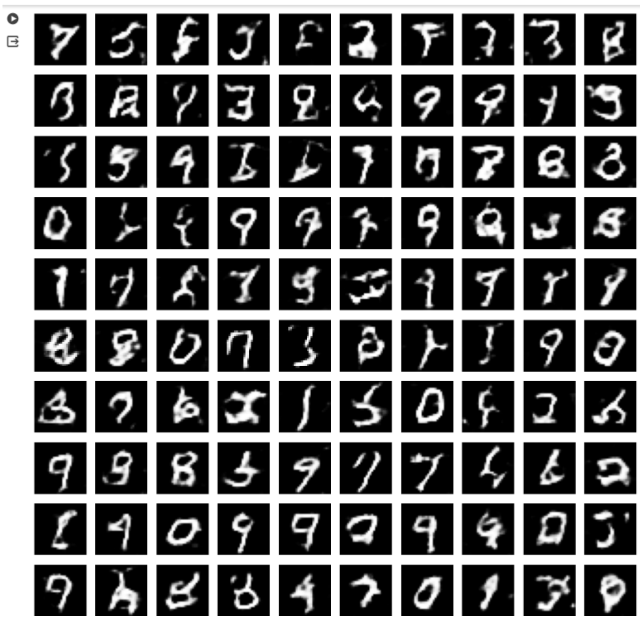
```
2/2 [==============================] - 1s 7ms/step
Epoch: 0         Discriminator Loss: 0.6859554648399353         Generator Loss: 0.684131383895874
2/2 [==============================] - 0s 5ms/step
2/2 [==============================] - 0s 4ms/step
```

```
2/2 [==============================] - 0s 4ms/step
2/2 [==============================] - 0s 5ms/step
2/2 [==============================] - 0s 4ms/step
4/4 [==============================] - 0s 30ms/step
```

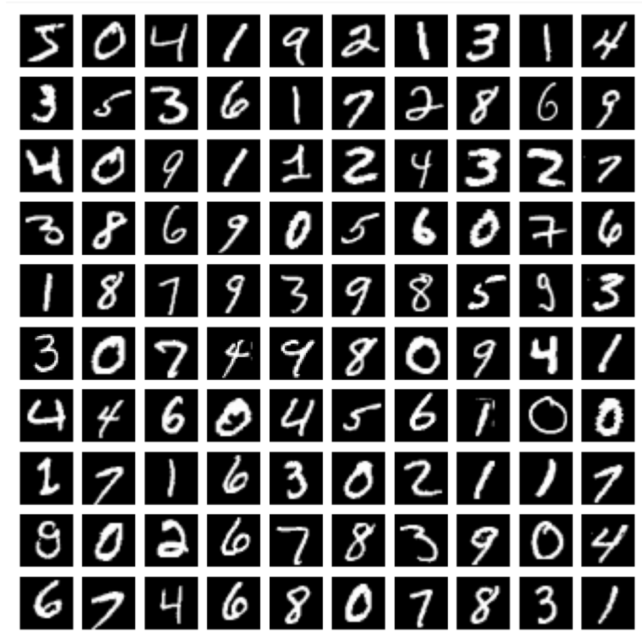
```
# Display generated images
plt.figure(figsize=(10, 10))
for i in range(100):
    plt.subplot(10, 10, i+1)
    plt.imshow(generated_images[i, :, :, 0], cmap='gray')
    plt.axis('off')
plt.tight_layout()
plt.show()
```

```python
# Visualize original images
plt.figure(figsize=(10, 10))
for i in range(100):
    plt.subplot(10, 10, i+1)
    plt.imshow(X_train[i, :, :, 0], cmap='gray')
    plt.axis('off')
plt.tight_layout()
plt.show()
```

Output:



(Generated images)



(Original images)

Conclusion: Hence, the experiment is conducted successfully and the results have been verified.

EXPERIMENT - 10

Aim:

Simple Audio recognition.

Experiment Conducted:

```
[1] pip install -U -q tensorflow tensorflow_datasets

    ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 589.8/589.8 MB 1.1 MB/s eta 0:00:00
    ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 4.8/4.8 MB 86.7 MB/s eta 0:00:00
    ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.2/2.2 MB 80.2 MB/s eta 0:00:00
    ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 5.5/5.5 MB 74.9 MB/s eta 0:00:00
    ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.0/1.0 MB 63.8 MB/s eta 0:00:00
ERROR: pip's dependency resolver does not currently take into account all the packages that are installed. This behaviour is the source of the following dependency conflicts.
tf-keras 2.15.1 requires tensorflow<2.16,>=2.15, but you have tensorflow 2.16.1 which is incompatible.
```

```python
import os
import pathlib

import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
import tensorflow as tf

from tensorflow.keras import layers
from tensorflow.keras import models
from IPython import display

# Set the seed value for experiment reproducibility.
seed = 42
tf.random.set_seed(seed)
np.random.seed(seed)
```

```python
[3] DATASET_PATH = 'data/mini_speech_commands'

data_dir = pathlib.Path(DATASET_PATH)
if not data_dir.exists():
  tf.keras.utils.get_file(
      'mini_speech_commands.zip',
      origin="http://storage.googleapis.com/download.tensorflow.org/data/mini_speech_commands.zip",
      extract=True,
      cache_dir='.', cache_subdir='data')
```

```
Downloading data from http://storage.googleapis.com/download.tensorflow.org/data/mini_speech_commands.zip
182082353/182082353 ━━━━━━━━━━━━━━━━━━━━ 2s 0us/step
```

```python
[4] commands = np.array(tf.io.gfile.listdir(str(data_dir)))
commands = commands[(commands != 'README.md') & (commands != '.DS_Store')]
print('Commands:', commands)
```

```
Commands: ['right' 'yes' 'up' 'left' 'go' 'no' 'stop' 'down']
```

```python
[5] train_ds, val_ds = tf.keras.utils.audio_dataset_from_directory(
    directory=data_dir,
    batch_size=64,
    validation_split=0.2,
    seed=0,
    output_sequence_length=16000,
    subset='both')

label_names = np.array(train_ds.class_names)
print()
print("label names:", label_names)
```

```
Found 8000 files belonging to 8 classes.
Using 6400 files for training.
Using 1600 files for validation.

label names: ['down' 'go' 'left' 'no' 'right' 'stop' 'up' 'yes']
```

```python
train_ds.element_spec
```

```
(TensorSpec(shape=(None, 16000, None), dtype=tf.float32, name=None),
 TensorSpec(shape=(None,), dtype=tf.int32, name=None))
```

```python
[7] def squeeze(audio, labels):
  audio = tf.squeeze(audio, axis=-1)
  return audio, labels

train_ds = train_ds.map(squeeze, tf.data.AUTOTUNE)
val_ds = val_ds.map(squeeze, tf.data.AUTOTUNE)
```

```python
[8] test_ds = val_ds.shard(num_shards=2, index=0)
val_ds = val_ds.shard(num_shards=2, index=1)
```

