CSC660 DEEP LEARNING

CONTINUOUS ASSESSMENT 1

NAME: GAYATHRI C

UNIQUE ID: E7122002

M.Sc. DEGREE(AI)-TERM 4

The rain in Australia dataset is attached this question paper as weatherAUS.csv file. This dataset contains daily weather observations from numerous Australian weather stations. The target variable RainTomorrow.

1. **Perform the necessary pre-processing. [2]**

SOURCE CODE:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
import pandas as pd
# Read the CSV file into a DataFrame
data = pd.read_csv('/content/drive/MyDrive/DeepLearning_Dataset/weatherAUS.csv')
data.info()
data.head()
```

```
data.head()
```

	Date	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir	WindGustSpeed	WindDir9am	...	Humidity9am	Humidity
0	2008-12-01	Albury	13.4	22.9	0.6	NaN	NaN	W	44.0	W	...	71.0	[illegible]
1	2008-12-02	Albury	7.4	25.1	0.0	NaN	NaN	WNW	44.0	NNW	...	44.0	[illegible]
2	2008-12-03	Albury	12.9	25.7	0.0	NaN	NaN	WSW	46.0	W	...	38.0	[illegible]
3	2008-12-04	Albury	9.2	28.0	0.0	NaN	NaN	NE	24.0	SE	...	45.0	[illegible]
4	2008-12-05	Albury	17.5	32.3	1.0	NaN	NaN	W	41.0	ENE	...	82.0	[illegible]

5 rows × 23 columns

```
# Handle missing values
data = data.dropna()
# Convert categorical variables into numerical format
le = LabelEncoder()
data['RainToday'] = le.fit_transform(data['RainToday'])
data['RainTomorrow'] = le.fit_transform(data['RainTomorrow'])
```

```
data = pd.get_dummies(data, drop_first=True)
# Normalize or standardize numerical features
scaler = StandardScaler()
numerical_cols = ['MinTemp', 'MaxTemp', 'Rainfall', 'Evaporation', 'Sunshine',
'WindGustSpeed', 'WindSpeed9am', 'WindSpeed3pm', 'Humidity9am', 'Humidity3pm',
'Pressure9am', 'Pressure3pm', 'Cloud9am', 'Cloud3pm', 'Temp9am', 'Temp3pm']
data[numerical_cols] = scaler.fit_transform(data[numerical_cols])
# Split the dataset into training and test sets
X = data.drop('RainTomorrow', axis=1)
y = data['RainTomorrow']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

2. **Build an ANN model. Plot accuracy and loss for training and validation dataset. [5]**

SOURCE CODE:

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
# Build the ANN model
model = Sequential()
model.add(Dense(64, input_dim=X_train.shape[1], activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(32, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
# Compile the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
# Train the model
history = model.fit(X_train, y_train, epochs=50, batch_size=32, validation_data=(X_test,
y_test))
```

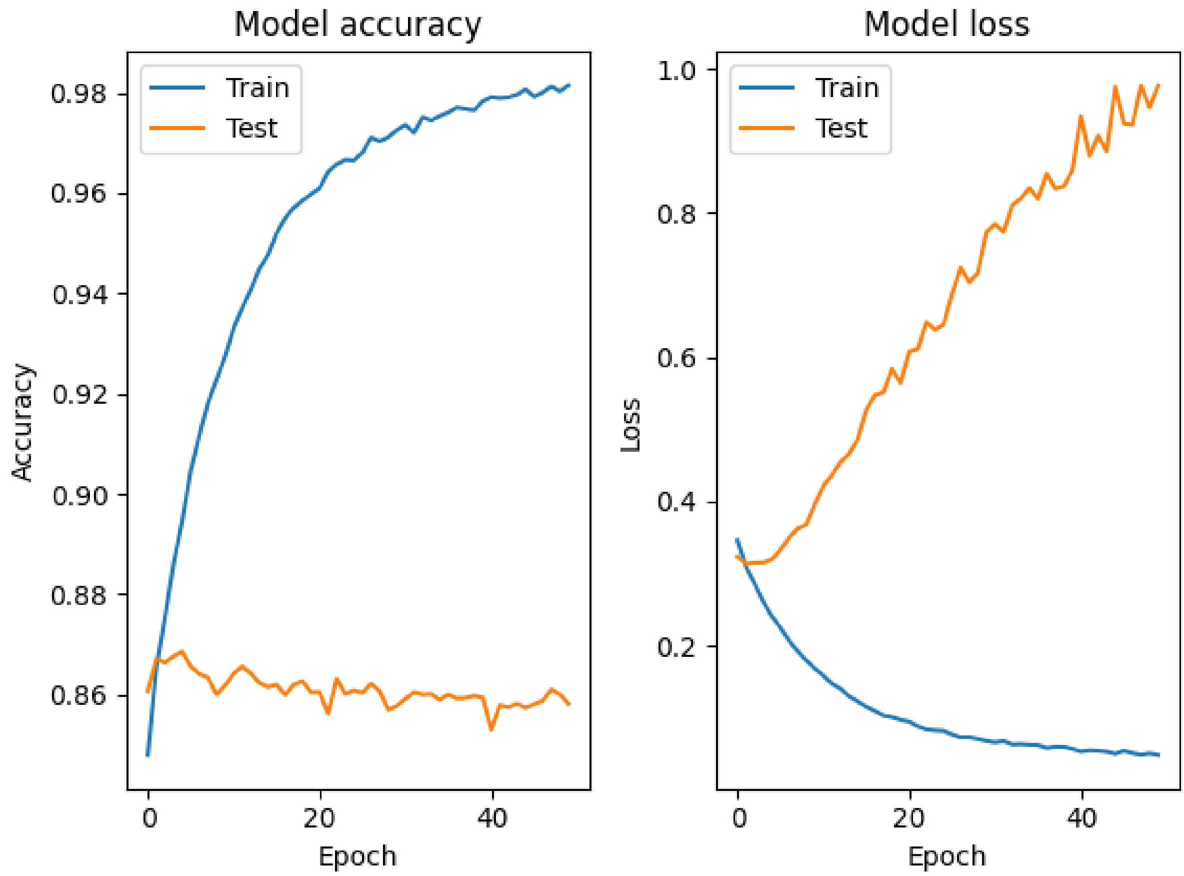
```
# Train the model
history = model.fit(X_train, y_train, epochs=50, batch_size=32, validation_data=(X_test, y_test))
```

