FUNDAMENTALS OF DEEP LEARNING

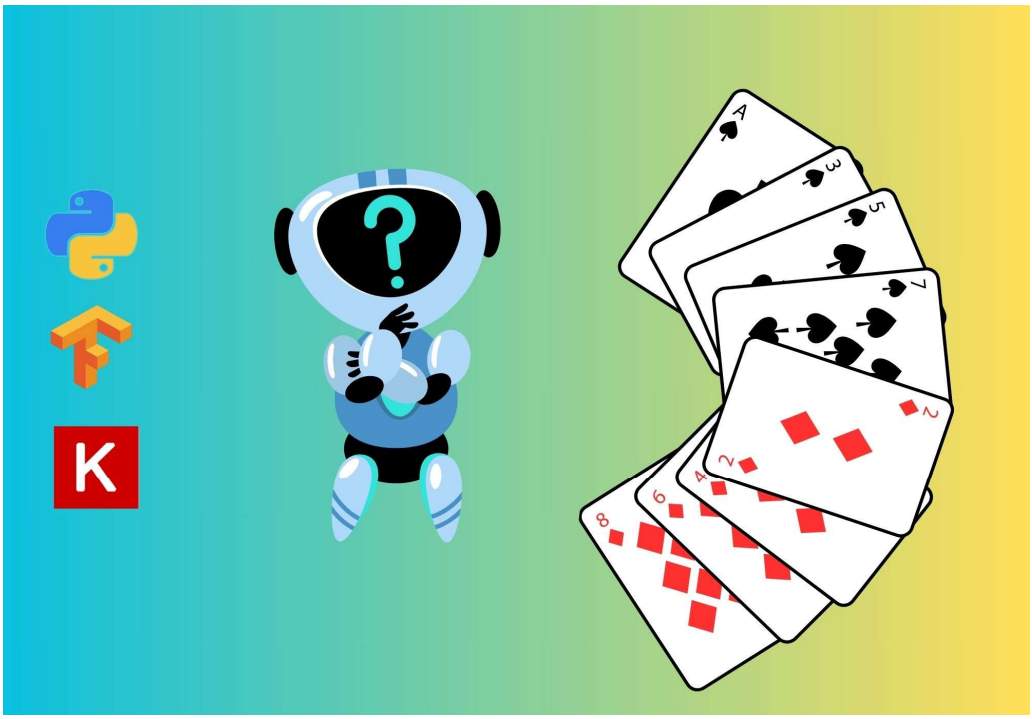
Build Convolutional Neural Networks and Transfer Learning Models with TensorFlow

STEP BY STEP EXPLNANATION FOR BUILDING IMAGE CLASSIFICATION MODELS

Shivang Kainthola

Convolutional Neural Networks with TensorFlow

In this article, we will build convolutional neural network models to classify images of playing cards, and examine the code step by step.



Given images of playing cards, the model has to classify the images into one of 53 types of cards.

We will use the cards classification dataset from Kaggle which has 7624 images in training split, 265 images in test split , 265 images in validation split with all color images in 224 X 224 X 3 jpg format.

The project follows the given steps for exploring the code for the task :

1) Downloading and opening the dataset from Kaggle

2) Making training, validation and testing splits from the dataset

3) Define functions for early stopping, learning rate scheduling and model checkpoint

4) Define a set of sequential layers for data augmentation

5) Build baseline CNN model, train it for 100 epochs without data augmentation and record its metrics after evaluating on the testing dataset.

5) Build a transfer learning CNN model, using EfficientNetB1 and including the data augmentation layers, train it for 100 epochs , and record its metrics after evaluating on the testing dataset.

6) Compare the performance metrics of both models and analyze the results.

The code is written in Python as a Jupyter Notebook.

Although it will work locally, I recommend running it on Kaggle or Google Colab with GPU resources, because training CNNs takes a lot of time on local CPU unless you have a solid GPU available locally.

Let us build this step-by-step :

1.1) Importing all necessary libraries :

```
import numpy as np
import cv2
import requests
import io
from PIL import Image
import pandas as pd
import tensorflow as tf
import os
import time
from tensorflow.keras.models import Sequential
from tensorflow.keras.models import Model
from tensorflow.keras.layers import GlobalAveragePooling2D, Dropout
from tensorflow.keras.layers import Dense, RandomFlip, RandomRotation, RandomHeight, RandomWidth,
RandomZoom, Rescaling
from tensorflow.keras import Sequential
from tensorflow.keras import layers
from keras.models import load_model, save_model
import matplotlib.pyplot as plt
from zipfile import ZipFile
from tensorflow.keras.utils import image_dataset_from_directory
```

1.2) Downloading the dataset

⟶ On running the command, which is available for every dataset hosted on Kaggle, the dataset is downloaded to the same directory as the notebook.

```
!kaggle datasets download -d gpiosenka/cards-image-datasetclassification
```

```
/opt/conda/lib/python3.10/pty.py:89: RuntimeWarning: os.fork() was called. os.fork() is incompa
tible with multithreaded code, and JAX is multithreaded, so this will likely lead to a deadloc
k.
  pid, fd = os.forkpty()

Dataset URL: https://www.kaggle.com/datasets/gpiosenka/cards-image-datasetclassification
License(s): CC0-1.0
Downloading cards-image-datasetclassification.zip to /kaggle/working
100%|                                     | 384M/385M [00:02<00:00, 211MB/s]
100%|                                     | 385M/385M [00:02<00:00, 179MB/s]
```

1.3) The dataset is downloaded as a compressed zip file.

⟶ We can extract all files to the same directory from the zip file using the ZipFile module.

```
# loading the temp.zip and creating a zip object
with ZipFile("cards-image-datasetclassification.zip", 'r') as zObject:

    # Extracting all the members of the zip
    # into a specific location.
    zObject.extractall()
```

1.4) Examining all folders within the directory :

⟶ The images are kept in folders train, test and valid respectively.

⟶ Each directory has sub-directories whose names are the name of category of the images i.e. the folder names are the names of the classes.

```
for dirpath, dirnames, filenames in os.walk("./"):
    print(f"There are {len(dirnames)} directories and {len(filenames)} images in '{dirpath}'.")
```

```
There are 4 directories and 4 images in './'.
There are 53 directories and 0 images in './test'.
There are 0 directories and 5 images in './test/jack of hearts'.
There are 0 directories and 5 images in './test/queen of diamonds'.
There are 0 directories and 5 images in './test/two of diamonds'.
There are 0 directories and 5 images in './test/eight of hearts'.
There are 0 directories and 5 images in './test/eight of spades'.
There are 0 directories and 5 images in './test/nine of spades'.
There are 0 directories and 5 images in './test/four of spades'.
There are 0 directories and 5 images in './test/six of clubs'.
There are 0 directories and 5 images in './test/four of diamonds'.
There are 0 directories and 5 images in './test/six of diamonds'.
There are 0 directories and 5 images in './test/queen of hearts'.
There are 0 directories and 5 images in './test/king of diamonds'.
There are 0 directories and 5 images in './test/seven of clubs'.
There are 0 directories and 5 images in './test/five of hearts'.
There are 0 directories and 5 images in './test/queen of clubs'.
```

1.5) Declaring parameters for the data :

⟶ batch size is a hyperparameter which controls the number of training examples used to compute the gradient of the loss function in each iteration.

⟶ The seed value ensures the model's results are reproducible.

⟶ *img_width*, *img_height* : are the dimensions of the image, all images in the dataset will be set to these dimensions

⟶ The *class_mode* parameters sets the approach for TensorFlow to process the images within the folders.

```
batch_size = 32
seed = 123
img_width, img_height = 224,224
class_m = "categorical"
```

1.6) Making training, testing and validation splits

⟶ Using TensorFlow's image_dataset_from_directory, we can build datasets from the folders containing the images.

⟶ The labels parameter is set to 'inferred', which means TensorFlow will label the image classes based on the name of the folder they are contained in.

⟶ All images are resized to the img_height and img_width dimensions (224,224) declared in step 1.5 .

```
train_dataset = image_dataset_from_directory(
    "./train",
    shuffle=True,
    seed=seed,
    labels = 'inferred',
    label_mode = 'categorical',
    image_size=(img_height, img_width),
    batch_size=batch_size)
```

```
Found 7624 files belonging to 53 classes.
```

```
valid_dataset = image_dataset_from_directory(
    "./valid",
    shuffle=False,
    seed=seed,
    labels = 'inferred',
    label_mode = 'categorical',
    image_size=(img_height, img_width),
    batch_size=batch_size)
```

```
Found 265 files belonging to 53 classes.
```

```
test_dataset = image_dataset_from_directory(
    "./test",
    shuffle=False,
    seed=seed,
    labels = 'inferred',
    label_mode = 'categorical',
    image_size=(img_height, img_width),
    batch_size=batch_size)
```

1.7) Getting the names of all classes/categories from the training dataset :

```
class_names = train_dataset.class_names
num_classes = len(class_names)
print(num_classes)
print("Class Names:", class_names)
```

```
53
Class Names: ['ace of clubs', 'ace of diamonds', 'ace of hearts', 'ace of spades', 'eight of cl
ubs', 'eight of diamonds', 'eight of hearts', 'eight of spades', 'five of clubs', 'five of diam
onds', 'five of hearts', 'five of spades', 'four of clubs', 'four of diamonds', 'four of heart
s', 'four of spades', 'jack of clubs', 'jack of diamonds', 'jack of hearts', 'jack of spades',
'joker', 'king of clubs', 'king of diamonds', 'king of hearts', 'king of spades', 'nine of club
s', 'nine of diamonds', 'nine of hearts', 'nine of spades', 'queen of clubs', 'queen of diamond
s', 'queen of hearts', 'queen of spades', 'seven of clubs', 'seven of diamonds', 'seven of hear
ts', 'seven of spades', 'six of clubs', 'six of diamonds', 'six of hearts', 'six of spades', 't
en of clubs', 'ten of diamonds', 'ten of hearts', 'ten of spades', 'three of clubs', 'three of
diamonds', 'three of hearts', 'three of spades', 'two of clubs', 'two of diamonds', 'two of hea
rts', 'two of spades']
```

1.8) Examining some pictures from the training dataset :

⟶ The .take(1) function extracts one image and its label from the train_dataset.

```
plt.figure(figsize=(10, 10))
for images, labels in train_dataset.take(1):
    for i in range(9):
        ax = plt.subplot(3, 3, i + 1)
        plt.imshow(images[i].numpy().astype("uint8"))
        label_index = np.argmax(labels[i].numpy())
        class_name = class_names[label_index]
        plt.title(class_name)
        plt.axis("off")
```

2) Tracking model performance

⟶ To record the metrics of each model in one place , we are making a dataframe.