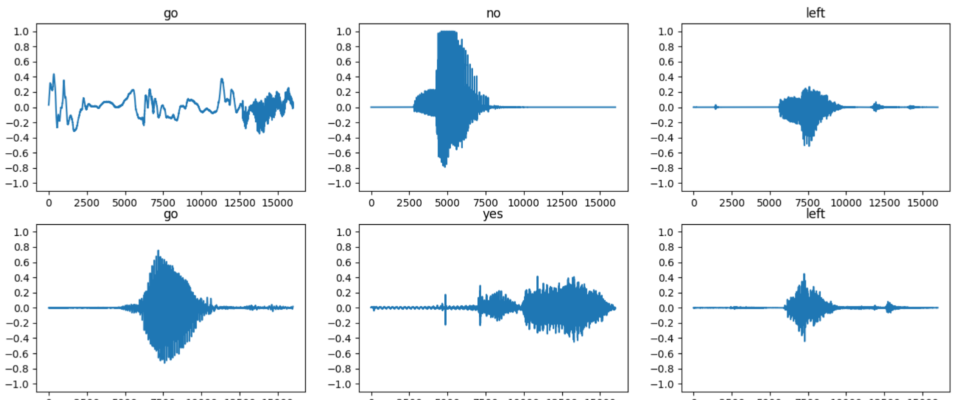
```python
[9] for example_audio, example_labels in train_ds.take(1):
  print(example_audio.shape)
  print(example_labels.shape)
```

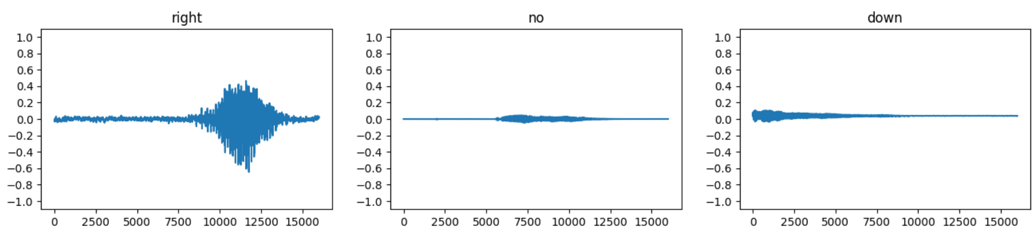
```
(64, 16000)
(64,)
```

```python
[10] label_names[[1,1,3,0]]
```

```
array(['go', 'go', 'no', 'down'], dtype='<U5')
```

```python
plt.figure(figsize=(16, 10))
rows = 3
cols = 3
n = rows * cols
for i in range(n):
  plt.subplot(rows, cols, i+1)
  audio_signal = example_audio[i]
  plt.plot(audio_signal)
  plt.title(label_names[example_labels[i]])
  plt.yticks(np.arange(-1.2, 1.2, 0.2))
  plt.ylim([-1.1, 1.1])
```






```python
[12] def get_spectrogram(waveform):
  # Convert the waveform to a spectrogram via a STFT.
  spectrogram = tf.signal.stft(
      waveform, frame_length=255, frame_step=128)
  # Obtain the magnitude of the STFT.
  spectrogram = tf.abs(spectrogram)
  # Add a `channels` dimension, so that the spectrogram can be used
  # as image-like input data with convolution layers (which expect
  # shape (`batch_size`, `height`, `width`, `channels`).
  spectrogram = spectrogram[..., tf.newaxis]
  return spectrogram
```

```python
for i in range(3):
  label = label_names[example_labels[i]]
  waveform = example_audio[i]
  spectrogram = get_spectrogram(waveform)

  print('Label:', label)
  print('Waveform shape:', waveform.shape)
  print('Spectrogram shape:', spectrogram.shape)
  print('Audio playback')
  display.display(display.Audio(waveform, rate=16000))
```

```
Label: go
Waveform shape: (16000,)
Spectrogram shape: (124, 129, 1)
Audio playback
```

0:00 / 0:01

```
Label: no
Waveform shape: (16000,)
Spectrogram shape: (124, 129, 1)
Audio playback
```

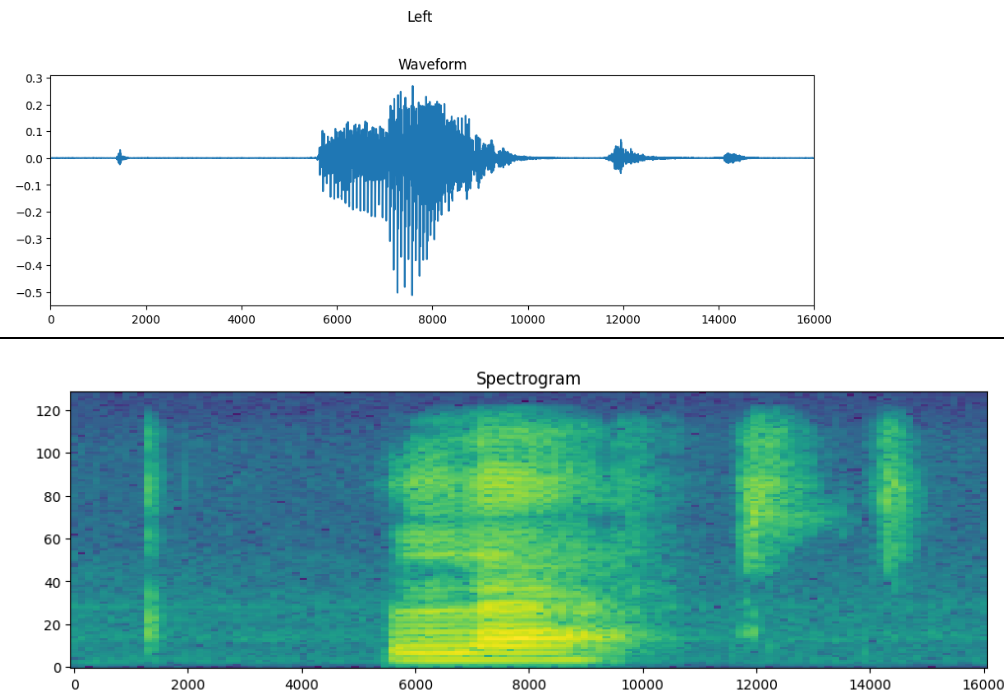
0:00 / 0:01

```
Label: left
Waveform shape: (16000,)
Spectrogram shape: (124, 129, 1)
Audio playback
```