```
Epoch 1/50
1411/1411 [==============================] - 12s 8ms/step - loss: 0.3460 - accuracy: 0.8480 - val_loss: 0.3235 - val_accuracy: 0.8608
Epoch 2/50
1411/1411 [==============================] - 9s 6ms/step - loss: 0.3091 - accuracy: 0.8654 - val_loss: 0.3138 - val_accuracy: 0.8672
Epoch 3/50
1411/1411 [==============================] - 11s 8ms/step - loss: 0.2862 - accuracy: 0.8754 - val_loss: 0.3151 - val_accuracy: 0.8664
Epoch 4/50
1411/1411 [==============================] - 11s 8ms/step - loss: 0.2612 - accuracy: 0.8861 - val_loss: 0.3153 - val_accuracy: 0.8677
Epoch 5/50
1411/1411 [==============================] - 8s 6ms/step - loss: 0.2407 - accuracy: 0.8945 - val_loss: 0.3196 - val_accuracy: 0.8686
Epoch 6/50
1411/1411 [==============================] - 11s 8ms/step - loss: 0.2252 - accuracy: 0.9046 - val_loss: 0.3329 - val_accuracy: 0.8657
Epoch 7/50
1411/1411 [==============================] - 9s 7ms/step - loss: 0.2072 - accuracy: 0.9117 - val_loss: 0.3494 - val_accuracy: 0.8641
Epoch 8/50
1411/1411 [==============================] - 9s 6ms/step - loss: 0.1925 - accuracy: 0.9181 - val_loss: 0.3624 - val_accuracy: 0.8634
```

```
1411/1411 [==============================] - 9s 7ms/step - loss: 0.0974 - accuracy: 0.9597 - val_loss: 0.5638 - val_accuracy: 0.8605
Epoch 21/50
1411/1411 [==============================] - 9s 6ms/step - loss: 0.0948 - accuracy: 0.9609 - val_loss: 0.6077 - val_accuracy: 0.8605
Epoch 22/50
1411/1411 [==============================] - 9s 6ms/step - loss: 0.0884 - accuracy: 0.9642 - val_loss: 0.6112 - val_accuracy: 0.8563
Epoch 23/50
1411/1411 [==============================] - 10s 7ms/step - loss: 0.0841 - accuracy: 0.9657 - val_loss: 0.6486 - val_accuracy: 0.8632
Epoch 24/50
1411/1411 [==============================] - 8s 6ms/step - loss: 0.0828 - accuracy: 0.9666 - val_loss: 0.6380 - val_accuracy: 0.8602
Epoch 25/50
1411/1411 [==============================] - 11s 7ms/step - loss: 0.0821 - accuracy: 0.9665 - val_loss: 0.6451 - val_accuracy: 0.8609
Epoch 26/50
1411/1411 [==============================] - 8s 6ms/step - loss: 0.0771 - accuracy: 0.9681 - val_loss: 0.6879 - val_accuracy: 0.8604
Epoch 27/50
1411/1411 [==============================] - 10s 7ms/step - loss: 0.0731 - accuracy: 0.9710 - val_loss: 0.7238 - val_accuracy: 0.8622
Epoch 28/50
1411/1411 [==============================] - 9s 6ms/step - loss: 0.0733 - accuracy: 0.9703 - val_loss: 0.7037 - val_accuracy: 0.8608
Epoch 29/50
1411/1411 [==============================] - 9s 6ms/step - loss: 0.0709 - accuracy: 0.9711 - val_loss: 0.7169 - val_accuracy: 0.8570
```

```
# Plot accuracy and loss
import matplotlib.pyplot as plt
plt.figure(figsize=(12, 4))
# Plot training & validation accuracy values
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('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.tight_layout()
plt.show()
```



3. **Implement two regularization techniques and analyze the performance before and after regularization [3]**

SOURCE CODE:

Two common regularization techniques are **Dropout** and **L1/L2 regularization.**

Dropout: This randomly drops a fraction of the input units to 0 at each update during training, which helps prevent overfitting.

L1/L2 regularization: These add a penalty to the loss function for large values of model parameters. L1 regularization leads to sparsity, while L2 regularization leads to smaller weights.

```
# Using Dropout and L2 regularization
model = Sequential()
```

model.add(Dense(64, input_dim=X_train.shape[1], activation='relu', kernel_regularizer=tf.keras.regularizers.l2(0.001)))

model.add(Dropout(0.5))

model.add(Dense(32, activation='relu', kernel_regularizer=tf.keras.regularizers.l2(0.001)))

model.add(Dense(1, activation='sigmoid', kernel_regularizer=tf.keras.regularizers.l2(0.001)))

from tensorflow.keras.optimizers import Adam

Compile the model

model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

Train the model

history = model.fit(X_train, y_train, epochs=50, batch_size=32, validation_data=(X_test, y_test))

```
1411/1411 [==============================] - 14s 10ms/step - loss: 0.3833 - accuracy: 0.8482 - val_loss: 0.3536 - val_acc
Epoch 2/50
1411/1411 [==============================] - 12s 8ms/step - loss: 0.3556 - accuracy: 0.8570 - val_loss: 0.3487 - val_accuracy: 0.8611
Epoch 3/50
1411/1411 [==============================] - 8s 6ms/step - loss: 0.3527 - accuracy: 0.8572 - val_loss: 0.3474 - val_accuracy: 0.8602
Epoch 4/50
1411/1411 [==============================] - 11s 7ms/step - loss: 0.3517 - accuracy: 0.8578 - val_loss: 0.3455 - val_accuracy: 0.8615
Epoch 5/50
1411/1411 [==============================] - 10s 7ms/step - loss: 0.3503 - accuracy: 0.8582 - val_loss: 0.3472 - val_accuracy: 0.8610
Epoch 6/50
1411/1411 [==============================] - 9s 6ms/step - loss: 0.3497 - accuracy: 0.8588 - val_loss: 0.3465 - val_accuracy: 0.8618
Epoch 7/50
1411/1411 [==============================] - 11s 8ms/step - loss: 0.3509 - accuracy: 0.8578 - val_loss: 0.3465 - val_accuracy: 0.8616
Epoch 8/50
1411/1411 [==============================] - 10s 7ms/step - loss: 0.3500 - accuracy: 0.8593 - val_loss: 0.3465 - val_accuracy: 0.8630
Epoch 9/50
1411/1411 [==============================] - 9s 6ms/step - loss: 0.3488 - accuracy: 0.8597 - val_loss: 0.3446 - val_accuracy: 0.8640
Epoch 10/50
1411/1411 [==============================] - 12s 8ms/step - loss: 0.3498 - accuracy: 0.8588 - val_loss: 0.3459 - val_accuracy: 0.8624
Epoch 11/50
1411/1411 [==============================] - 13s 9ms/step - loss: 0.3489 - accuracy: 0.8604 - val_loss: 0.3468 - val_accuracy: 0.8628
Epoch 12/50
1411/1411 [==============================] - 9s 6ms/step - loss: 0.3491 - accuracy: 0.8591 - val_loss: 0.3460 - val_accuracy: 0.8631
Epoch 13/50
1411/1411 [==============================] - 10s 7ms/step - loss: 0.3498 - accuracy: 0.8588 - val_loss: 0.3455 - val_accuracy: 0.8632
```