⟶ After evaluating the models on testing datasets, we will save their metrics to this dataframe.

```
column_names1 = ['Model_Name','Accuracy','Precision','Training_time(s)']
datatypes = {'Model_Name': str, 'Accuracy': float, 'Precision': float,'Training_time(s)':float}
df = pd.DataFrame(columns=column_names1).astype(datatypes)
```

3) Defining functions for early stopping, model checkpoint and learning rate scheduling

3.1) Early Stopping

⟶ The EarlyStopping function in keras.callbacks module can be used to define a callback for stopping the fitting the neural network early.

⟶ The monitor parameter takes the metric that you TensorFlow to monitor, and the patience parameter is the number of epochs that TensorFlow will wait.

Here, TensorFlow will monitor the 'val_loss' (validation loss) of the neural network while it is training, and if the val_loss DOES NOT IMPROVE for 10 epochs, the training will be stopped.

⟶ We make a modified function for our baseline model, with a patience value of 15, to let it train leniently.

```
early_stopping_callback = tf.keras.callbacks.EarlyStopping(monitor="val_loss",patience=10)
```

Modifying for our baseline model : {-}

```
early_stopping_callback1 = tf.keras.callbacks.EarlyStopping(monitor="val_loss",patience=15)
```

3.2) Model Checkpoint

⟶ The model checkpoint saves the model instance for the best value of the monitored metric.

⟶ In the one we define here, the neural network is saved with the given file name, for the best training iteration where it has the maximum validation accuracy.

```
model_checkpoint = tf.keras.callbacks.ModelCheckpoint('best_baseline_model.keras', monitor='val_ac
curacy', save_best_only=True, mode='max')
```

3.3) Learning Rate reduction/scheduling

⟶ A high learning rate can help train the model faster, but it can also lead to overshooting the minima for the loss function.

⟶ Using the keras.callbacks.ReduceLROnPlateau() we can set the learning rate to be reduced after every given number of epochs (patience value) if the monitored metric plateaus i.e. doesn't increase or decrease .

```
reduce_lr_callback = tf.keras.callbacks.ReduceLROnPlateau(monitor="val_loss",
                                                          factor=0.2,
                                                          patience=5,
                                                          verbose=1,
                                                          min_lr=1e-7)
```

3.4) Defining a Data Augmentation layer

⟶ Data augmentation is the process of making some changes to images in the existing dataset to generate more training examples.

⟶ By flipping, rotating, zooming in, changing height/width and rescaling images, you can generate variations of the same image, and help the model learn better.

⟶ All these operations are carried out based on the declared authentication factor.

⟶ Here, we are building a sequential neural network (collection of layers) that carries out data augmentation on images. This can be added to our neural network.

```
AUGMENTATION_FACTOR = 0.1
```

```
augmentation_layer = Sequential([
    RandomFlip("horizontal", seed=123),
    RandomRotation(AUGMENTATION_FACTOR, seed=123),
    RandomZoom(AUGMENTATION_FACTOR, seed=123),
    RandomHeight(AUGMENTATION_FACTOR, seed=123),
    RandomWidth(AUGMENTATION_FACTOR, seed=123),
    Rescaling(1/255.),
], name="augmentation_layer")
```

3.5) Building the baseline CNN

⟶ Our baseline model will have :

1) An Input Layer with input_shape : 224 (height), 224(width), 3 (channels, RGB)

2) 4 Convolution blocks with

a Convolution Layer, Batch Normalization layer, Max Pooling layer and a Dropout layer each.

3) A Flatten layer

4) A Fully connected Dense layer with 256 units followed by another Dropout layer

5) An Output Layer having Softmax function as the activation function, with units equal to the number of classes.

```
model1 = Sequential([
  layers.Input(shape=(img_height, img_width, 3)),
  layers.Conv2D(32, 5, padding='same', activation='relu'),
  layers.BatchNormalization(),
  layers.MaxPooling2D(),
  layers.Dropout(0.2),

  layers.Conv2D(64, 5, padding='same', activation='relu'),
  layers.BatchNormalization(),
  layers.MaxPooling2D(),
  layers.Dropout(0.2),

  layers.Conv2D(128, 5, padding='same', activation='relu'),
  layers.BatchNormalization(),
  layers.MaxPooling2D(),
  layers.Dropout(0.2),

  layers.Conv2D(256, 5, padding='same', activation='relu'),
  layers.BatchNormalization(),
  layers.MaxPooling2D(),
  layers.Dropout(0.2),

  layers.Flatten(),
  layers.Dense(256, activation='relu'),
  layers.Dropout(0.2),
  layers.Dense(53, activation='softmax')
])
```

3.6) Compiling the baseline models

⟶ We set the optimizer to use the Adam optimizer.

⟶ The learning rate is kept to its default value i.e. 0.001.

⟶ Since our problem is *multi-class classification*, we can use the Categorical Crossentropy loss function.

⟶ The model will calculate its accuracy and precision for every evaluation on the validation/testing set, since they are passed to the metrics parameter.

```
model1.compile(optimizer='Adam',
               loss=tf.keras.losses.CategoricalCrossentropy(from_logits=True),
               metrics=['accuracy','precision'])
```

3.7) Verifying the structure of baseline model and number of parameters :

```
model1.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 224, 224, 32)	2,432
batch_normalization (BatchNormalization)	(None, 224, 224, 32)	128
max_pooling2d (MaxPooling2D)	(None, 112, 112, 32)	0
dropout (Dropout)	(None, 112, 112, 32)	0
conv2d_1 (Conv2D)	(None, 112, 112, 64)	51,264
batch_normalization_1 (BatchNormalization)	(None, 112, 112, 64)	256
max_pooling2d_1 (MaxPooling2D)	(None, 56, 56, 64)	0
dropout_1 (Dropout)	(None, 56, 56, 64)	0
conv2d_2 (Conv2D)	(None, 56, 56, 128)	204,928
batch_normalization_2 (BatchNormalization)	(None, 56, 56, 128)	512
max_pooling2d_2 (MaxPooling2D)	(None, 28, 28, 128)	0
dropout_2 (Dropout)	(None, 28, 28, 128)	0
conv2d_3 (Conv2D)	(None, 28, 28, 256)	819,456
batch_normalization_3 (BatchNormalization)	(None, 28, 28, 256)	1,024
max_pooling2d_3 (MaxPooling2D)	(None, 14, 14, 256)	0
dropout_3 (Dropout)	(None, 14, 14, 256)	0
flatten (Flatten)	(None, 50176)	0
dense (Dense)	(None, 256)	12,845,312
dropout_4 (Dropout)	(None, 256)	0
dense_1 (Dense)	(None, 53)	13,621

Total params: 13,938,933 (53.17 MB)
Trainable params: 13,937,973 (53.17 MB)
Non-trainable params: 960 (3.75 KB)

3.8) Fitting the model for 100 epochs :

⟶ Setting verbose = 1 in the fit() function ensures we will see the progression of the model's training and its metrics for each epoch.

⟶ All recorded metrics for every epoch is stored in the history1 variable.

⟶ Also recording the duration of the model fitting by using the time() functions.

```
start1=time.time()
history1 = model1.fit(
  train_dataset,
  validation_data=valid_dataset,
  epochs=100,verbose=1,
callbacks=[early_stopping_callback1,model_checkpoint,reduce_lr_callback])
```

```
Epoch 1/100

/opt/conda/lib/python3.10/site-packages/keras/src/backend/tensorflow/nn.py:560: UserWarning: "`
categorical_crossentropy` received `from_logits=True`, but the `output` argument was produced b
y a Softmax activation and thus does not represent logits. Was this intended?
  output, from_logits = _get_logits(

  2/239 ━━━━━━━━━━━━━━━━━━━━ 11s 50ms/step - accuracy: 0.0156 - loss: 17.4239 - precision: 0.030
3

WARNING: All log messages before absl::InitializeLog() is called are written to STDERR
I0000 00:00:1719741571.827157     120 device_compiler.h:186] Compiled cluster using XLA!  This
line is logged at most once for the lifetime of the process.