0:00 / 0:01

[14]
```python
def plot_spectrogram(spectrogram, ax):
  if len(spectrogram.shape) > 2:
    assert len(spectrogram.shape) == 3
    spectrogram = np.squeeze(spectrogram, axis=-1)
  # Convert the frequencies to log scale and transpose, so that the time is
  # represented on the x-axis (columns).
  # Add an epsilon to avoid taking a log of zero.
  log_spec = np.log(spectrogram.T + np.finfo(float).eps)
  height = log_spec.shape[0]
  width = log_spec.shape[1]
  X = np.linspace(0, np.size(spectrogram), num=width, dtype=int)
  Y = range(height)
  ax.pcolormesh(X, Y, log_spec)
```

```python
fig, axes = plt.subplots(2, figsize=(12, 8))
timescale = np.arange(waveform.shape[0])
axes[0].plot(timescale, waveform.numpy())
axes[0].set_title('Waveform')
axes[0].set_xlim([0, 16000])

plot_spectrogram(spectrogram.numpy(), axes[1])
axes[1].set_title('Spectrogram')
plt.suptitle(label.title())
plt.show()
```

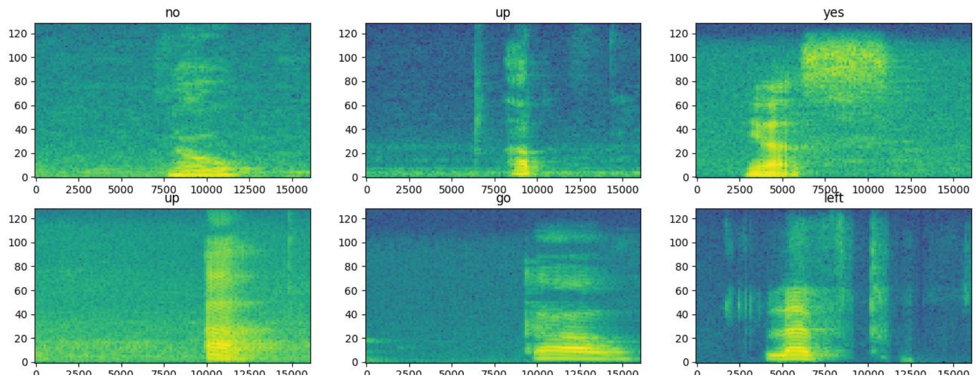


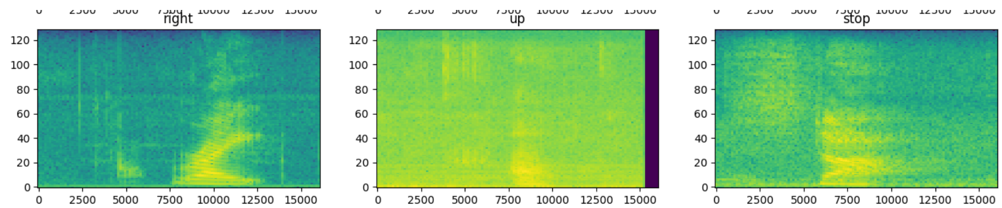
[16]
```python
def make_spec_ds(ds):
  return ds.map(
      map_func=lambda audio,label: (get_spectrogram(audio), label),
      num_parallel_calls=tf.data.AUTOTUNE)
```

[17]
```python
train_spectrogram_ds = make_spec_ds(train_ds)
val_spectrogram_ds = make_spec_ds(val_ds)
test_spectrogram_ds = make_spec_ds(test_ds)
```

[18]
```python
for example_spectrograms, example_spect_labels in train_spectrogram_ds.take(1):
  break
```

```python
rows = 3
cols = 3
n = rows*cols
fig, axes = plt.subplots(rows, cols, figsize=(16, 9))

for i in range(n):
    r = i // cols
    c = i % cols
    ax = axes[r][c]
    plot_spectrogram(example_spectrograms[i].numpy(), ax)
    ax.set_title(label_names[example_spect_labels[i].numpy()])

plt.show()
```






```
[20] train_spectrogram_ds = train_spectrogram_ds.cache().shuffle(10000).prefetch(tf.data.AUTOTUNE)
     val_spectrogram_ds = val_spectrogram_ds.cache().prefetch(tf.data.AUTOTUNE)
     test_spectrogram_ds = test_spectrogram_ds.cache().prefetch(tf.data.AUTOTUNE)
```

```
input_shape = example_spectrograms.shape[1:]
print('Input shape:', input_shape)
num_labels = len(label_names)

# Instantiate the `tf.keras.layers.Normalization` layer.
norm_layer = layers.Normalization()
# Fit the state of the layer to the spectrograms
# with `Normalization.adapt`.
norm_layer.adapt(data=train_spectrogram_ds.map(map_func=lambda spec, label: spec))

model = models.Sequential([
    layers.Input(shape=input_shape),
    # Downsample the input.
    layers.Resizing(32, 32),
    # Normalize.
    norm_layer,
    layers.Conv2D(32, 3, activation='relu'),
    layers.Conv2D(64, 3, activation='relu'),
    layers.MaxPooling2D(),
    layers.Dropout(0.25),
    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dropout(0.5),
    layers.Dense(num_labels),
])

model.summary()
```

```
Input shape: (124, 129, 1)
Model: "sequential"
```