```
1411/1411 [==============================] - 10s 7ms/step - loss: 0.3485 - accuracy: 0.8600 - val_loss: 0.3456 - val_accur
Epoch 40/50
1411/1411 [==============================] - 9s 6ms/step - loss: 0.3485 - accuracy: 0.8600 - val_loss: 0.3469 - val_accuracy: 0.8625
Epoch 41/50
1411/1411 [==============================] - 11s 8ms/step - loss: 0.3478 - accuracy: 0.8601 - val_loss: 0.3458 - val_accuracy: 0.8627
Epoch 42/50
1411/1411 [==============================] - 10s 7ms/step - loss: 0.3473 - accuracy: 0.8608 - val_loss: 0.3454 - val_accuracy: 0.8640
Epoch 43/50
1411/1411 [==============================] - 9s 7ms/step - loss: 0.3478 - accuracy: 0.8609 - val_loss: 0.3431 - val_accuracy: 0.8625
Epoch 44/50
1411/1411 [==============================] - 10s 7ms/step - loss: 0.3484 - accuracy: 0.8614 - val_loss: 0.3496 - val_accuracy: 0.8617
Epoch 45/50
1411/1411 [==============================] - 11s 8ms/step - loss: 0.3483 - accuracy: 0.8600 - val_loss: 0.3455 - val_accuracy: 0.8633
Epoch 46/50
1411/1411 [==============================] - 10s 7ms/step - loss: 0.3483 - accuracy: 0.8593 - val_loss: 0.3476 - val_accuracy: 0.8620
Epoch 47/50
1411/1411 [==============================] - 12s 8ms/step - loss: 0.3483 - accuracy: 0.8605 - val_loss: 0.3434 - val_accuracy: 0.8627
Epoch 48/50
1411/1411 [==============================] - 12s 9ms/step - loss: 0.3481 - accuracy: 0.8603 - val_loss: 0.3449 - val_accuracy: 0.8633
Epoch 49/50
1411/1411 [==============================] - 10s 7ms/step - loss: 0.3483 - accuracy: 0.8605 - val_loss: 0.3466 - val_accuracy: 0.8637
Epoch 50/50
1411/1411 [==============================] - 11s 8ms/step - loss: 0.3480 - accuracy: 0.8605 - val_loss: 0.3447 - val_accuracy: 0.8633
```

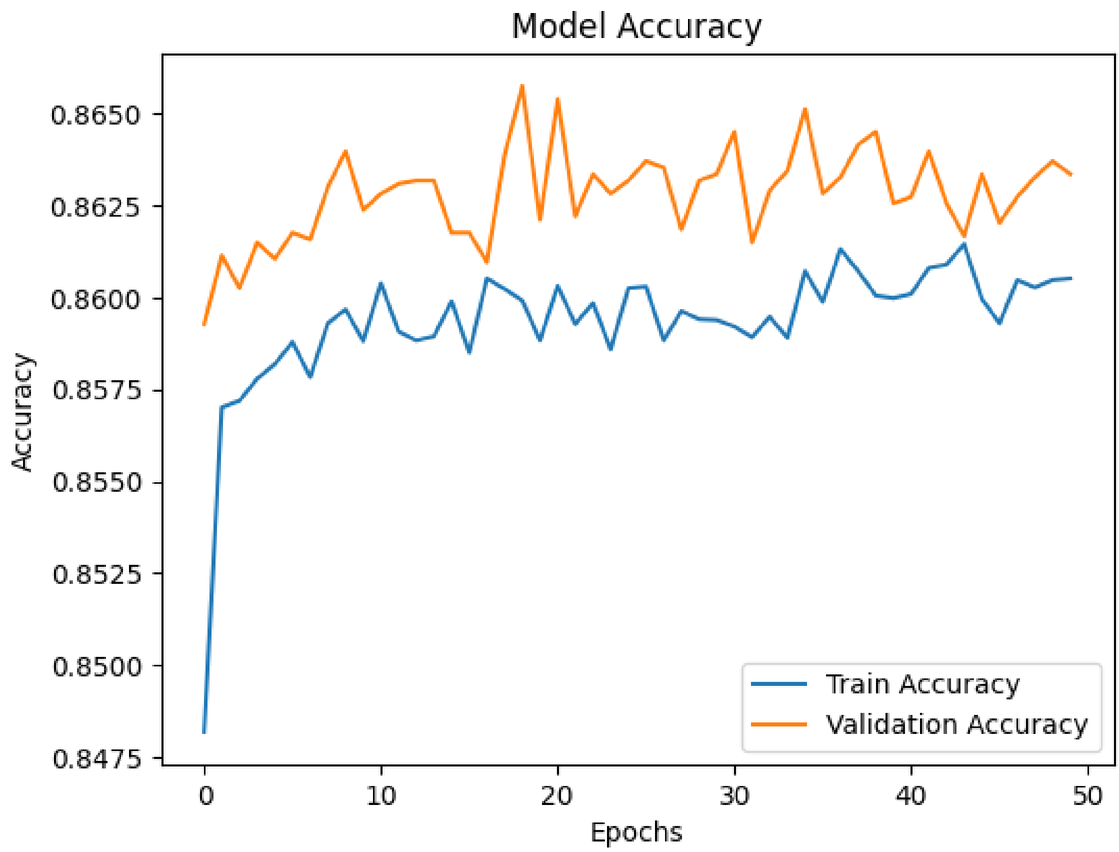
Plot accuracy

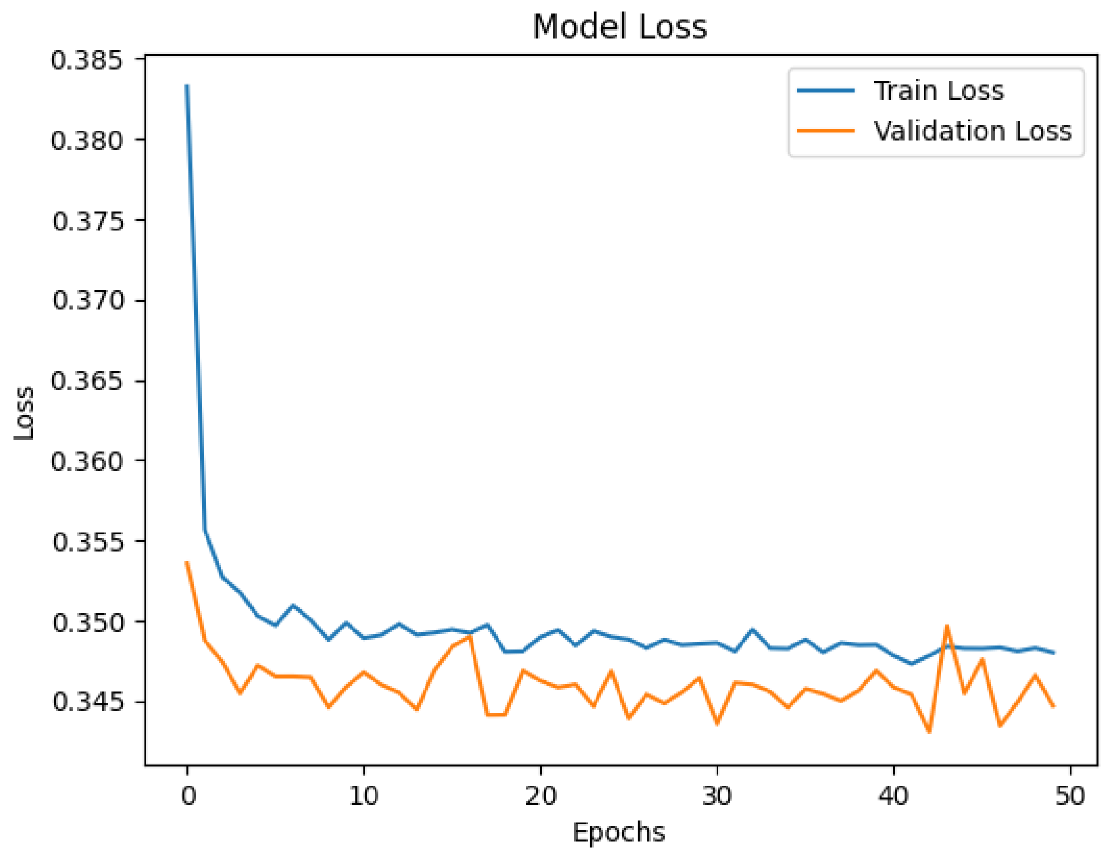
plt.plot(history.history['accuracy'], label='Train Accuracy')

plt.plot(history.history['val_accuracy'], label='Validation Accuracy')

plt.title('Model Accuracy')