239/239 ━━━━━━━━━━━━━━━━━━━━ 46s 104ms/step - accuracy: 0.0435 - loss: 10.1870 - precision: 0.06
56 - val_accuracy: 0.0868 - val_loss: 4.8806 - val_precision: 0.1200 - learning_rate: 0.0010
Epoch 2/100
239/239 ━━━━━━━━━━━━━━━━━━━━ 11s 45ms/step - accuracy: 0.0748 - loss: 3.5851 - precision: 0.4034
- val_accuracy: 0.1094 - val_loss: 3.1953 - val_precision: 0.3333 - learning_rate: 0.0010
Epoch 3/100
```

```
239/239 ━━━━━━━━━━━━━━━━━━━━ 10s 42ms/step - accuracy: 0.9233 - loss: 0.2302 - precision: 0.9631
- val_accuracy: 0.9170 - val_loss: 0.5329 - val_precision: 0.9409 - learning_rate: 8.0000e-06
Epoch 65/100
239/239 ━━━━━━━━━━━━━━━━━━━━ 10s 42ms/step - accuracy: 0.9261 - loss: 0.2334 - precision: 0.9615
- val_accuracy: 0.9170 - val_loss: 0.5305 - val_precision: 0.9409 - learning_rate: 8.0000e-06
Epoch 66/100
239/239 ━━━━━━━━━━━━━━━━━━━━ 10s 42ms/step - accuracy: 0.9255 - loss: 0.2310 - precision: 0.9606
- val_accuracy: 0.9170 - val_loss: 0.5313 - val_precision: 0.9412 - learning_rate: 8.0000e-06
Epoch 67/100
239/239 ━━━━━━━━━━━━━━━━━━━━ 0s 41ms/step - accuracy: 0.9227 - loss: 0.2408 - precision: 0.9600
Epoch 67: ReduceLROnPlateau reducing learning rate to 1.6000001778593287e-06.
239/239 ━━━━━━━━━━━━━━━━━━━━ 10s 42ms/step - accuracy: 0.9227 - loss: 0.2408 - precision: 0.9600
- val_accuracy: 0.9170 - val_loss: 0.5333 - val_precision: 0.9412 - learning_rate: 8.0000e-06
```

```
end1=time.time()
```

⟶ The fitting stopped at the 67th epoch as the Early Stopping callback function was invoked.

3.9) Visualizing the performance of the neural network over the epochs :

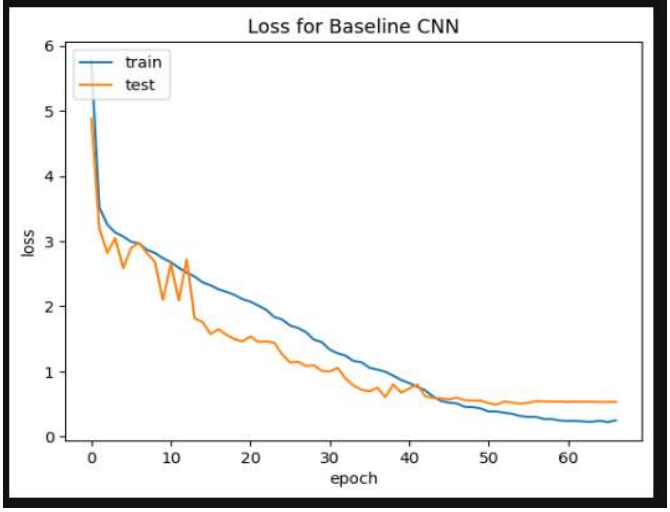
⟶ The accuracy and loss values for all epochs can be accessed from the history1 variable.

⟶ Accuracy measures the proportion of correctly classified samples out of all samples in the dataset.

⟶ Loss, also known as the cost function or objective function, measures the difference between the model's predictions and the actual true labels.

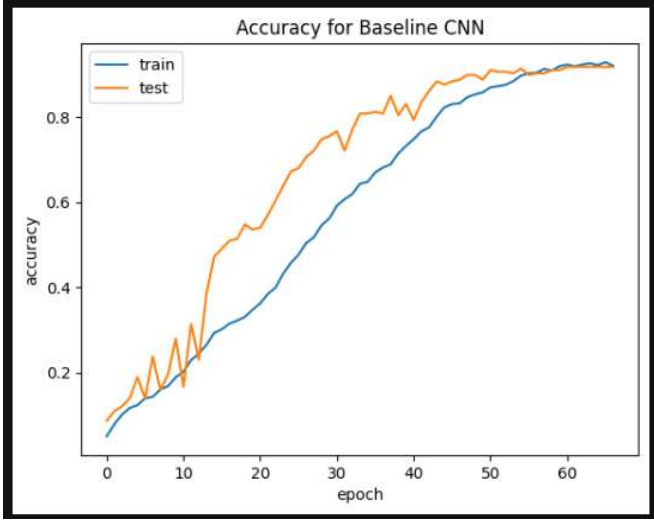
```
plt.plot(history1.history['loss'])
plt.plot(history1.history['val_loss'])
plt.title('Loss for Baseline CNN')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
```

```
<matplotlib.legend.Legend at 0x79500022bd90>
```




```
plt.plot(history1.history['accuracy'])
plt.plot(history1.history['val_accuracy'])
plt.title('Accuracy for Baseline CNN')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
```

```
<matplotlib.legend.Legend at 0x795024218520>
```



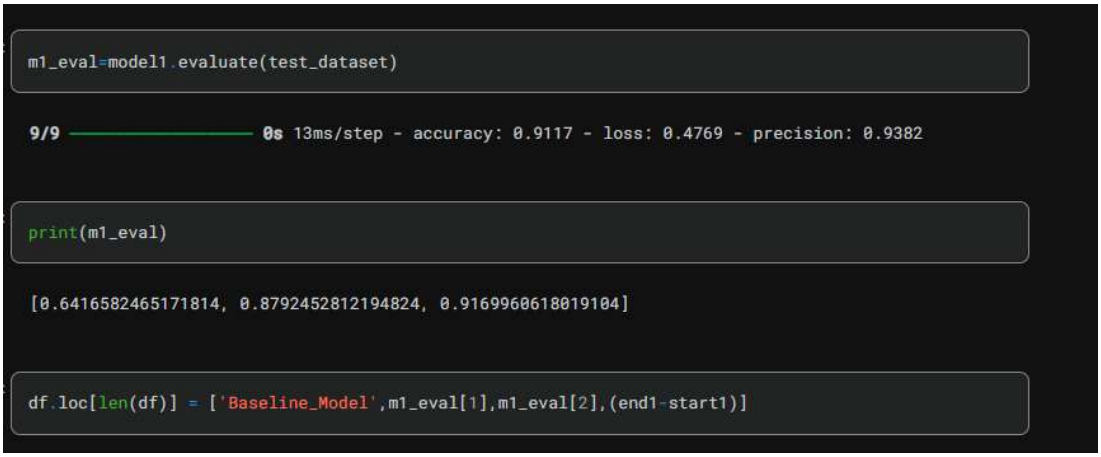
⟶ The training and validation loss have both decreased consistently and *seemingly* plateauing after the 60th epoch.

⟶ Training accuracy is increasing consistently and plateauing off after around 90%. Validation accuracy initially oscillates, diverging from the training accuracy between the 10th and 50th epoch, but finally converges with the training accuracy curve.

3.10) Evaluating the neural network on testing dataset :

⟶ We evaluate the model on the testing dataset.

⟶ The calculated metrics are then stored in the dataframe declared earlier to store the model performance metrics.



4) Building the transfer learning model with EfficientNetB1

4.1) Downloading the pre-trained base model

⟶ From tf.keras.applications, we are importing EfficientNetB1.

⟶ The parameter include_top being set to False ensures we only import the base (early) layers and also passing the input_shape argument for our image sizes.

⟶ We will set the model.training = False because we want to use the pre-trained weights and not train it further.

4.2) Building the transfer learning model :

⟶ Our transfer learning neural network will have :

1) An Input layer

2) Data augmentation layer (defined in step 3.4)

3) EfficientNetB1 base layers

4) A Global Average Pooling layer

5) A fully connected Dense layer with 256 units followed by a Batch Normalization layer

6) An output layer similar to our first model

```
model2 = tf.keras.Sequential([
    tf.keras.layers.Input(shape=(img_height,img_width) + (3, ), name="input_layer"),
    augmentation_layer,
    efnetb1,
    tf.keras.layers.GlobalAveragePooling2D(name="global_average_pooling_layer"),
    tf.keras.layers.Dense(256, activation=tf.keras.activations.relu),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Dense(len(class_names), activation=tf.keras.activations.softmax)
])
```

4.3) Compiling the neural network :

```
model2.compile(optimizer='Adam',
               loss=tf.keras.losses.CategoricalCrossentropy(from_logits=True),
               metrics=['accuracy','precision'])
```

4.4) Examining the structure of the network :

```
model2.summary()
```

Model: "sequential_1"

Layer (type)	Output Shape	Param #
augmentation_layer (Sequential)	(None, None, None, 3)	0
efficientnetb1 (Functional)	(None, None, None, 1280)	6,575,239
global_average_pooling_layer (GlobalAveragePooling2D)	(None, 1280)	0
dense_2 (Dense)	(None, 256)	327,936
batch_normalization_4 (BatchNormalization)	(None, 256)	1,024
dense_3 (Dense)	(None, 53)	13,621

Total params: 6,917,820 (26.39 MB)
Trainable params: 6,855,253 (26.15 MB)
Non-trainable params: 62,567 (244.41 KB)

4.5) Defining model checkpoint callback for the model and fitting it for 100 epochs :

```
model2_checkpoint = tf.keras.callbacks.ModelCheckpoint('best_efnetb0_model.keras', monitor='val_accuracy', save_best_only=True, mode='max')
```