Layer (type)	Output Shape	Param #
resizing (Resizing)	(None, 32, 32, 1)	0
normalization (Normalization)	(None, 32, 32, 1)	3
conv2d (Conv2D)	(None, 30, 30, 32)	320
conv2d_1 (Conv2D)	(None, 28, 28, 64)	18,496
max_pooling2d (MaxPooling2D)	(None, 14, 14, 64)	0
dropout (Dropout)	(None, 14, 14, 64)	0
flatten (Flatten)	(None, 12544)	0
dense (Dense)	(None, 128)	1,605,760
dropout_1 (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 8)	1,032

```
Total params: 1,625,611 (6.20 MB)
Trainable params: 1,625,608 (6.20 MB)
Non-trainable params: 3 (16.00 B)
```

```
[22] model.compile(
         optimizer=tf.keras.optimizers.Adam(),
         loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
         metrics=['accuracy'],
     )
```

```
EPOCHS = 10
history = model.fit(
    train_spectrogram_ds,
    validation_data=val_spectrogram_ds,
    epochs=EPOCHS,
    callbacks=tf.keras.callbacks.EarlyStopping(verbose=1, patience=2),
)
```

```
Epoch 1/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 38s 353ms/step - accuracy: 0.2814 - loss: 1.9261 - val_accuracy: 0.6042 - val_loss: 1.3336
Epoch 2/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 27s 273ms/step - accuracy: 0.5620 - loss: 1.2813 - val_accuracy: 0.7344 - val_loss: 0.9180
Epoch 3/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 39s 259ms/step - accuracy: 0.6797 - loss: 0.9223 - val_accuracy: 0.7721 - val_loss: 0.7437
Epoch 4/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 36s 211ms/step - accuracy: 0.7352 - loss: 0.7276 - val_accuracy: 0.7878 - val_loss: 0.6636
Epoch 5/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 19s 188ms/step - accuracy: 0.7883 - loss: 0.6207 - val_accuracy: 0.8268 - val_loss: 0.5748
Epoch 6/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 22s 200ms/step - accuracy: 0.8133 - loss: 0.5442 - val_accuracy: 0.8346 - val_loss: 0.5365
Epoch 7/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 20s 199ms/step - accuracy: 0.8235 - loss: 0.4907 - val_accuracy: 0.8372 - val_loss: 0.5209
Epoch 8/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 23s 221ms/step - accuracy: 0.8412 - loss: 0.4353 - val_accuracy: 0.8398 - val_loss: 0.4950
Epoch 9/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 20s 201ms/step - accuracy: 0.8807 - loss: 0.3569 - val_accuracy: 0.8451 - val_loss: 0.4714
Epoch 10/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 24s 242ms/step - accuracy: 0.8826 - loss: 0.3339 - val_accuracy: 0.8568 - val_loss: 0.4500
```

```
metrics = history.history
plt.figure(figsize=(16,6))
plt.subplot(1,2,1)
plt.plot(history.epoch, metrics['loss'], metrics['val_loss'])
plt.legend(['loss', 'val_loss'])
plt.ylim([0, max(plt.ylim())])
plt.xlabel('Epoch')
plt.ylabel('Loss [CrossEntropy]')

plt.subplot(1,2,2)
plt.plot(history.epoch, 100*np.array(metrics['accuracy']), 100*np.array(metrics['val_accuracy']))
plt.legend(['accuracy', 'val_accuracy'])
plt.ylim([0, 100])
plt.xlabel('Epoch')
plt.ylabel('Accuracy [%]')
```

```
[25] model.evaluate(test_spectrogram_ds, return_dict=True)

     13/13 ━━━━━━━━━━━━━━━━━━━━ 2s 158ms/step - accuracy: 0.8272 - loss: 0.5049
     {'accuracy': 0.8425480723381042, 'loss': 0.4785589575767517}
```

```
[26] y_pred = model.predict(test_spectrogram_ds)

     13/13 ━━━━━━━━━━━━━━━━━━━━ 1s 44ms/step
```

```
[27] y_pred = tf.argmax(y_pred, axis=1)
```

```
[28] y_true = tf.concat(list(test_spectrogram_ds.map(lambda s,lab: lab)), axis=0)
```

```
confusion_mtx = tf.math.confusion_matrix(y_true, y_pred)
plt.figure(figsize=(10, 8))
sns.heatmap(confusion_mtx,
            xticklabels=label_names,
            yticklabels=label_names,
            annot=True, fmt='g')
plt.xlabel('Prediction')
plt.ylabel('Label')
plt.show()
```

```
x = data_dir/'no/01bb6a2a_nohash_0.wav'
x = tf.io.read_file(str(x))
x, sample_rate = tf.audio.decode_wav(x, desired_channels=1, desired_samples=16000,)
x = tf.squeeze(x, axis=-1)
waveform = x
x = get_spectrogram(x)
x = x[tf.newaxis,...]

prediction = model(x)
x_labels = ['no', 'yes', 'down', 'go', 'left', 'up', 'right', 'stop']
plt.bar(x_labels, tf.nn.softmax(prediction[0]))
plt.title('No')
plt.show()

display.display(display.Audio(waveform, rate=16000))
```

```
class ExportModel(tf.Module):
  def __init__(self, model):
    self.model = model

    # Accept either a string-filename or a batch of waveforms.
    # YOu could add additional signatures for a single wave, or a ragged-batch.
    self.__call__.get_concrete_function(
        x=tf.TensorSpec(shape=(), dtype=tf.string))
    self.__call__.get_concrete_function(
       x=tf.TensorSpec(shape=[None, 16000], dtype=tf.float32))


  @tf.function
  def __call__(self, x):
    # If they pass a string, load the file and decode it.
    if x.dtype == tf.string:
      x = tf.io.read_file(x)
      x, _ = tf.audio.decode_wav(x, desired_channels=1, desired_samples=16000,)
      x = tf.squeeze(x, axis=-1)
      x = x[tf.newaxis, :]

    x = get_spectrogram(x)
    result = self.model(x, training=False)

    class_ids = tf.argmax(result, axis=-1)
    class_names = tf.gather(label_names, class_ids)
    return {'predictions':result,
            'class_ids': class_ids,
            'class_names': class_names}
```

```
[32] export = ExportModel(model)
     export(tf.constant(str(data_dir/'no/01bb6a2a_nohash_0.wav')))
```