```
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()

# Plot loss
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Model Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

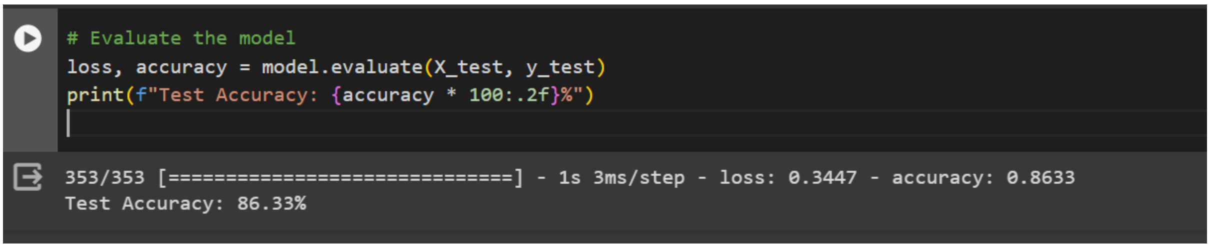




Evaluate the model

loss, accuracy = model.evaluate(X_test, y_test)

print(f"Test Accuracy: {accuracy * 100:.2f}%")



```
# Evaluate the model
loss, accuracy = model.evaluate(X_test, y_test)
print(f"Test Accuracy: {accuracy * 100:.2f}%")
```

```
353/353 [==============================] - 1s 3ms/step - loss: 0.3447 - accuracy: 0.8633
Test Accuracy: 86.33%
```

INTERPRETATION:

* With the introduction of regularization, the training and validation accuracies are closer, which suggests the regularization techniques helped mitigate overfitting.

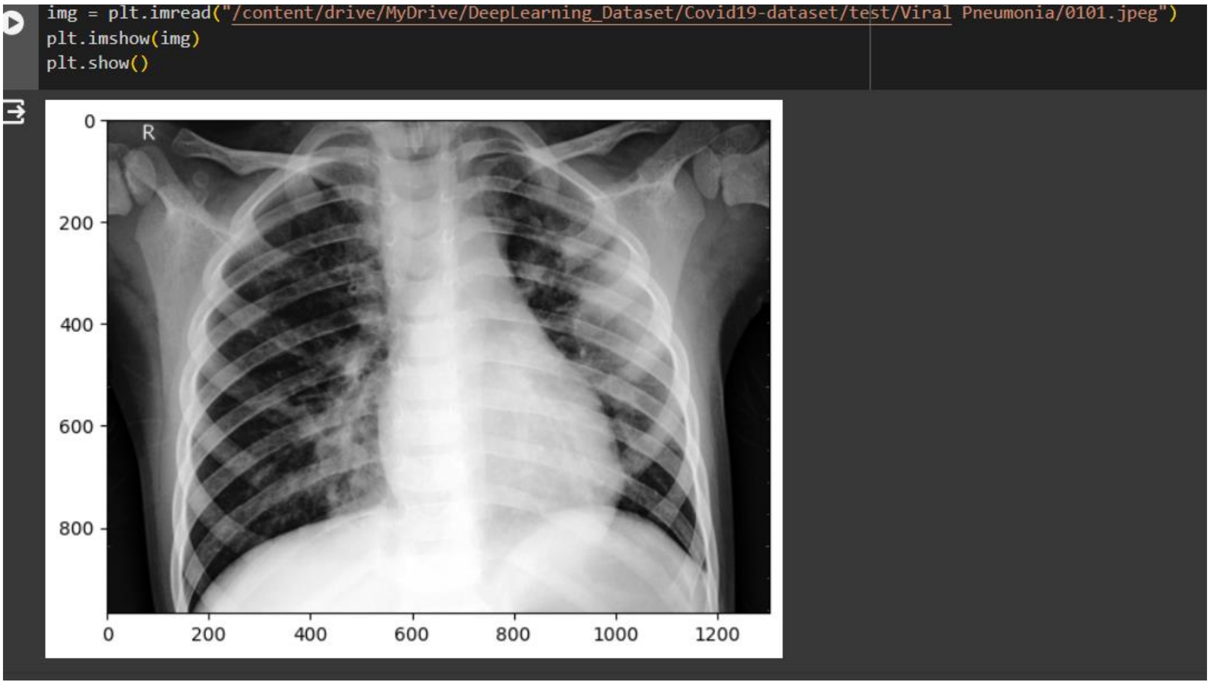
* The test accuracy is 86.33%, which is slightly better than the validation accuracy, indicating that the model generalizes well to unseen data.

* L2 regularization has introduced a penalty on large weights, which has made the model's parameters more conservative, preventing them from fitting too closely to the noise in the training data.