```
start2=time.time()
history2 = model2.fit(
  train_dataset,
  validation_data=valid_dataset,
  epochs=100,verbose=1,callbacks=[early_stopping_callback,model2_checkpoint,reduce_lr_callback])
```

```
Epoch 27/100
239/239 ━━━━━━━━━━━━━━━━━━━━ 85s 357ms/step - accuracy: 0.9849 - loss: 0.0463 - precision: 0.987
7 - val_accuracy: 0.9736 - val_loss: 0.0711 - val_precision: 0.9773 - learning_rate: 4.0000e-05
Epoch 28/100
239/239 ━━━━━━━━━━━━━━━━━━━━ 86s 359ms/step - accuracy: 0.9880 - loss: 0.0349 - precision: 0.989
9 - val_accuracy: 0.9736 - val_loss: 0.0681 - val_precision: 0.9736 - learning_rate: 4.0000e-05
Epoch 29/100
239/239 ━━━━━━━━━━━━━━━━━━━━ 87s 362ms/step - accuracy: 0.9920 - loss: 0.0248 - precision: 0.993
9 - val_accuracy: 0.9698 - val_loss: 0.0697 - val_precision: 0.9698 - learning_rate: 4.0000e-05
Epoch 30/100
239/239 ━━━━━━━━━━━━━━━━━━━━ 0s 358ms/step - accuracy: 0.9936 - loss: 0.0245 - precision: 0.9955
Epoch 30: ReduceLROnPlateau reducing learning rate to 8.000000525498762e-06.
239/239 ━━━━━━━━━━━━━━━━━━━━ 86s 360ms/step - accuracy: 0.9936 - loss: 0.0245 - precision: 0.995
5 - val_accuracy: 0.9660 - val_loss: 0.0714 - val_precision: 0.9660 - learning_rate: 4.0000e-05
```

```
end2=time.time()
```

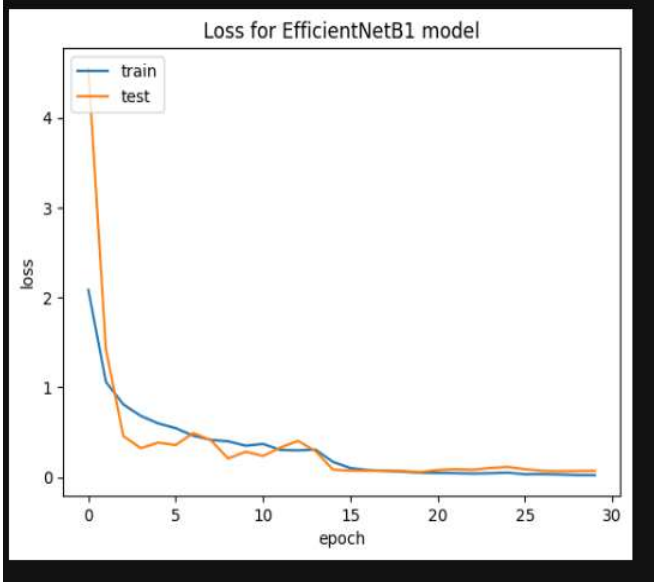
⟶ The model fitting stops at the 30th epoch.

⟶ Notice, how this transfer learning model converged to training accuracy of around 99% within 30 epochs, whereas our baseline CNN model reached values around 92% by the 67th epoch.

4.6) Visualizing the transfer learning model's training :

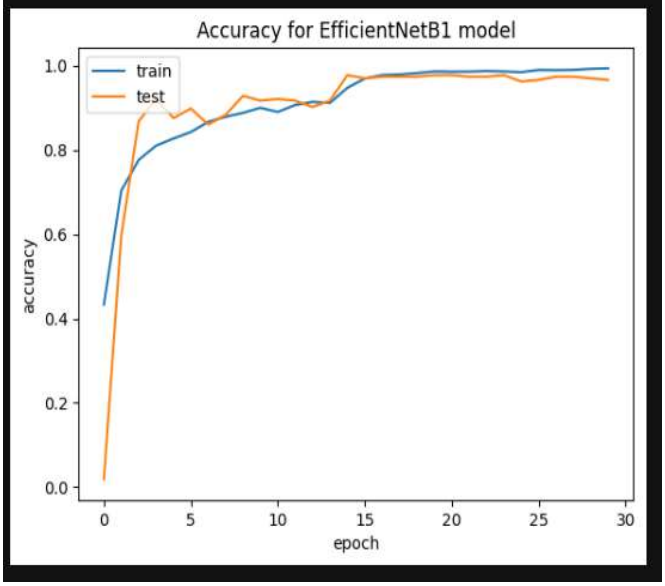
```
plt.plot(history2.history['loss'])
plt.plot(history2.history['val_loss'])
plt.title('Loss for EfficientNetB1 model')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
```

```
<matplotlib.legend.Legend at 0x794b81c53be0>
```




```
plt.plot(history2.history['accuracy'])
plt.plot(history2.history['val_accuracy'])
plt.title('Accuracy for EfficientNetB1 model')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
```

```
<matplotlib.legend.Legend at 0x794b334e98a0>
```



⟶ The training as well as validation loss curves appear to almost be merging after the 15th epoch, and plateau towards zero.

⟶ Similar to the loss curve, the training and validation accuracy curves also appear to be very close, almost reaching 100%.

4.8) Evaluating the transfer learning model on the testing dataset and saving its metrics :

```
m2_eval=model2.evaluate(test_dataset)
```

```
9/9 ━━━━━━━━━━━━━━━━━━━━ 1s 61ms/step - accuracy: 0.9606 - loss: 0.1418 - precision: 0.9720
```

```
df.loc[len(df)] = ['EfficientNetB1',m2_eval[1],m2_eval[2],(end2-start2)]
```

5) Comparing the performance of both models :

```
print(df)
```

	Model_Name	Accuracy	Precision	Training_time(s)
0	Baseline_Model	0.879245	0.916996	747.207493
1	EfficientNetB1	0.943396	0.957854	2800.077686

6) Analyzing the results :

⟶ The baseline model achieved good performance but the transfer learning model achieved higher accuracy and precision within 30 epochs, but every epoch/training step took considerably longer time.

⟶ Here are some questions you should think about :

1) Which model would work better, if we had more data ?

2) How would the baseline model work if we included the data augmentation layer ?

3) How well would either of these models perform on any unseen picture of playing cards, if you could test them on a picture you upload yourself ?

4) Is the fitting time (and resources spent) on the transfer learning model worth its performance ? In what scenarios is it not a good trade-off ?

5) Should you train the models further, which one and why ?

Feel free to get the code and continue experimenting on your own!

Take a look at the entire code in action for this project :

cards-classification-vanila-cnn-vs-efficientnet

July 1, 2024

#

Cards classification using Convolutional Neural Networks {-}

Comparing performance of vanilla CNN with pre-trained transfer learning model

1. Importing all libraries :

[93]:
```
import numpy as np
import cv2
import requests
import io
from PIL import Image
import pandas as pd
import tensorflow as tf
import os
import time
from tensorflow.keras.models import Sequential
from tensorflow.keras.models import Model
from tensorflow.keras.layers import GlobalAveragePooling2D, Dropout
from tensorflow.keras.layers import Dense, RandomFlip, RandomRotation,␣
 ↪RandomHeight, RandomWidth, RandomZoom, Rescaling
from tensorflow.keras import Sequential
from tensorflow.keras import layers
from keras.models import load_model, save_model
import matplotlib.pyplot as plt
from zipfile import ZipFile
from tensorflow.keras.utils import image_dataset_from_directory
```

2. Downloading dataset and accessing directories

[2]:
```
!kaggle datasets download -d gpiosenka/cards-image-datasetclassification
```

```
/opt/conda/lib/python3.10/pty.py:89: RuntimeWarning: os.fork() was called.
os.fork() is incompatible with multithreaded code, and JAX is multithreaded, so
this will likely lead to a deadlock.
  pid, fd = os.forkpty()
```

```
Dataset URL: https://www.kaggle.com/datasets/gpiosenka/cards-image-
datasetclassification
```

```
License(s): CC0-1.0
Downloading cards-image-datasetclassification.zip to /kaggle/working
100%|                     | 384M/385M [00:02<00:00, 211MB/s]
100%|                     | 385M/385M [00:02<00:00, 179MB/s]
```

[3]:
```python
# loading the temp.zip and creating a zip object
with ZipFile("cards-image-datasetclassification.zip", 'r') as zObject:

    # Extracting all the members of the zip
    # into a specific location.
    zObject.extractall()
```

[6]:
```python
for dirpath, dirnames, filenames in os.walk("./"):
    print(f"There are {len(dirnames)} directories and {len(filenames)} images␣
 ↪in '{dirpath}'.")
```