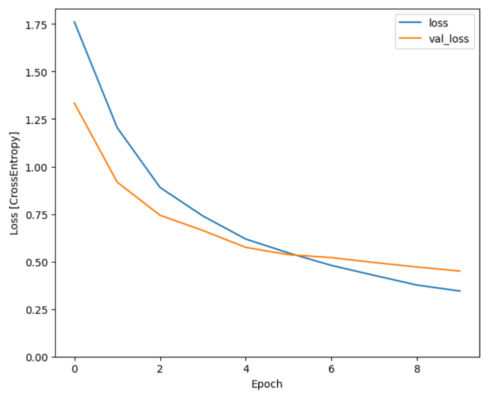
```
{'predictions': <tf.Tensor: shape=(1, 8), dtype=float32, numpy=
 array([[ 2.4269834,  3.6909025, -2.406927 ,  4.960789 , -2.9688535,
         -2.1688123, -2.7355509, -1.0081533]], dtype=float32)>,
 'class_ids': <tf.Tensor: shape=(1,), dtype=int64, numpy=array([3])>,
 'class_names': <tf.Tensor: shape=(1,), dtype=string, numpy=array([b'no'], dtype=object)>}
```

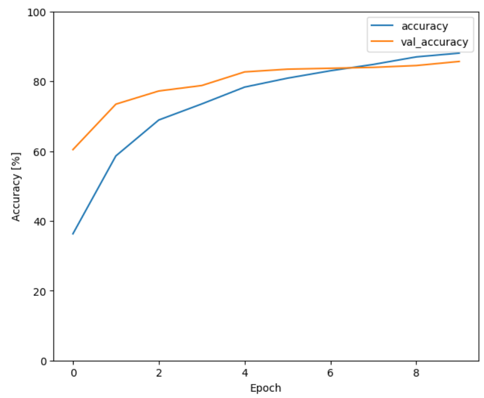
```
tf.saved_model.save(export, "saved")
imported = tf.saved_model.load("saved")
imported(waveform[tf.newaxis, :])
```

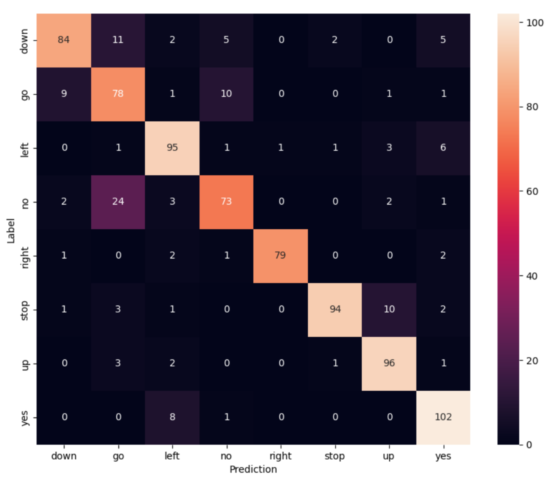
```
{'class_ids': <tf.Tensor: shape=(1,), dtype=int64, numpy=array([3])>,
 'predictions': <tf.Tensor: shape=(1, 8), dtype=float32, numpy=
 array([[ 2.4269834,  3.6909025, -2.406927 ,  4.960789 , -2.9688535,
         -2.1688123, -2.7355509, -1.0081533]], dtype=float32)>,
 'class_names': <tf.Tensor: shape=(1,), dtype=string, numpy=array([b'no'], dtype=object)>}
```

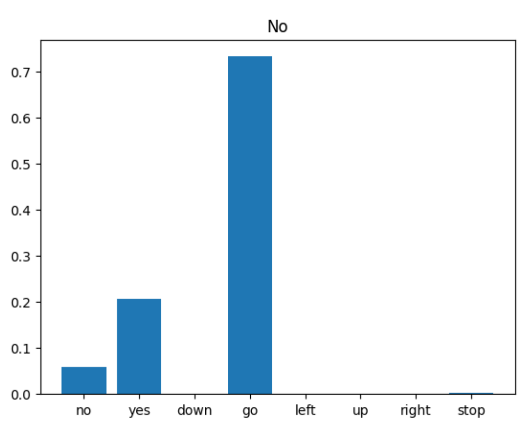
Output:

```
Text(0, 0.5, 'Accuracy [%]')
```









0:00 / 0:01

Conclusion: Hence, the experiment is conducted successfully and the results have been verified.

EXPERIMENT - 11

Aim:

Text generation using RNN.