4. For the chosen dataset, build a CNN model with at least 80%accuracy. [4]

SOURCE CODE:

```
import matplotlib.pyplot as plt

img = plt.imread("/content/drive/MyDrive/DeepLearning_Dataset/Covid19-dataset/test/Viral Pneumonia/0101.jpeg")

plt.imshow(img)

plt.show()
```




```
pip install split-folders

import splitfolders

splitfolders.ratio("/content/drive/MyDrive/DeepLearning_Dataset/Covid19-dataset/test",output = "output",seed = 1337,ratio = (.8, .2))

from tensorflow.keras.preprocessing.image import ImageDataGenerator

# all images will be rescaled by 1./255

train_data = ImageDataGenerator(
    rescale = 1./255,)

test_data = ImageDataGenerator(rescale = 1./255)

train_generator =train_data.flow_from_directory(
    "/content/drive/MyDrive/DeepLearning_Dataset/Covid19-dataset/train",
    target_size = (224,224),
    batch_size = 20,
    class_mode = 'categorical')

validation_generator = test_data.flow_from_directory(
    "/content/drive/MyDrive/DeepLearning_Dataset/Covid19-dataset/test",
```

target_size = (224,224),

batch_size = 20,

class_mode = 'categorical')

```
Copying files: 66 files [00:00, 179.56 files/s]
```

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator
# all images will be rescaled by 1./255
train_data = ImageDataGenerator(
    rescale = 1./255,)
test_data = ImageDataGenerator(rescale = 1./255)
train_generator =train_data.flow_from_directory(
    "/content/drive/MyDrive/DeepLearning_Dataset/Covid19-dataset/train",
    target_size = (224,224),
    batch_size = 20,
    class_mode = 'categorical')
validation_generator = test_data.flow_from_directory(
    "/content/drive/MyDrive/DeepLearning_Dataset/Covid19-dataset/test",
    target_size = (224,224),
    batch_size = 20,
    class_mode = 'categorical')
```

```
Found 261 images belonging to 3 classes.
Found 66 images belonging to 3 classes.
```

num_classes = 3

input_shape = (224,224,3)

import tensorflow as tf

from tensorflow import keras

from tensorflow.keras import layers

from tensorflow.keras.models import Sequential

model = Sequential([

layers.Rescaling(1./255, input_shape=(input_shape)), #input layer

layers.Conv2D(16,3,padding = 'same',activation = 'relu'), # 16 is no of filters , filter size is 3*3

layers.MaxPooling2D(),

layers.Conv2D(32,3,padding = 'same',activation = 'relu'),

layers.MaxPooling2D(),

layers.Conv2D(64,3,padding = 'same',activation = 'relu'),

layers.MaxPooling2D(),

layers.Flatten(),

```
    layers.Dense(128,activation = 'relu'), # 1 st hidden layer has 128 neurons
    layers.Dense(256,activation = 'relu'), # 2 nd hidden layer has 256 neurons
    layers.Dense(256,activation = 'relu'), # 3 rd hidden layer has 256 neurons
    layers.Dense(num_classes,activation = 'softmax')
])
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 rescaling_1 (Rescaling)     (None, 224, 224, 3)       0

 conv2d_3 (Conv2D)           (None, 224, 224, 16)      448

 max_pooling2d_3 (MaxPoolin  (None, 112, 112, 16)      0
 g2D)

 conv2d_4 (Conv2D)           (None, 112, 112, 32)      4640

 max_pooling2d_4 (MaxPoolin  (None, 56, 56, 32)        0
 g2D)

 conv2d_5 (Conv2D)           (None, 56, 56, 64)        18496

 max_pooling2d_5 (MaxPoolin  (None, 28, 28, 64)        0
 g2D)

 flatten_1 (Flatten)         (None, 50176)             0

 dense_4 (Dense)             (None, 128)               6422656

 dense_5 (Dense)             (None, 256)               33024

 dense_6 (Dense)             (None, 256)               65792
```

```
model.compile(optimizer ='adam',
            loss = tf.keras.losses.CategoricalCrossentropy(),
            metrics = ['accuracy'])
epochs = 10
history1 = model.fit(
    train_generator ,
    validation_data = validation_generator,
    epochs = epochs
)
```

```
Epoch 1/10
14/14 [==============================] - 23s 2s/step - loss: 1.0955 - accuracy: 0.4559 - val_loss: 1.0947 - val_accuracy: 0.3939
Epoch 2/10
14/14 [==============================] - 21s 1s/step - loss: 1.0853 - accuracy: 0.4559 - val_loss: 1.0913 - val_accuracy: 0.3939
Epoch 3/10
14/14 [==============================] - 21s 2s/step - loss: 1.0754 - accuracy: 0.4559 - val_loss: 1.1059 - val_accuracy: 0.3939
Epoch 4/10
14/14 [==============================] - 21s 1s/step - loss: 1.0738 - accuracy: 0.4559 - val_loss: 1.1151 - val_accuracy: 0.3939
Epoch 5/10
14/14 [==============================] - 22s 1s/step - loss: 1.0718 - accuracy: 0.4559 - val_loss: 1.0909 - val_accuracy: 0.3939
Epoch 6/10
14/14 [==============================] - 28s 2s/step - loss: 1.0724 - accuracy: 0.4559 - val_loss: 1.0935 - val_accuracy: 0.3939
Epoch 7/10
14/14 [==============================] - 20s 1s/step - loss: 1.0695 - accuracy: 0.4559 - val_loss: 1.1045 - val_accuracy: 0.3939
Epoch 8/10
14/14 [==============================] - 21s 1s/step - loss: 1.0682 - accuracy: 0.4559 - val_loss: 1.0967 - val_accuracy: 0.3939
Epoch 9/10
14/14 [==============================] - 22s 1s/step - loss: 1.0680 - accuracy: 0.4559 - val_loss: 1.0917 - val_accuracy: 0.3939
Epoch 10/10
14/14 [==============================] - 22s 1s/step - loss: 1.0707 - accuracy: 0.4559 - val_loss: 1.0911 - val_accuracy: 0.3939
```

5.Now include 5 data augmentation techniques appropriate to your dataset and build CNN on augmented images. [4]

SOURCE CODE:

```
#image generator is to augment the images
from tensorflow.keras.preprocessing.image import ImageDataGenerator
# all images will be rescaled by 1./255
train_data = ImageDataGenerator(
    rescale = 1./255,
    rotation_range = 40,
    width_shift_range = 0.2,
    height_shift_range = 0.2,
    shear_range = 0.2,
    horizontal_flip = True,)
test_data = ImageDataGenerator(rescale = 1./255)
train_generator =train_data.flow_from_directory(
    "/content/drive/MyDrive/DeepLearning_Dataset/Covid19-dataset/train",
    target_size = (224,224),
    batch_size = 20,
    class_mode = 'categorical')
validation_generator = test_data.flow_from_directory(
    "/content/drive/MyDrive/DeepLearning_Dataset/Covid19-dataset/test",
    target_size = (224,224),
```

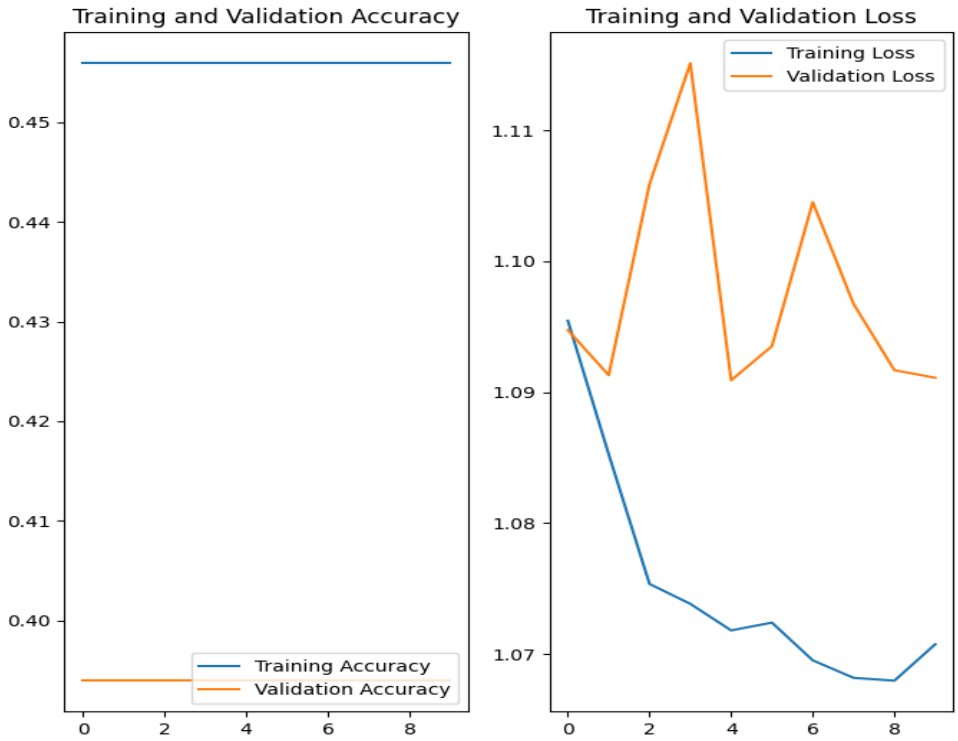
```
    batch_size = 20,
    class_mode = 'categorical')
num_classes = 3
input_shape = (224,224,3)
model = Sequential([
    layers.Rescaling(1./255, input_shape=(input_shape)), #input layer
    layers.Conv2D(16,3,padding = 'same',activation = 'relu'), # 16 is no of filters , filter size is
3*3
    layers.MaxPooling2D(),
    layers.Conv2D(32,3,padding = 'same',activation = 'relu'), # padding = 'same', input size =
output size of an image
    layers.MaxPooling2D(),
    layers.Conv2D(64,3,padding = 'same',activation = 'relu'),
    layers.MaxPooling2D(),
    layers.Flatten(),
    layers.Dense(128,activation = 'relu'), # 1 st hidden layer has 128 neurons
    layers.Dense(256,activation = 'relu'), # 2 nd hidden layer has 256 neurons
    layers.Dense(256,activation = 'relu'), # 3 rd hidden layer has 256 neurons
    layers.Dense(num_classes,activation = 'softmax')
])
model.compile(optimizer ='adam',
            loss = tf.keras.losses.CategoricalCrossentropy(),
            metrics = ['accuracy'])
epochs = 10
history2 = model.fit(
    train_generator ,
    validation_data = validation_generator,
    epochs = epochs
)
```

```
Epoch 1/10
14/14 [==============================] - 32s 2s/step - loss: 1.0945 - accuracy: 0.4023 - val_loss: 1.0914 - val_accuracy: 0.3939
Epoch 2/10
14/14 [==============================] - 31s 2s/step - loss: 1.0735 - accuracy: 0.4559 - val_loss: 1.0934 - val_accuracy: 0.3939
Epoch 3/10
14/14 [==============================] - 36s 3s/step - loss: 1.0798 - accuracy: 0.4559 - val_loss: 1.0919 - val_accuracy: 0.3939
Epoch 4/10
14/14 [==============================] - 31s 2s/step - loss: 1.0801 - accuracy: 0.4559 - val_loss: 1.0907 - val_accuracy: 0.3939
Epoch 5/10
14/14 [==============================] - 25s 2s/step - loss: 1.0747 - accuracy: 0.4559 - val_loss: 1.0951 - val_accuracy: 0.3939
Epoch 6/10
14/14 [==============================] - 26s 2s/step - loss: 1.0687 - accuracy: 0.4559 - val_loss: 1.0976 - val_accuracy: 0.3939
Epoch 7/10
14/14 [==============================] - 28s 2s/step - loss: 1.0696 - accuracy: 0.4559 - val_loss: 1.1277 - val_accuracy: 0.3939
Epoch 8/10
14/14 [==============================] - 24s 2s/step - loss: 1.0704 - accuracy: 0.4559 - val_loss: 1.0939 - val_accuracy: 0.3939
Epoch 9/10
14/14 [==============================] - 28s 2s/step - loss: 1.0710 - accuracy: 0.4559 - val_loss: 1.0913 - val_accuracy: 0.3939
Epoch 10/10
14/14 [==============================] - 25s 2s/step - loss: 1.0719 - accuracy: 0.4559 - val_loss: 1.0936 - val_accuracy: 0.3939
```

6. Compare the performance of above two models [2]

SOURCE CODE:

```
acc = history1.history['accuracy']
val_acc = history1.history['val_accuracy']
loss = history1.history['loss']
val_loss = history1.history['val_loss']
epochs_range = range(epochs)
plt.figure(figsize=(8,8))
plt.subplot(1, 2, 1)
plt.plot(epochs_range, acc, label='Training Accuracy')
plt.plot(epochs_range, val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')
plt.subplot(1, 2, 2)
plt.plot(epochs_range, loss, label='Training Loss')
plt.plot(epochs_range, val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()
```




```
acc_1 = history2.history['accuracy']
val_acc = history2.history['val_accuracy']
loss = history2.history['loss']
val_loss = history2.history['val_loss']
epochs_range = range(epochs)
plt.figure(figsize=(8,8))
plt.subplot(1, 2, 1)
plt.plot(epochs_range, acc, label='Training Accuracy')
plt.plot(epochs_range, val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')
plt.subplot(1, 2, 2)
plt.plot(epochs_range, loss, label='Training Loss')
plt.plot(epochs_range, val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()
```

7. Choose a pre-trained model and implement from scratch on the chosen dataset [5]

SOURCE CODE:

```
from tensorflow.keras.layers import Conv2D, DepthwiseConv2D, ReLU, BatchNormalization,
add,Softmax, AveragePooling2D, Dense, Input, GlobalAveragePooling2D

from tensorflow.keras.models import Model

def expansion_block(x,t,filters,block_id):

    prefix = 'block_{}_'.format(block_id)

    total_filters = t*filters

    x = Conv2D(total_filters,1,padding='same',use_bias=False, name = prefix +'expand')(x)

    x = BatchNormalization(name=prefix +'expand_bn')(x)

    x = ReLU(6,name = prefix +'expand_relu')(x)

    return x


def depthwise_block(x,stride,block_id):

    prefix = 'block_{}_'.format(block_id)

    x = DepthwiseConv2D(3,strides=(stride,stride),padding ='same', use_bias = False, name =
prefix + 'depthwise_conv')(x)
```