```
There are 4 directories and 4 images in './'.
There are 53 directories and 0 images in './test'.
There are 0 directories and 5 images in './test/jack of hearts'.
There are 0 directories and 5 images in './test/queen of diamonds'.
There are 0 directories and 5 images in './test/two of diamonds'.
There are 0 directories and 5 images in './test/eight of hearts'.
There are 0 directories and 5 images in './test/eight of spades'.
There are 0 directories and 5 images in './test/nine of spades'.
There are 0 directories and 5 images in './test/four of spades'.
There are 0 directories and 5 images in './test/six of clubs'.
There are 0 directories and 5 images in './test/four of diamonds'.
There are 0 directories and 5 images in './test/six of diamonds'.
There are 0 directories and 5 images in './test/queen of hearts'.
There are 0 directories and 5 images in './test/king of diamonds'.
There are 0 directories and 5 images in './test/seven of clubs'.
There are 0 directories and 5 images in './test/five of hearts'.
There are 0 directories and 5 images in './test/queen of clubs'.
There are 0 directories and 5 images in './test/five of diamonds'.
There are 0 directories and 5 images in './test/eight of diamonds'.
There are 0 directories and 5 images in './test/four of clubs'.
There are 0 directories and 5 images in './test/six of spades'.
There are 0 directories and 5 images in './test/seven of diamonds'.
There are 0 directories and 5 images in './test/two of clubs'.
There are 0 directories and 5 images in './test/jack of diamonds'.
There are 0 directories and 5 images in './test/four of hearts'.
There are 0 directories and 5 images in './test/jack of spades'.
There are 0 directories and 5 images in './test/queen of spades'.
There are 0 directories and 5 images in './test/eight of clubs'.
There are 0 directories and 5 images in './test/three of spades'.
There are 0 directories and 5 images in './test/nine of diamonds'.
There are 0 directories and 5 images in './test/nine of hearts'.
There are 0 directories and 5 images in './test/ace of hearts'.
```

```
There are 0 directories and 5 images in './test/two of spades'.
There are 0 directories and 5 images in './test/joker'.
There are 0 directories and 5 images in './test/king of spades'.
There are 0 directories and 5 images in './test/three of hearts'.
There are 0 directories and 5 images in './test/ten of hearts'.
There are 0 directories and 5 images in './test/ten of diamonds'.
There are 0 directories and 5 images in './test/five of spades'.
There are 0 directories and 5 images in './test/ace of spades'.
There are 0 directories and 5 images in './test/five of clubs'.
There are 0 directories and 5 images in './test/ace of diamonds'.
There are 0 directories and 5 images in './test/ace of clubs'.
There are 0 directories and 5 images in './test/three of diamonds'.
There are 0 directories and 5 images in './test/nine of clubs'.
There are 0 directories and 5 images in './test/two of hearts'.
There are 0 directories and 5 images in './test/king of clubs'.
There are 0 directories and 5 images in './test/seven of spades'.
There are 0 directories and 5 images in './test/ten of spades'.
There are 0 directories and 5 images in './test/jack of clubs'.
There are 0 directories and 5 images in './test/seven of hearts'.
There are 0 directories and 5 images in './test/three of clubs'.
There are 0 directories and 5 images in './test/ten of clubs'.
There are 0 directories and 5 images in './test/king of hearts'.
There are 0 directories and 5 images in './test/six of hearts'.
There are 53 directories and 0 images in './valid'.
There are 0 directories and 5 images in './valid/jack of hearts'.
There are 0 directories and 5 images in './valid/queen of diamonds'.
There are 0 directories and 5 images in './valid/two of diamonds'.
There are 0 directories and 5 images in './valid/eight of hearts'.
There are 0 directories and 5 images in './valid/eight of spades'.
There are 0 directories and 5 images in './valid/nine of spades'.
There are 0 directories and 5 images in './valid/four of spades'.
There are 0 directories and 5 images in './valid/six of clubs'.
There are 0 directories and 5 images in './valid/four of diamonds'.
There are 0 directories and 5 images in './valid/six of diamonds'.
There are 0 directories and 5 images in './valid/queen of hearts'.
There are 0 directories and 5 images in './valid/king of diamonds'.
There are 0 directories and 5 images in './valid/seven of clubs'.
There are 0 directories and 5 images in './valid/five of hearts'.
There are 0 directories and 5 images in './valid/queen of clubs'.
There are 0 directories and 5 images in './valid/five of diamonds'.
There are 0 directories and 5 images in './valid/eight of diamonds'.
There are 0 directories and 5 images in './valid/four of clubs'.
There are 0 directories and 5 images in './valid/six of spades'.
There are 0 directories and 5 images in './valid/seven of diamonds'.
There are 0 directories and 5 images in './valid/two of clubs'.
There are 0 directories and 5 images in './valid/jack of diamonds'.
There are 0 directories and 5 images in './valid/four of hearts'.
There are 0 directories and 5 images in './valid/jack of spades'.
```

```
There are 0 directories and 5 images in './valid/queen of spades'.
There are 0 directories and 5 images in './valid/eight of clubs'.
There are 0 directories and 5 images in './valid/three of spades'.
There are 0 directories and 5 images in './valid/nine of diamonds'.
There are 0 directories and 5 images in './valid/nine of hearts'.
There are 0 directories and 5 images in './valid/ace of hearts'.
There are 0 directories and 5 images in './valid/two of spades'.
There are 0 directories and 5 images in './valid/joker'.
There are 0 directories and 5 images in './valid/king of spades'.
There are 0 directories and 5 images in './valid/three of hearts'.
There are 0 directories and 5 images in './valid/ten of hearts'.
There are 0 directories and 5 images in './valid/ten of diamonds'.
There are 0 directories and 5 images in './valid/five of spades'.
There are 0 directories and 5 images in './valid/ace of spades'.
There are 0 directories and 5 images in './valid/five of clubs'.
There are 0 directories and 5 images in './valid/ace of diamonds'.
There are 0 directories and 5 images in './valid/ace of clubs'.
There are 0 directories and 5 images in './valid/three of diamonds'.
There are 0 directories and 5 images in './valid/nine of clubs'.
There are 0 directories and 5 images in './valid/two of hearts'.
There are 0 directories and 5 images in './valid/king of clubs'.
There are 0 directories and 5 images in './valid/seven of spades'.
There are 0 directories and 5 images in './valid/ten of spades'.
There are 0 directories and 5 images in './valid/jack of clubs'.
There are 0 directories and 5 images in './valid/seven of hearts'.
There are 0 directories and 5 images in './valid/three of clubs'.
There are 0 directories and 5 images in './valid/ten of clubs'.
There are 0 directories and 5 images in './valid/king of hearts'.
There are 0 directories and 5 images in './valid/six of hearts'.
There are 53 directories and 0 images in './train'.
There are 0 directories and 168 images in './train/jack of hearts'.
There are 0 directories and 163 images in './train/queen of diamonds'.
There are 0 directories and 133 images in './train/two of diamonds'.
There are 0 directories and 152 images in './train/eight of hearts'.
There are 0 directories and 135 images in './train/eight of spades'.
There are 0 directories and 154 images in './train/nine of spades'.
There are 0 directories and 140 images in './train/four of spades'.
There are 0 directories and 152 images in './train/six of clubs'.
There are 0 directories and 114 images in './train/four of diamonds'.
There are 0 directories and 139 images in './train/six of diamonds'.
There are 0 directories and 139 images in './train/queen of hearts'.
There are 0 directories and 135 images in './train/king of diamonds'.
There are 0 directories and 108 images in './train/seven of clubs'.
There are 0 directories and 136 images in './train/five of hearts'.
There are 0 directories and 161 images in './train/queen of clubs'.
There are 0 directories and 138 images in './train/five of diamonds'.
There are 0 directories and 159 images in './train/eight of diamonds'.
There are 0 directories and 157 images in './train/four of clubs'.
```

```
There are 0 directories and 158 images in './train/six of spades'.
There are 0 directories and 124 images in './train/seven of diamonds'.
There are 0 directories and 130 images in './train/two of clubs'.
There are 0 directories and 160 images in './train/jack of diamonds'.
There are 0 directories and 154 images in './train/four of hearts'.
There are 0 directories and 172 images in './train/jack of spades'.
There are 0 directories and 162 images in './train/queen of spades'.
There are 0 directories and 138 images in './train/eight of clubs'.
There are 0 directories and 142 images in './train/three of spades'.
There are 0 directories and 129 images in './train/nine of diamonds'.
There are 0 directories and 133 images in './train/nine of hearts'.
There are 0 directories and 171 images in './train/ace of hearts'.
There are 0 directories and 155 images in './train/two of spades'.
There are 0 directories and 115 images in './train/joker'.
There are 0 directories and 151 images in './train/king of spades'.
There are 0 directories and 113 images in './train/three of hearts'.
There are 0 directories and 129 images in './train/ten of hearts'.
There are 0 directories and 151 images in './train/ten of diamonds'.
There are 0 directories and 158 images in './train/five of spades'.
There are 0 directories and 181 images in './train/ace of spades'.
There are 0 directories and 150 images in './train/five of clubs'.
There are 0 directories and 129 images in './train/ace of diamonds'.
There are 0 directories and 120 images in './train/ace of clubs'.
There are 0 directories and 153 images in './train/three of diamonds'.
There are 0 directories and 124 images in './train/nine of clubs'.
There are 0 directories and 155 images in './train/two of hearts'.
There are 0 directories and 128 images in './train/king of clubs'.
There are 0 directories and 165 images in './train/seven of spades'.
There are 0 directories and 158 images in './train/ten of spades'.
There are 0 directories and 171 images in './train/jack of clubs'.
There are 0 directories and 143 images in './train/seven of hearts'.
There are 0 directories and 126 images in './train/three of clubs'.
There are 0 directories and 141 images in './train/ten of clubs'.
There are 0 directories and 125 images in './train/king of hearts'.
There are 0 directories and 127 images in './train/six of hearts'.
There are 0 directories and 0 images in './.virtual_documents'.
```

3. Preparing Training, Testing and Validation splits :

3.1 Declaring batch size, seed, image dimensions :

[4]:
```
batch_size = 32
seed = 123
img_width, img_height = 224,224
class_m = "categorical"
```

3.2 Making training, testing and validation sets from directories :

[5]:
```
train_dataset = image_dataset_from_directory(
    "./train",
    shuffle=True,
    seed=seed,
    labels = 'inferred',
    label_mode = 'categorical',
    image_size=(img_height, img_width),
    batch_size=batch_size)
```