Experiment Conducted:

```
[ ] import tensorflow as tf

    import numpy as np
    import os
    import time
```

```
[ ] path_to_file = tf.keras.utils.get_file('shakespeare.txt', 'https://storage.googleapis.com/download.tensorflow.org/data/shakespeare.txt')

    Downloading data from https://storage.googleapis.com/download.tensorflow.org/data/shakespeare.txt
    1115394/1115394 [==============================] - 0s 0us/step
```

```
[ ] # Read, then decode for py2 compat.
    text = open(path_to_file, 'rb').read().decode(encoding='utf-8')
    # length of text is the number of characters in it
    print(f'Length of text: {len(text)} characters')

    Length of text: 1115394 characters
```

```
    # Take a look at the first 250 characters in text
    print(text[:250])

    First Citizen:
    Before we proceed any further, hear me speak.

    All:
    Speak, speak.

    First Citizen:
    You are all resolved rather to die than to famish?

    All:
    Resolved. resolved.

    First Citizen:
    First, you know Caius Marcius is chief enemy to the people.
```

```
[ ] # The unique characters in the file
    vocab = sorted(set(text))
    print(f'{len(vocab)} unique characters')

    65 unique characters
```

```
[ ] example_texts = ['abcdefg', 'xyz']

    chars = tf.strings.unicode_split(example_texts, input_encoding='UTF-8')
    chars

    <tf.RaggedTensor [[b'a', b'b', b'c', b'd', b'e', b'f', b'g'], [b'x', b'y', b'z']]>
```

```
[ ] ids_from_chars = tf.keras.layers.StringLookup(
        vocabulary=list(vocab), mask_token=None)
```

```
[ ] ids = ids_from_chars(chars)
    ids

    <tf.RaggedTensor [[40, 41, 42, 43, 44, 45, 46], [63, 64, 65]]>
```

```
[ ] chars_from_ids = tf.keras.layers.StringLookup(
        vocabulary=ids_from_chars.get_vocabulary(), invert=True, mask_token=None)
```

```
[ ] chars = chars_from_ids(ids)
    chars

    <tf.RaggedTensor [[b'a', b'b', b'c', b'd', b'e', b'f', b'g'], [b'x', b'y', b'z']]>
```

```
[ ] tf.strings.reduce_join(chars, axis=-1).numpy()

    array([b'abcdefg', b'xyz'], dtype=object)
```

```
[ ] def text_from_ids(ids):
      return tf.strings.reduce_join(chars_from_ids(ids), axis=-1)
```

```
[ ] all_ids = ids_from_chars(tf.strings.unicode_split(text, 'UTF-8'))
    all_ids

    <tf.Tensor: shape=(1115394,), dtype=int64, numpy=array([19, 48, 57, ..., 46,  9,  1])>
```

```
[ ] ids_dataset = tf.data.Dataset.from_tensor_slices(all_ids)
```

```
[ ] for ids in ids_dataset.take(10):
        print(chars_from_ids(ids).numpy().decode('utf-8'))

    F
    i
    r
    s
    t

    C
    i
    t
    i
```

```
[ ] seq_length = 100
```

```
[ ] sequences = ids_dataset.batch(seq_length+1, drop_remainder=True)

    for seq in sequences.take(1):
      print(chars_from_ids(seq))
```

```
tf.Tensor(
[b'F' b'i' b'r' b's' b't' b' ' b'C' b'i' b't' b'i' b'z' b'e' b'n' b':'
 b'\n' b'B' b'e' b'f' b'o' b'r' b'e' b' ' b'w' b'e' b' ' b'p' b'r' b'o'
 b'c' b'e' b'e' b'd' b' ' b'a' b'n' b'y' b' ' b'f' b'u' b'r' b't' b'h'
 b'e' b'r' b',' b' ' b'h' b'e' b'a' b'r' b' ' b'm' b'e' b' ' b's' b'p'
 b'e' b'a' b'k' b'.' b'\n' b'\n' b'A' b'l' b'l' b':' b'\n' b'S' b'p' b'e'
 b'a' b'k' b',' b' ' b's' b'p' b'e' b'a' b'k' b'.' b'\n' b'\n' b'F' b'i'
 b'r' b's' b't' b' ' b'C' b'i' b't' b'i' b'z' b'e' b'n' b':' b'\n' b'Y'
 b'o' b'u' b' '], shape=(101,), dtype=string)
```

```
[ ] for seq in sequences.take(5):
      print(text_from_ids(seq).numpy())
```

```
b'First Citizen:\nBefore we proceed any further, hear me speak.\n\nAll:\nSpeak, speak.\n\nFirst Citizen:\nYou '
b'are all resolved rather to die than to famish?\n\nAll:\nResolved. resolved.\n\nFirst Citizen:\nFirst, you k'
b"now Caius Marcius is chief enemy to the people.\n\nAll:\nWe know't, we know't.\n\nFirst Citizen:\nLet us ki"
b"ll him, and we'll have corn at our own price.\nIs't a verdict?\n\nAll:\nNo more talking on't; let it be d"
b'one: away, away!\n\nSecond Citizen:\nOne word, good citizens.\n\nFirst Citizen:\nWe are accounted poor citi'
```

```
[ ] def split_input_target(sequence):
        input_text = sequence[:-1]
        target_text = sequence[1:]
        return input_text, target_text
```

```
[ ] split_input_target(list("Tensorflow"))
```

```
(['T', 'e', 'n', 's', 'o', 'r', 'f', 'l', 'o'],
 ['e', 'n', 's', 'o', 'r', 'f', 'l', 'o', 'w'])
```

```
[ ] dataset = sequences.map(split_input_target)
```

```
[ ] for input_example, target_example in dataset.take(1):
        print("Input :", text_from_ids(input_example).numpy())
        print("Target:", text_from_ids(target_example).numpy())
```

```
Input : b'First Citizen:\nBefore we proceed any further, hear me speak.\n\nAll:\nSpeak, speak.\n\nFirst Citizen:\nYou'
Target: b'irst Citizen:\nBefore we proceed any further, hear me speak.\n\nAll:\nSpeak, speak.\n\nFirst Citizen:\nYou '
```

```
[ ] # Batch size
    BATCH_SIZE = 64

    # Buffer size to shuffle the dataset
    # (TF data is designed to work with possibly infinite sequences,
    # so it doesn't attempt to shuffle the entire sequence in memory. Instead,
    # it maintains a buffer in which it shuffles elements).
    BUFFER_SIZE = 10000

    dataset = (
        dataset
        .shuffle(BUFFER_SIZE)
        .batch(BATCH_SIZE, drop_remainder=True)
        .prefetch(tf.data.experimental.AUTOTUNE))

    dataset
```

```
<_PrefetchDataset element_spec=(TensorSpec(shape=(64, 100), dtype=tf.int64, name=None), TensorSpec(shape=(64, 100), dtype=tf.int64, name=None))>
```

```
[ ] # Length of the vocabulary in StringLookup Layer
    vocab_size = len(ids_from_chars.get_vocabulary())

    # The embedding dimension
    embedding_dim = 256

    # Number of RNN units
    rnn_units = 1024
```

```
[ ] class MyModel(tf.keras.Model):
      def __init__(self, vocab_size, embedding_dim, rnn_units):
        super().__init__(self)
        self.embedding = tf.keras.layers.Embedding(vocab_size, embedding_dim)
        self.gru = tf.keras.layers.GRU(rnn_units,
                                       return_sequences=True,
                                       return_state=True)
        self.dense = tf.keras.layers.Dense(vocab_size)

      def call(self, inputs, states=None, return_state=False, training=False):
        x = inputs
        x = self.embedding(x, training=training)
        if states is None:
          states = self.gru.get_initial_state(x)
        x, states = self.gru(x, initial_state=states, training=training)
        x = self.dense(x, training=training)

        if return_state:
          return x, states
        else:
          return x
```

```
[ ] model = MyModel(
        vocab_size=vocab_size,
        embedding_dim=embedding_dim,
        rnn_units=rnn_units)
```

```
[ ] for input_example_batch, target_example_batch in dataset.take(1):
        example_batch_predictions = model(input_example_batch)
        print(example_batch_predictions.shape, "# (batch_size, sequence_length, vocab_size)")
```

```
(64, 100, 66) # (batch_size, sequence_length, vocab_size)
```

```
[ ] model.summary()
```

```
Model: "my_model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       multiple                  16896

 gru (GRU)                   multiple                  3938304

 dense (Dense)               multiple                  67650

=================================================================
Total params: 4022850 (15.35 MB)
Trainable params: 4022850 (15.35 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
[ ] sampled_indices = tf.random.categorical(example_batch_predictions[0], num_samples=1)