```
    x = BatchNormalization(name=prefix +'dw_bn')(x)
    x = ReLU(6,name=prefix +'dw_relu')(x)
    return x

def projection_block(x,out_channels,block_id):
    prefix = 'block_{}_'.format(block_id)
    x = Conv2D(filters = out_channels,kernel_size = 1,padding='same',use_bias=False,name=
prefix + 'compress')(x)
    x = BatchNormalization(name=prefix +'compress_bn')(x)
    return x
def Bottleneck(x,t,filters, out_channels,stride,block_id):
    y = expansion_block(x,t,filters,block_id)
    y = depthwise_block(y,stride,block_id)
    y = projection_block(y, out_channels,block_id)
    if y.shape[-1]==x.shape[-1]:
        y = add([x,y])
    return y
def MobileNetV2(input_image = (224,224,3), n_classes=3):
    input = Input (input_shape)
    x = Conv2D(32,3,strides=(2,2),padding='same', use_bias=False)(input)
    x = BatchNormalization(name='conv1_bn')(x)
    x = ReLU(6, name='conv1_relu')(x)
    # 17 Bottlenecks
    x = depthwise_block(x,stride=1,block_id=1)
    x = projection_block(x, out_channels=16,block_id=1)
    x = Bottleneck(x, t = 6, filters = x.shape[-1], out_channels = 24, stride = 2,block_id = 2)
    x = Bottleneck(x, t = 6, filters = x.shape[-1], out_channels = 24, stride = 1,block_id = 3)
    x = Bottleneck(x, t = 6, filters = x.shape[-1], out_channels = 32, stride = 2,block_id = 4)
    x = Bottleneck(x, t = 6, filters = x.shape[-1], out_channels = 32, stride = 1,block_id = 5)
    x = Bottleneck(x, t = 6, filters = x.shape[-1], out_channels = 32, stride = 1,block_id = 6)
    x = Bottleneck(x, t = 6, filters = x.shape[-1], out_channels = 64, stride = 2,block_id = 7)
    x = Bottleneck(x, t = 6, filters = x.shape[-1], out_channels = 64, stride = 1,block_id = 8)
```

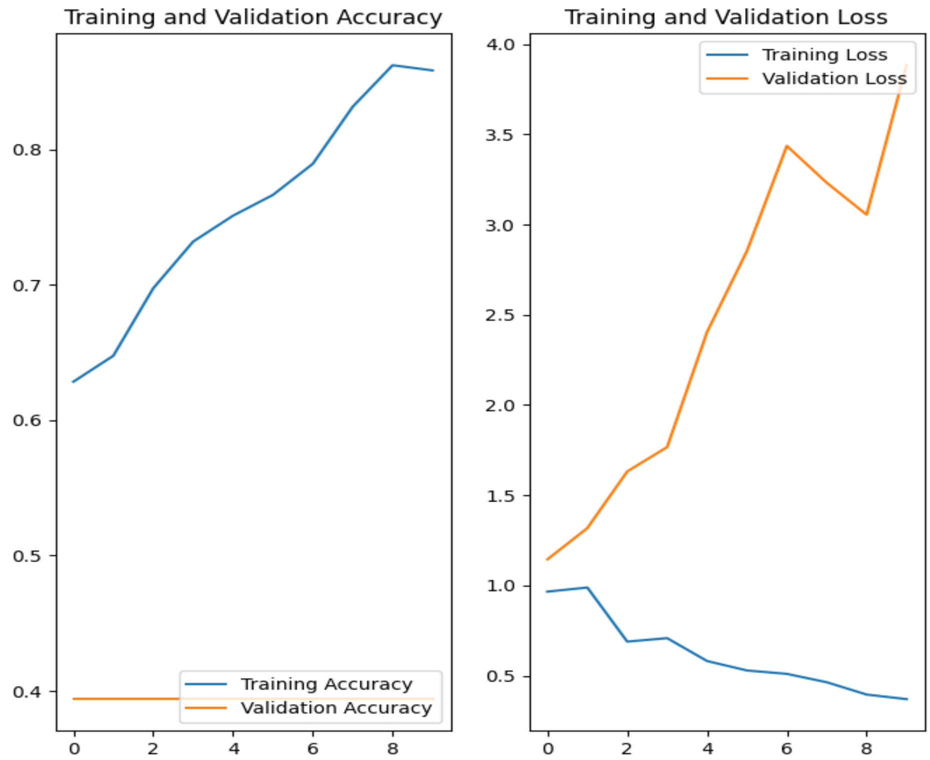
```
    x = Bottleneck(x, t = 6, filters = x.shape[-1], out_channels = 64, stride = 1,block_id = 9)
    x = Bottleneck(x, t = 6, filters = x.shape[-1], out_channels = 64, stride = 1,block_id = 10)
    x = Bottleneck(x, t = 6, filters = x.shape[-1], out_channels = 96, stride = 1,block_id = 11)
    x = Bottleneck(x, t = 6, filters = x.shape[-1], out_channels = 96, stride = 1,block_id = 12)
    x = Bottleneck(x, t = 6, filters = x.shape[-1], out_channels = 96, stride = 1,block_id = 13)
    x = Bottleneck(x, t = 6, filters = x.shape[-1], out_channels = 160, stride = 2,block_id = 14)
    x = Bottleneck(x, t = 6, filters = x.shape[-1], out_channels = 160, stride = 1,block_id = 15)
    x = Bottleneck(x, t = 6, filters = x.shape[-1], out_channels = 160, stride = 1,block_id = 16)
    x = Bottleneck(x, t = 6, filters = x.shape[-1], out_channels = 320, stride = 1,block_id = 17)
    x = Conv2D(filters = 1280,kernel_size = 1,padding='same',use_bias=False, name =
'last_conv')(x)
    x = BatchNormalization(name='last_bn')(x)
    x = ReLU(6,name='last_relu')(x)
    x = GlobalAveragePooling2D()(x)
    output = Dense(3,activation='softmax')(x)
    model = Model(input, output)
    return model
n_classes = 3
input_shape = (224,224,3)
model = MobileNetV2(input_shape,n_classes)
model.summary()
```

```
 hNormalization)

 add_9 (Add)                 (None, 7, 7, 160)            0         ['add_8[0][0]',
                                                                     'block_16_compress_bn[0][0]']

 block_17_expand (Conv2D)    (None, 7, 7, 960)            153600    ['add_9[0][0]']

 block_17_expand_bn (BatchN  (None, 7, 7, 960)            3840      ['block_17_expand[0][0]']
 ormalization)

 block_17_expand_relu (ReLU  (None, 7, 7, 960)            0         ['block_17_expand_bn[0][0]']
 )

 block_17_depthwise_conv (D  (None, 7, 7, 960)            8640      ['block_17_expand_relu[0][0]']
 epthwiseConv2D)

 block_17_dw_bn (BatchNorma  (None, 7, 7, 960)            3840      ['block_17_depthwise_conv[0][0
 lization)                                                          ]']

 block_17_dw_relu (ReLU)     (None, 7, 7, 960)            0         ['block_17_dw_bn[0][0]']

 block_17_compress (Conv2D)  (None, 7, 7, 320)            307200    ['block_17_dw_relu[0][0]']

 block_17_compress_bn (Batc  (None, 7, 7, 320)            1280      ['block_17_compress[0][0]']
 hNormalization)

 last_conv (Conv2D)          (None, 7, 7, 1280)           409600    ['block_17_compress_bn[0][0]']

 last_bn (BatchNormalizatio  (None, 7, 7, 1280)           5120      ['last_conv[0][0]']
 n)

 last_relu (ReLU)            (None, 7, 7, 1280)           0         ['last_bn[0][0]']

 global_average_pooling2d (  (None, 1280)                 0         ['last_relu[0][0]']
 GlobalAveragePooling2D)

 dense_12 (Dense)            (None, 3)                    3843      ['global_average_pooling2d[0][
                                                                    0]']