```
Found 7624 files belonging to 53 classes.
```

[6]:
```
valid_dataset = image_dataset_from_directory(
    "./valid",
    shuffle=False,
    seed=seed,
    labels = 'inferred',
    label_mode = 'categorical',
    image_size=(img_height, img_width),
    batch_size=batch_size)
```

```
Found 265 files belonging to 53 classes.
```

[7]:
```
test_dataset = image_dataset_from_directory(
    "./test",
    shuffle=False,
    seed=seed,
    labels = 'inferred',
    label_mode = 'categorical',
    image_size=(img_height, img_width),
    batch_size=batch_size)
```

```
Found 265 files belonging to 53 classes.
```

3.3 Getting class names :

[8]:
```
class_names = train_dataset.class_names
num_classes = len(class_names)
print(num_classes)
print("Class Names:", class_names)
```

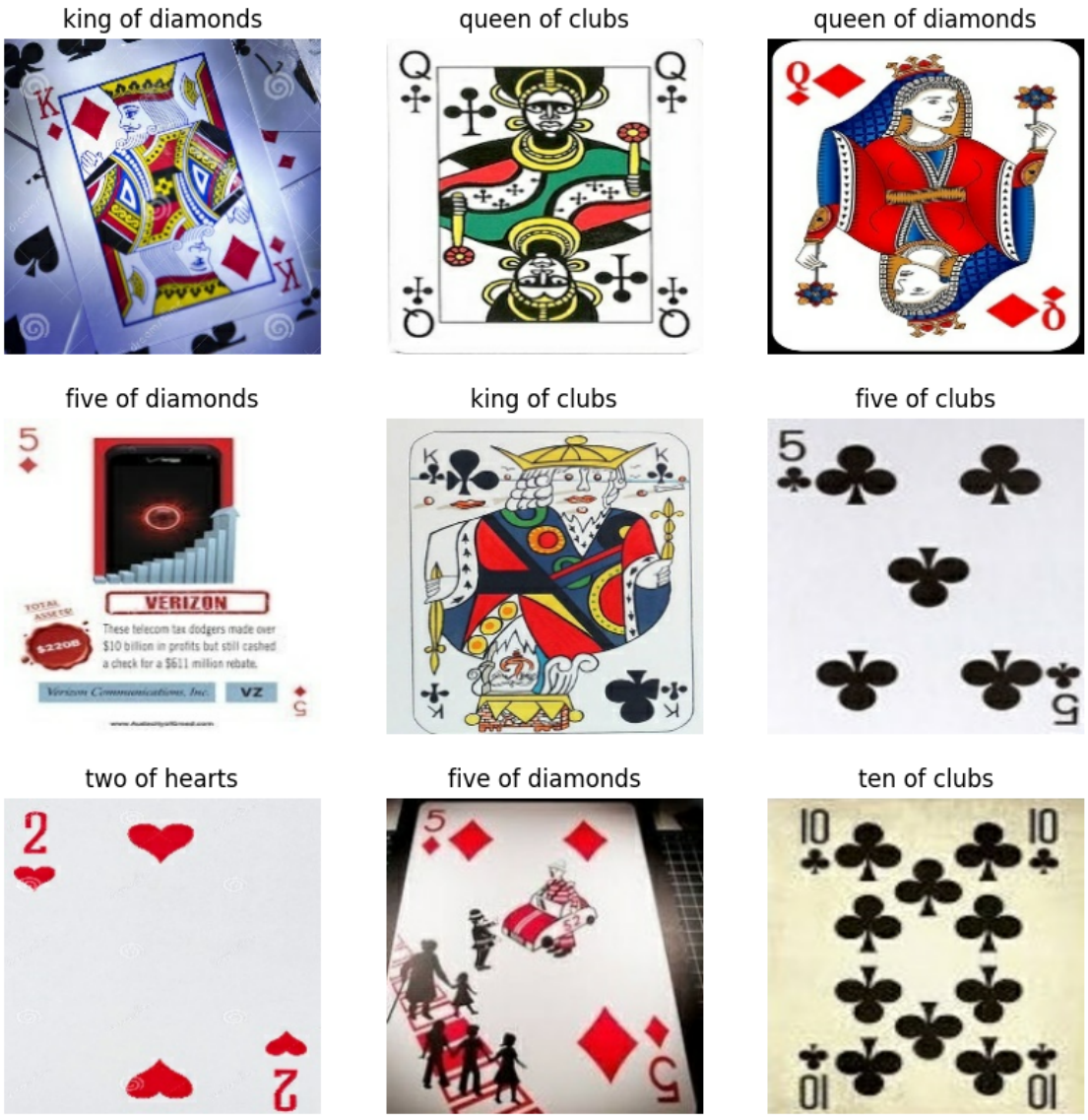
```
53
Class Names: ['ace of clubs', 'ace of diamonds', 'ace of hearts', 'ace of
spades', 'eight of clubs', 'eight of diamonds', 'eight of hearts', 'eight of
spades', 'five of clubs', 'five of diamonds', 'five of hearts', 'five of
spades', 'four of clubs', 'four of diamonds', 'four of hearts', 'four of
spades', 'jack of clubs', 'jack of diamonds', 'jack of hearts', 'jack of
spades', 'joker', 'king of clubs', 'king of diamonds', 'king of hearts', 'king
of spades', 'nine of clubs', 'nine of diamonds', 'nine of hearts', 'nine of
spades', 'queen of clubs', 'queen of diamonds', 'queen of hearts', 'queen of
spades', 'seven of clubs', 'seven of diamonds', 'seven of hearts', 'seven of
```

```
spades', 'six of clubs', 'six of diamonds', 'six of hearts', 'six of spades',
'ten of clubs', 'ten of diamonds', 'ten of hearts', 'ten of spades', 'three of
clubs', 'three of diamonds', 'three of hearts', 'three of spades', 'two of
clubs', 'two of diamonds', 'two of hearts', 'two of spades']
```

---

4. Visualising some images from the training dataset :

[9]:
```
plt.figure(figsize=(10, 10))
for images, labels in train_dataset.take(1):
      for i in range(9):
        ax = plt.subplot(3, 3, i + 1)
        plt.imshow(images[i].numpy().astype("uint8"))
        label_index = np.argmax(labels[i].numpy())
        class_name = class_names[label_index]
        plt.title(class_name)
        plt.axis("off")
```



---

5. Making a dataframe to track the performance of different models :

[11]:
```
column_names1 = ['Model_Name','Accuracy','Precision','Training_time(s)']
datatypes = {'Model_Name': str, 'Accuracy': float, 'Precision':␣
 ↪float,'Training_time(s)':float}
df = pd.DataFrame(columns=column_names1).astype(datatypes)
```

6. Building CNN models

Defining functions for Early Stopping, Learning Rate Scheduling and Saving best model checkpoint:

[12]:
```
early_stopping_callback = tf.keras.callbacks.
↪EarlyStopping(monitor="val_loss",patience=10)
```

Modifying for our baseline model : {-}

[14]:
```
early_stopping_callback1 = tf.keras.callbacks.
↪EarlyStopping(monitor="val_loss",patience=15)
```

[15]:
```
model_checkpoint = tf.keras.callbacks.ModelCheckpoint('best_baseline_model.
↪keras', monitor='val_accuracy', save_best_only=True, mode='max')
```

[16]:
```
reduce_lr_callback = tf.keras.callbacks.ReduceLROnPlateau(monitor="val_loss",
                                                          factor=0.2,
                                                          patience=5,
                                                          verbose=1,
                                                          min_lr=1e-7)
```

Defining a layer for data augmentation :

[17]:
```
AUGMENTATION_FACTOR = 0.1
```

[18]:
```
augmentation_layer = Sequential([
    RandomFlip("horizontal", seed=123),
    RandomRotation(AUGMENTATION_FACTOR, seed=123),
    RandomZoom(AUGMENTATION_FACTOR, seed=123),
    RandomHeight(AUGMENTATION_FACTOR, seed=123),
    RandomWidth(AUGMENTATION_FACTOR, seed=123),
    Rescaling(1/255.),
], name="augmentation_layer")
```

6.1 Building baseline CNN model :

- This model has:

1) Four convolutional layers
2) Max Pooling layers
3) Batch Normalization layers
4) One Dense Layer

! Without any data augmentation.

[19]:
```
model1 = Sequential([
  layers.Input(shape=(img_height, img_width, 3)),
  layers.Conv2D(32, 5, padding='same', activation='relu'),
  layers.BatchNormalization(),
```

```
  layers.MaxPooling2D(),
  layers.Dropout(0.2),

  layers.Conv2D(64, 5, padding='same', activation='relu'),
  layers.BatchNormalization(),
  layers.MaxPooling2D(),
  layers.Dropout(0.2),

  layers.Conv2D(128, 5, padding='same', activation='relu'),
  layers.BatchNormalization(),
  layers.MaxPooling2D(),
  layers.Dropout(0.2),

  layers.Conv2D(256, 5, padding='same', activation='relu'),
  layers.BatchNormalization(),
  layers.MaxPooling2D(),
  layers.Dropout(0.2),

  layers.Flatten(),
  layers.Dense(256, activation='relu'),
  layers.Dropout(0.2),
  layers.Dense(53, activation='softmax')
])
```

6.1.1 Compiling baseline model and checking its structure:

[20]:
```
model1.compile(optimizer='Adam',
               loss=tf.keras.losses.CategoricalCrossentropy(from_logits=True),
               metrics=['accuracy','precision'])
```

[21]:
```
model1.summary()
```