    sampled_indices = tf.squeeze(sampled_indices, axis=-1).numpy()
```

```
[ ] sampled_indices
```

```
array([31,  8, 36, 39, 54, 15, 57, 64, 21, 18, 53, 44, 26,  7, 24, 64, 35,
       14, 56, 25, 10, 25, 41, 10, 27, 46, 49, 40, 32, 10,  8, 19, 22, 13,
       28,  5, 49, 62, 24, 13, 46,  7,  7, 35,  5, 20, 24, 62, 65, 25, 19,
       65, 35, 34, 62, 25, 23, 30, 41, 21, 16, 65, 33, 54, 57, 37, 41, 58,
       47, 21, 54, 16,  9, 61, 14,  0, 43, 58, 51, 33,  2, 44, 42, 26,  2,
       12, 10, 28,  8, 10, 26, 54,  5, 51, 52,  8, 34, 10, 55, 30])
```

```
[ ] print("Input:\n", text_from_ids(input_example_batch[0]).numpy())
    print()
    print("Next Char Predictions:\n", text_from_ids(sampled_indices).numpy())
```

```
Input:
 b's follows, if you will not change your purpose\nBut undergo this flight, make for Sicilia,\nAnd there '

Next Char Predictions:
 b'R-WZoBryHEneM,KyVAqL3Lb3NgjaS3-FI?O&jwK?g,,V&GKwzLFzVUwLJQbHCzTorXbshHoC.vA[UNK]dslT ecM ;3O-3Mo&lm-U3pQ'
```

```
[ ] loss = tf.losses.SparseCategoricalCrossentropy(from_logits=True)
```

```
[ ] example_batch_mean_loss = loss(target_example_batch, example_batch_predictions)
    print("Prediction shape: ", example_batch_predictions.shape, " # (batch_size, sequence_length, vocab_size)")
    print("Mean loss:        ", example_batch_mean_loss)
```

```
Prediction shape:  (64, 100, 66)  # (batch_size, sequence_length, vocab_size)
Mean loss:         tf.Tensor(4.189323, shape=(), dtype=float32)
```

```
[ ] tf.exp(example_batch_mean_loss).numpy()
```

```
65.9781
```

```
[ ] model.compile(optimizer='adam', loss=loss)
```

```
[ ] # Directory where the checkpoints will be saved
    checkpoint_dir = './training_checkpoints'
    # Name of the checkpoint files
    checkpoint_prefix = os.path.join(checkpoint_dir, "ckpt_{epoch}")

    checkpoint_callback = tf.keras.callbacks.ModelCheckpoint(
        filepath=checkpoint_prefix,
        save_weights_only=True)
```

```
EPOCHS = 20
```

```
history = model.fit(dataset, epochs=EPOCHS, callbacks=[checkpoint_callback])
```

```
Epoch 1/20
172/172 [==============================] - 13s 51ms/step - loss: 2.7380
```

```
class OneStep(tf.keras.Model):
  def __init__(self, model, chars_from_ids, ids_from_chars, temperature=1.0):
    super().__init__()
    self.temperature = temperature
    self.model = model
    self.chars_from_ids = chars_from_ids
    self.ids_from_chars = ids_from_chars

    # Create a mask to prevent "[UNK]" from being generated.
    skip_ids = self.ids_from_chars(['[UNK]'])[:, None]
    sparse_mask = tf.SparseTensor(
        # Put a -inf at each bad index.
        values=[-float('inf')]*len(skip_ids),
        indices=skip_ids,
        # Match the shape to the vocabulary
        dense_shape=[len(ids_from_chars.get_vocabulary())])
    self.prediction_mask = tf.sparse.to_dense(sparse_mask)

  @tf.function
  def generate_one_step(self, inputs, states=None):
    # Convert strings to token IDs.
    input_chars = tf.strings.unicode_split(inputs, 'UTF-8')
    input_ids = self.ids_from_chars(input_chars).to_tensor()

    # Run the model.
    # predicted_logits.shape is [batch, char, next_char_logits]
    predicted_logits, states = self.model(inputs=input_ids, states=states,
                                          return_state=True)
    # Only use the last prediction.
    predicted_logits = predicted_logits[:, -1, :]
    predicted_logits = predicted_logits/self.temperature
    # Apply the prediction mask: prevent "[UNK]" from being generated.
    predicted_logits = predicted_logits + self.prediction_mask

    # Sample the output logits to generate token IDs.
    predicted_ids = tf.random.categorical(predicted_logits, num_samples=1)
    predicted_ids = tf.squeeze(predicted_ids, axis=-1)

    # Convert from token ids to characters
    predicted_chars = self.chars_from_ids(predicted_ids)

    # Return the characters and model state.
    return predicted_chars, states
```

```
[ ] one_step_model = OneStep(model, chars_from_ids, ids_from_chars)
```

```
start = time.time()
states = None
next_char = tf.constant(['ROMEO:'])
result = [next_char]

for n in range(1000):
  next_char, states = one_step_model.generate_one_step(next_char, states=states)
  result.append(next_char)

result = tf.strings.join(result)
end = time.time()
print(result[0].numpy().decode('utf-8'), '\n\n' + '_'*80)
print('\nRun time:', end - start)
```

```
[ ] start = time.time()
    states = None
    next_char = tf.constant(['ROMEO:', 'ROMEO:', 'ROMEO:', 'ROMEO:', 'ROMEO:'])
    result = [next_char]

    for n in range(1000):
      next_char, states = one_step_model.generate_one_step(next_char, states=states)
      result.append(next_char)

    result = tf.strings.join(result)
    end = time.time()
    print(result, '\n\n' + '_'*80)
    print('\nRun time:', end - start)
```

```
[ ] tf.saved_model.save(one_step_model, 'one_step')
    one_step_reloaded = tf.saved_model.load('one_step')
```

```
WARNING:tensorflow:Skipping full serialization of Keras layer <__main__.OneStep object at 0x7c263b9a8580>, because it is not built.
WARNING:tensorflow:Model's `__init__()` arguments contain non-serializable objects. Please implement a `get_config()` method in the subclassed Model for proper saving and loading. Defaulting to empty config.
WARNING:tensorflow:Model's `__init__()` arguments contain non-serializable objects. Please implement a `get_config()` method in the subclassed Model for proper saving and loading. Defaulting to empty config.
```

```
[ ] states = None
    next_char = tf.constant(['ROMEO:'])
    result = [next_char]

    for n in range(100):
      next_char, states = one_step_reloaded.generate_one_step(next_char, states=states)
      result.append(next_char)

    print(tf.strings.join(result)[0].numpy().decode("utf-8"))
```

```
[ ] class CustomTraining(MyModel):
      @tf.function
      def train_step(self, inputs):
          inputs, labels = inputs
          with tf.GradientTape() as tape:
              predictions = self(inputs, training=True)
              loss = self.loss(labels, predictions)
          grads = tape.gradient(loss, model.trainable_variables)
          self.optimizer.apply_gradients(zip(grads, model.trainable_variables))

          return {'loss': loss}
```

```
[ ] model = CustomTraining(
        vocab_size=len(ids_from_chars.get_vocabulary()),
        embedding_dim=embedding_dim,
        rnn_units=rnn_units)
```

```
[ ] model.compile(optimizer = tf.keras.optimizers.Adam(),
                  loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True))
```

```
[ ] model.fit(dataset, epochs=1)
```

```
172/172 [==============================] - 13s 52ms/step - loss: 2.7221
<keras.src.callbacks.History at 0x7c2611b79660>
```

```
EPOCHS = 10

mean = tf.metrics.Mean()

for epoch in range(EPOCHS):
    start = time.time()

    mean.reset_states()
    for (batch_n, (inp, target)) in enumerate(dataset):
        logs = model.train_step([inp, target])
        mean.update_state(logs['loss'])

        if batch_n % 50 == 0:
            template = f"Epoch {epoch+1} Batch {batch_n} Loss {logs['loss']:.4f}"
            print(template)

    # saving (checkpoint) the model every 5 epochs
    if (epoch + 1) % 5 == 0:
        model.save_weights(checkpoint_prefix.format(epoch=epoch))

    print()
    print(f'Epoch {epoch+1} Loss: {mean.result().numpy():.4f}')
    print(f'Time taken for 1 epoch {time.time() - start:.2f} sec')
    print("_"*80)

model.save_weights(checkpoint_prefix.format(epoch=epoch))
```