==================================================================================================
Total params: 2261827 (8.63 MB)
Trainable params: 2227715 (8.50 MB)
Non-trainable params: 34112 (133.25 KB)
__________________________________________________________________________________________________
```

```
import tensorflow as tf
model.compile(optimizer ='adam',
          loss = tf.keras.losses.CategoricalCrossentropy(),
          metrics = ['accuracy'])
epoch = 10
history3 = model.fit(
  train_generator ,
  validation_data = validation_generator,
  epochs = epoch)
```

```
Epoch 1/10
14/14 [==============================] - 76s 4s/step - loss: 0.9655 - accuracy: 0.6284 - val_loss: 1.1445 - val_accuracy: 0.3939
Epoch 2/10
14/14 [==============================] - 57s 4s/step - loss: 0.9883 - accuracy: 0.6475 - val_loss: 1.3178 - val_accuracy: 0.3939
Epoch 3/10
14/14 [==============================] - 60s 4s/step - loss: 0.6884 - accuracy: 0.6973 - val_loss: 1.6317 - val_accuracy: 0.3939
Epoch 4/10
14/14 [==============================] - 55s 4s/step - loss: 0.7077 - accuracy: 0.7318 - val_loss: 1.7668 - val_accuracy: 0.3939
Epoch 5/10
14/14 [==============================] - 57s 4s/step - loss: 0.5809 - accuracy: 0.7510 - val_loss: 2.4045 - val_accuracy: 0.3939
Epoch 6/10
14/14 [==============================] - 54s 4s/step - loss: 0.5285 - accuracy: 0.7663 - val_loss: 2.8537 - val_accuracy: 0.3939
Epoch 7/10
14/14 [==============================] - 62s 4s/step - loss: 0.5097 - accuracy: 0.7893 - val_loss: 3.4363 - val_accuracy: 0.3939
Epoch 8/10
14/14 [==============================] - 57s 4s/step - loss: 0.4633 - accuracy: 0.8314 - val_loss: 3.2319 - val_accuracy: 0.3939
Epoch 9/10
14/14 [==============================] - 61s 4s/step - loss: 0.3949 - accuracy: 0.8621 - val_loss: 3.0549 - val_accuracy: 0.3939
Epoch 10/10
14/14 [==============================] - 60s 4s/step - loss: 0.3697 - accuracy: 0.8582 - val_loss: 3.8832 - val_accuracy: 0.3939
```

```
acc = history3.history['accuracy']
val_acc = history3.history['val_accuracy']
loss = history3.history['loss']
val_loss = history3.history['val_loss']
epochs_range = range(epochs)
plt.figure(figsize=(8,8))
plt.subplot(1, 2, 1)
plt.plot(epochs_range, acc, label='Training Accuracy')
plt.plot(epochs_range, val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')
plt.subplot(1, 2, 2)
plt.plot(epochs_range, loss, label='Training Loss')
plt.plot(epochs_range, val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()
```



8. Choose a pre-trained model and implement as transfer learning on the chosen dataset [5]

SOURCE CODE:

```
from tensorflow.keras import Model

from tensorflow.keras.layers import
Conv2D,Dense,MaxPooling2D,Dropout,Flatten,GlobalAveragePooling2D

from tensorflow.keras.models import Sequential

from tensorflow.keras.applications.mobilenet_v2 import MobileNetV2, preprocess_input

Model_V2 = MobileNetV2(weights='imagenet',include_top = False, input_shape =
(224,224,3))

x = Model_V2.output

x = GlobalAveragePooling2D()(x)

output = Dense(units = 3, activation='softmax')(x)

# The last 15 layers fine tune

for layer in Model_V2.layers[:15]:

    layer.trainable = False

model = Model(inputs=Model_V2.input, outputs=output)

model.compile(optimizer ='adam',

            loss = tf.keras.losses.CategoricalCrossentropy(),
```

```
            metrics = ['accuracy'])
epochs = 10
history4 = model.fit(
   train_generator ,
   validation_data = validation_generator,
   epochs = epochs
)
```

```
Epoch 1/10
14/14 [==============================] - 65s 4s/step - loss: 0.5906 - accuracy: 0.7778 - val_loss: 3.9056 - val_accuracy: 0.6667
Epoch 2/10
14/14 [==============================] - 51s 4s/step - loss: 0.4469 - accuracy: 0.8276 - val_loss: 14.3271 - val_accuracy: 0.3030
Epoch 3/10
14/14 [==============================] - 48s 3s/step - loss: 0.3002 - accuracy: 0.8966 - val_loss: 6.8390 - val_accuracy: 0.3788
Epoch 4/10
14/14 [==============================] - 48s 3s/step - loss: 0.2238 - accuracy: 0.9272 - val_loss: 2.8853 - val_accuracy: 0.5000
Epoch 5/10
14/14 [==============================] - 48s 3s/step - loss: 0.3481 - accuracy: 0.8966 - val_loss: 6.2693 - val_accuracy: 0.3182
Epoch 6/10
14/14 [==============================] - 45s 3s/step - loss: 0.3089 - accuracy: 0.9195 - val_loss: 2.8205 - val_accuracy: 0.4242
Epoch 7/10
14/14 [==============================] - 45s 3s/step - loss: 0.2282 - accuracy: 0.9195 - val_loss: 1.3171 - val_accuracy: 0.6364
Epoch 8/10
14/14 [==============================] - 44s 3s/step - loss: 0.1925 - accuracy: 0.9234 - val_loss: 3.4247 - val_accuracy: 0.4394
Epoch 9/10
14/14 [==============================] - 41s 3s/step - loss: 0.1427 - accuracy: 0.9349 - val_loss: 2.0372 - val_accuracy: 0.6667
Epoch 10/10
14/14 [==============================] - 44s 3s/step - loss: 0.1481 - accuracy: 0.9540 - val_loss: 4.8823 - val_accuracy: 0.3788
```

```
acc = history4.history['accuracy']
val_acc = history4.history['val_accuracy']
loss = history4.history['loss']
val_loss = history4.history['val_loss']
epochs_range = range(epochs)
plt.figure(figsize=(8,8))
plt.subplot(1, 2, 1)
plt.plot(epochs_range, acc, label='Training Accuracy')
plt.plot(epochs_range, val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')
plt.subplot(1, 2, 2)
plt.plot(epochs_range, loss, label='Training Loss')
plt.plot(epochs_range, val_loss, label='Validation Loss')
plt.legend(loc='upper right')
```

```
plt.title('Training and Validation Loss')
plt.show()
```