```
Model: "sequential"
```

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 224, 224, 32)	2,432
batch_normalization (BatchNormalization)	(None, 224, 224, 32)	128
max_pooling2d (MaxPooling2D)	(None, 112, 112, 32)	0
dropout (Dropout)	(None, 112, 112, 32)	0
conv2d_1 (Conv2D)	(None, 112, 112, 64)	51,264

```
batch_normalization_1 (BatchNormalization)   (None, 112, 112, 64)        256
max_pooling2d_1 (MaxPooling2D)               (None, 56, 56, 64)          0
dropout_1 (Dropout)                          (None, 56, 56, 64)          0
conv2d_2 (Conv2D)                            (None, 56, 56, 128)       204,928
batch_normalization_2 (BatchNormalization)   (None, 56, 56, 128)         512
max_pooling2d_2 (MaxPooling2D)               (None, 28, 28, 128)         0
dropout_2 (Dropout)                          (None, 28, 28, 128)         0
conv2d_3 (Conv2D)                            (None, 28, 28, 256)       819,456
batch_normalization_3 (BatchNormalization)   (None, 28, 28, 256)       1,024
max_pooling2d_3 (MaxPooling2D)               (None, 14, 14, 256)         0
dropout_3 (Dropout)                          (None, 14, 14, 256)         0
flatten (Flatten)                            (None, 50176)               0
dense (Dense)                                (None, 256)            12,845,312
dropout_4 (Dropout)                          (None, 256)                 0
dense_1 (Dense)                              (None, 53)                13,621

Total params: 13,938,933 (53.17 MB)

Trainable params: 13,937,973 (53.17 MB)

Non-trainable params: 960 (3.75 KB)
```

6.1.2 Fitting the model for 100 epochs with Early Stopping, LR scheduling and model checkpoint callbacks :

[22]:
```
start1=time.time()
history1 = model1.fit(
  train_dataset,
```

```
  validation_data=valid_dataset,
  epochs=100,verbose=1,
callbacks=[early_stopping_callback1,model_checkpoint,reduce_lr_callback])
```

```
Epoch 1/100
```

```
/opt/conda/lib/python3.10/site-packages/keras/src/backend/tensorflow/nn.py:560:
UserWarning: "`categorical_crossentropy` received `from_logits=True`, but the
`output` argument was produced by a Softmax activation and thus does not
represent logits. Was this intended?
  output, from_logits = _get_logits(
```

```
  2/239             11s 50ms/step - accuracy:
0.0156 - loss: 17.4239 - precision: 0.0303
```

```
WARNING: All log messages before absl::InitializeLog() is called are written to
STDERR
I0000 00:00:1719741571.827157     120 device_compiler.h:186] Compiled cluster
using XLA!  This line is logged at most once for the lifetime of the process.
```

```
239/239             46s 104ms/step -
accuracy: 0.0435 - loss: 10.1870 - precision: 0.0656 - val_accuracy: 0.0868 -
val_loss: 4.8806 - val_precision: 0.1200 - learning_rate: 0.0010
Epoch 2/100
239/239             11s 45ms/step -
accuracy: 0.0748 - loss: 3.5851 - precision: 0.4034 - val_accuracy: 0.1094 -
val_loss: 3.1953 - val_precision: 0.3333 - learning_rate: 0.0010
Epoch 3/100
239/239             11s 45ms/step -
accuracy: 0.0939 - loss: 3.3334 - precision: 0.4136 - val_accuracy: 0.1208 -
val_loss: 2.8182 - val_precision: 0.5000 - learning_rate: 0.0010
Epoch 4/100
239/239             11s 45ms/step -
accuracy: 0.1055 - loss: 3.1871 - precision: 0.3918 - val_accuracy: 0.1396 -
val_loss: 3.0495 - val_precision: 0.4000 - learning_rate: 0.0010
Epoch 5/100
239/239             11s 45ms/step -
accuracy: 0.1132 - loss: 3.1201 - precision: 0.4120 - val_accuracy: 0.1887 -
val_loss: 2.5843 - val_precision: 0.4286 - learning_rate: 0.0010
Epoch 6/100
239/239             10s 42ms/step -
accuracy: 0.1372 - loss: 3.0215 - precision: 0.4430 - val_accuracy: 0.1396 -
val_loss: 2.8944 - val_precision: 0.5238 - learning_rate: 0.0010
Epoch 7/100
239/239             11s 45ms/step -
accuracy: 0.1273 - loss: 3.0795 - precision: 0.4368 - val_accuracy: 0.2377 -
val_loss: 2.9743 - val_precision: 0.4603 - learning_rate: 0.0010
Epoch 8/100
239/239             10s 42ms/step -
accuracy: 0.1510 - loss: 2.9061 - precision: 0.4506 - val_accuracy: 0.1585 -
```

```
val_loss: 2.8141 - val_precision: 0.4000 - learning_rate: 0.0010
Epoch 9/100
239/239             10s 42ms/step -
accuracy: 0.1634 - loss: 2.8393 - precision: 0.4914 - val_accuracy: 0.1962 -
val_loss: 2.6810 - val_precision: 0.5405 - learning_rate: 0.0010
Epoch 10/100
239/239             11s 46ms/step -
accuracy: 0.1772 - loss: 2.7841 - precision: 0.5090 - val_accuracy: 0.2792 -
val_loss: 2.0956 - val_precision: 0.5467 - learning_rate: 0.0010
Epoch 11/100
239/239             10s 42ms/step -
accuracy: 0.1869 - loss: 2.7358 - precision: 0.5051 - val_accuracy: 0.1660 -
val_loss: 2.6569 - val_precision: 0.4615 - learning_rate: 0.0010
Epoch 12/100
239/239             11s 45ms/step -
accuracy: 0.2206 - loss: 2.6301 - precision: 0.5941 - val_accuracy: 0.3132 -
val_loss: 2.0894 - val_precision: 0.6765 - learning_rate: 0.0010
Epoch 13/100
239/239             10s 42ms/step -
accuracy: 0.2367 - loss: 2.5463 - precision: 0.6186 - val_accuracy: 0.2302 -
val_loss: 2.7195 - val_precision: 0.5333 - learning_rate: 0.0010
Epoch 14/100
239/239             11s 45ms/step -
accuracy: 0.2611 - loss: 2.4881 - precision: 0.6448 - val_accuracy: 0.3849 -
val_loss: 1.8133 - val_precision: 0.6667 - learning_rate: 0.0010
Epoch 15/100
239/239             11s 45ms/step -
accuracy: 0.2835 - loss: 2.3845 - precision: 0.6635 - val_accuracy: 0.4717 -
val_loss: 1.7582 - val_precision: 0.7797 - learning_rate: 0.0010
Epoch 16/100
239/239             11s 45ms/step -
accuracy: 0.2891 - loss: 2.3498 - precision: 0.6697 - val_accuracy: 0.4906 -
val_loss: 1.5737 - val_precision: 0.7769 - learning_rate: 0.0010
Epoch 17/100
239/239             11s 45ms/step -
accuracy: 0.3174 - loss: 2.2636 - precision: 0.6900 - val_accuracy: 0.5094 -
val_loss: 1.6471 - val_precision: 0.7913 - learning_rate: 0.0010
Epoch 18/100
239/239             11s 45ms/step -
accuracy: 0.3214 - loss: 2.2278 - precision: 0.6970 - val_accuracy: 0.5132 -
val_loss: 1.5613 - val_precision: 0.8257 - learning_rate: 0.0010
Epoch 19/100
239/239             11s 45ms/step -
accuracy: 0.3250 - loss: 2.1873 - precision: 0.7161 - val_accuracy: 0.5472 -
val_loss: 1.4954 - val_precision: 0.8462 - learning_rate: 0.0010
Epoch 20/100
239/239             10s 42ms/step -
accuracy: 0.3409 - loss: 2.1279 - precision: 0.7169 - val_accuracy: 0.5358 -
```