Output:

```
ROMEO:
Two lords that bears that title substancily
Hath made her. Come, lend me with your respects,
And we are but one indeed, and stuff'd
With poltrows warm.

PAULINA:
I'll pass here;
How many of she, my prophecies, answer,
To bake awaked to some unwillings.

AUFIDIUS:
There's a woman on the earth.

NORTHUMBERLAND:
Save your own bond fetch me her time?

SEBASTIAN:
A daughter of mine he rounds not to be found,
I mean of moubting hatren hast togster.

MERCUTIONA:
Your brother's son shall be to aspire them
To good his hounds and had shrone-thing.

DUKE OF YORK:
Vixe unsweary home and happoser writ you.

DUKE VINCENTIO:
There's please you bring me to the bourney husband
And made the house of Soldiers. Hastings, spent!
As thou mayst tell him Aury, stand by ambments.

ELBOW:
Is the gods be the bloody king!

KATHARINA:
Their mother? why, how now! what's the matter, sir;
But for this night, which going
A beauty of a bleeding smade.
Go with my heart. My very wisdom to
Then break the flinty times to

________________________________________________________________________________

Run time: 3.103527784347534
```

```
tf.Tensor(
[b"ROMEO:\nThe lords of Rome; come. Gault cannot abroac; not hath too man\nMy spirits ruined as those thronging work,\nYou, the extremest her words:\n'Tis thought upon: 'Heart's ease, and that\nThen I am all dit dinner.\n\nYORK:\nO, what of her mad?\n\nDUKE OF YORK:\nThen, either, master; I must confess,
 b"ROMEO:\nThe pot of mine own breath to save his,\nShe drep proved by for. They treek,\nLaws lightning before I secont her air,\nAnd meet him from such friendship deposing\nFor visitors.\nIs the cunning this amazent,\nEven to their grave. Then for my master's! I' thee\nhad better to the general of thy t
 b"ROMEO:\nWhet crook less man that makes her match's frost;\nAnd set a quectly grief makes him on him.\n\nBUCKINGHAM:\n\nKING RICHARD II:\nWe did obder and consume the glory knock.\n\nKATHARINA:\nGo, beat them good, I'll tell you on\nThe day world in what plant thou thine eefience\nTaken order and o'er
 b"ROMEO:\nThe most rewellings send him from our discontent.\n\nYORK:\nO prince can, father, great king myself.\n\nLADY ANNE:\nSome frother, for it were pitied, sir; and speak fair love.\nMadam, be still we being want, our wit throng.\nWhy, they have pardons for, sir; cast\nIs for my body's a very pin'd.
 b"ROMEO:\nSpeak, speak not a secord distress'd.\n\nPost:\n\nAUFIDIUS:\nMy conscience is too harlour.\nWhy, there is my mother; why, they speak?\n\nNORFOLK:\nIs this the age of York is there, at she,\nBy you denied tockunding agry, and was\nOur practise in the bonest. Romeo! ha! let us as aboard;\nMy exi

________________________________________________________________________________

Run time: 3.146984338760376
```

```
ROMEO:
Let it not; but ere they come?
While pity to you both pardon from my name?
Is contrarist it! I am,
```

```
________________________________________________________________________________
Epoch 7 Batch 0 Loss 1.2602
Epoch 7 Batch 50 Loss 1.2625
Epoch 7 Batch 100 Loss 1.2575
Epoch 7 Batch 150 Loss 1.2652

Epoch 7 Loss: 1.2877
Time taken for 1 epoch 10.32 sec
________________________________________________________________________________
Epoch 8 Batch 0 Loss 1.2132
Epoch 8 Batch 50 Loss 1.2511
Epoch 8 Batch 100 Loss 1.2404
Epoch 8 Batch 150 Loss 1.2811

Epoch 8 Loss: 1.2460
Time taken for 1 epoch 10.28 sec
________________________________________________________________________________
Epoch 9 Batch 0 Loss 1.1680
Epoch 9 Batch 50 Loss 1.2267
Epoch 9 Batch 100 Loss 1.2225
Epoch 9 Batch 150 Loss 1.2380

Epoch 9 Loss: 1.2067
Time taken for 1 epoch 10.30 sec
________________________________________________________________________________
Epoch 10 Batch 0 Loss 1.1215
Epoch 10 Batch 50 Loss 1.1367
Epoch 10 Batch 100 Loss 1.1851
Epoch 10 Batch 150 Loss 1.2080

Epoch 10 Loss: 1.1672
Time taken for 1 epoch 10.51 sec
```

Conclusion: Hence, the experiment is conducted successfully and the results have been verified.