```
val_loss: 1.4599 - val_precision: 0.8607 - learning_rate: 0.0010
Epoch 21/100
239/239             10s 42ms/step -
accuracy: 0.3594 - loss: 2.0692 - precision: 0.7476 - val_accuracy: 0.5396 -
val_loss: 1.5347 - val_precision: 0.8615 - learning_rate: 0.0010
Epoch 22/100
239/239             11s 45ms/step -
accuracy: 0.3760 - loss: 2.0275 - precision: 0.7567 - val_accuracy: 0.5698 -
val_loss: 1.4519 - val_precision: 0.8730 - learning_rate: 0.0010
Epoch 23/100
239/239             11s 45ms/step -
accuracy: 0.3905 - loss: 1.9741 - precision: 0.7726 - val_accuracy: 0.6038 -
val_loss: 1.4607 - val_precision: 0.9028 - learning_rate: 0.0010
Epoch 24/100
239/239             11s 45ms/step -
accuracy: 0.4250 - loss: 1.8592 - precision: 0.7879 - val_accuracy: 0.6377 -
val_loss: 1.4401 - val_precision: 0.9021 - learning_rate: 0.0010
Epoch 25/100
239/239             11s 45ms/step -
accuracy: 0.4530 - loss: 1.8166 - precision: 0.7881 - val_accuracy: 0.6717 -
val_loss: 1.2646 - val_precision: 0.9057 - learning_rate: 0.0010
Epoch 26/100
239/239             11s 45ms/step -
accuracy: 0.4621 - loss: 1.7475 - precision: 0.7812 - val_accuracy: 0.6792 -
val_loss: 1.1365 - val_precision: 0.9438 - learning_rate: 0.0010
Epoch 27/100
239/239             11s 45ms/step -
accuracy: 0.4980 - loss: 1.6909 - precision: 0.8188 - val_accuracy: 0.7057 -
val_loss: 1.1492 - val_precision: 0.9217 - learning_rate: 0.0010
Epoch 28/100
239/239             11s 45ms/step -
accuracy: 0.5000 - loss: 1.6433 - precision: 0.8089 - val_accuracy: 0.7208 -
val_loss: 1.0839 - val_precision: 0.9249 - learning_rate: 0.0010
Epoch 29/100
239/239             11s 45ms/step -
accuracy: 0.5265 - loss: 1.5639 - precision: 0.8115 - val_accuracy: 0.7472 -
val_loss: 1.0938 - val_precision: 0.9202 - learning_rate: 0.0010
Epoch 30/100
239/239             11s 45ms/step -
accuracy: 0.5519 - loss: 1.4686 - precision: 0.8360 - val_accuracy: 0.7547 -
val_loss: 1.0088 - val_precision: 0.9188 - learning_rate: 0.0010
Epoch 31/100
239/239             11s 45ms/step -
accuracy: 0.5758 - loss: 1.3645 - precision: 0.8536 - val_accuracy: 0.7660 -
val_loss: 0.9957 - val_precision: 0.9415 - learning_rate: 0.0010
Epoch 32/100
239/239             10s 42ms/step -
accuracy: 0.5959 - loss: 1.3119 - precision: 0.8478 - val_accuracy: 0.7208 -
```

```
val_loss: 1.0539 - val_precision: 0.8495 - learning_rate: 0.0010
Epoch 33/100
239/239             11s 45ms/step -
accuracy: 0.6064 - loss: 1.2773 - precision: 0.8481 - val_accuracy: 0.7698 -
val_loss: 0.8875 - val_precision: 0.9000 - learning_rate: 0.0010
Epoch 34/100
239/239             11s 45ms/step -
accuracy: 0.6343 - loss: 1.1885 - precision: 0.8511 - val_accuracy: 0.8075 -
val_loss: 0.7834 - val_precision: 0.9124 - learning_rate: 0.0010
Epoch 35/100
239/239             10s 42ms/step -
accuracy: 0.6461 - loss: 1.1455 - precision: 0.8648 - val_accuracy: 0.8075 -
val_loss: 0.7181 - val_precision: 0.9352 - learning_rate: 0.0010
Epoch 36/100
239/239             11s 45ms/step -
accuracy: 0.6647 - loss: 1.0607 - precision: 0.8672 - val_accuracy: 0.8113 -
val_loss: 0.6919 - val_precision: 0.9107 - learning_rate: 0.0010
Epoch 37/100
239/239             10s 42ms/step -
accuracy: 0.6786 - loss: 1.0318 - precision: 0.8743 - val_accuracy: 0.8075 -
val_loss: 0.7489 - val_precision: 0.9009 - learning_rate: 0.0010
Epoch 38/100
239/239             11s 45ms/step -
accuracy: 0.6807 - loss: 1.0254 - precision: 0.8606 - val_accuracy: 0.8491 -
val_loss: 0.6054 - val_precision: 0.9295 - learning_rate: 0.0010
Epoch 39/100
239/239             10s 42ms/step -
accuracy: 0.7163 - loss: 0.9249 - precision: 0.8773 - val_accuracy: 0.8038 -
val_loss: 0.7992 - val_precision: 0.8996 - learning_rate: 0.0010
Epoch 40/100
239/239             10s 42ms/step -
accuracy: 0.7165 - loss: 0.9044 - precision: 0.8911 - val_accuracy: 0.8302 -
val_loss: 0.6728 - val_precision: 0.9138 - learning_rate: 0.0010
Epoch 41/100
239/239             10s 42ms/step -
accuracy: 0.7361 - loss: 0.8544 - precision: 0.8825 - val_accuracy: 0.7925 -
val_loss: 0.7350 - val_precision: 0.9061 - learning_rate: 0.0010
Epoch 42/100
239/239             10s 42ms/step -
accuracy: 0.7596 - loss: 0.7593 - precision: 0.8936 - val_accuracy: 0.8340 -
val_loss: 0.7993 - val_precision: 0.8857 - learning_rate: 0.0010
Epoch 43/100
239/239             0s 41ms/step -
accuracy: 0.7717 - loss: 0.7411 - precision: 0.8935
Epoch 43: ReduceLROnPlateau reducing learning rate to 0.00020000000949949026.
239/239             11s 45ms/step -
accuracy: 0.7717 - loss: 0.7410 - precision: 0.8935 - val_accuracy: 0.8604 -
val_loss: 0.6229 - val_precision: 0.9397 - learning_rate: 0.0010
```

```
Epoch 44/100
239/239             11s 45ms/step -
accuracy: 0.7859 - loss: 0.6696 - precision: 0.9038 - val_accuracy: 0.8830 -
val_loss: 0.5907 - val_precision: 0.9306 - learning_rate: 2.0000e-04
Epoch 45/100
239/239             10s 42ms/step -
accuracy: 0.8132 - loss: 0.5676 - precision: 0.9208 - val_accuracy: 0.8755 -
val_loss: 0.5887 - val_precision: 0.9184 - learning_rate: 2.0000e-04
Epoch 46/100
239/239             10s 42ms/step -
accuracy: 0.8295 - loss: 0.5174 - precision: 0.9305 - val_accuracy: 0.8830 -
val_loss: 0.5699 - val_precision: 0.9312 - learning_rate: 2.0000e-04
Epoch 47/100
239/239             11s 45ms/step -
accuracy: 0.8312 - loss: 0.5150 - precision: 0.9276 - val_accuracy: 0.8868 -
val_loss: 0.5979 - val_precision: 0.9163 - learning_rate: 2.0000e-04
Epoch 48/100
239/239             11s 45ms/step -
accuracy: 0.8394 - loss: 0.4768 - precision: 0.9310 - val_accuracy: 0.8981 -
val_loss: 0.5562 - val_precision: 0.9323 - learning_rate: 2.0000e-04
Epoch 49/100
239/239             10s 42ms/step -
accuracy: 0.8591 - loss: 0.4387 - precision: 0.9352 - val_accuracy: 0.8981 -
val_loss: 0.5537 - val_precision: 0.9252 - learning_rate: 2.0000e-04
Epoch 50/100
239/239             10s 42ms/step -
accuracy: 0.8597 - loss: 0.4206 - precision: 0.9418 - val_accuracy: 0.8868 -
val_loss: 0.5501 - val_precision: 0.9203 - learning_rate: 2.0000e-04
Epoch 51/100
239/239             11s 45ms/step -
accuracy: 0.8622 - loss: 0.3933 - precision: 0.9389 - val_accuracy: 0.9094 -
val_loss: 0.5107 - val_precision: 0.9402 - learning_rate: 2.0000e-04
Epoch 52/100
239/239             10s 42ms/step -
accuracy: 0.8633 - loss: 0.4060 - precision: 0.9351 - val_accuracy: 0.9057 -
val_loss: 0.4888 - val_precision: 0.9400 - learning_rate: 2.0000e-04
Epoch 53/100
239/239             10s 42ms/step -
accuracy: 0.8664 - loss: 0.3824 - precision: 0.9394 - val_accuracy: 0.9057 -
val_loss: 0.5395 - val_precision: 0.9261 - learning_rate: 2.0000e-04
Epoch 54/100
239/239             10s 42ms/step -
accuracy: 0.8839 - loss: 0.3462 - precision: 0.9456 - val_accuracy: 0.9019 -
val_loss: 0.5189 - val_precision: 0.9325 - learning_rate: 2.0000e-04
Epoch 55/100
239/239             11s 45ms/step -
accuracy: 0.8953 - loss: 0.3254 - precision: 0.9522 - val_accuracy: 0.9132 -
val_loss: 0.5042 - val_precision: 0.9449 - learning_rate: 2.0000e-04
```

```
Epoch 56/100
239/239             10s 42ms/step -
accuracy: 0.8961 - loss: 0.3186 - precision: 0.9472 - val_accuracy: 0.8981 -
val_loss: 0.5166 - val_precision: 0.9255 - learning_rate: 2.0000e-04
Epoch 57/100
239/239             0s 41ms/step -
accuracy: 0.9018 - loss: 0.3052 - precision: 0.9500
Epoch 57: ReduceLROnPlateau reducing learning rate to 4.0000001899898055e-05.
239/239             10s 42ms/step -
accuracy: 0.9018 - loss: 0.3052 - precision: 0.9500 - val_accuracy: 0.9019 -
val_loss: 0.5429 - val_precision: 0.9370 - learning_rate: 2.0000e-04
Epoch 58/100
239/239             10s 42ms/step -
accuracy: 0.9082 - loss: 0.2796 - precision: 0.9582 - val_accuracy: 0.9019 -
val_loss: 0.5380 - val_precision: 0.9370 - learning_rate: 4.0000e-05
Epoch 59/100
239/239             10s 42ms/step -
accuracy: 0.9058 - loss: 0.2694 - precision: 0.9513 - val_accuracy: 0.9094 -
val_loss: 0.5341 - val_precision: 0.9333 - learning_rate: 4.0000e-05
Epoch 60/100
239/239             10s 42ms/step -
accuracy: 0.9175 - loss: 0.2434 - precision: 0.9557 - val_accuracy: 0.9094 -
val_loss: 0.5369 - val_precision: 0.9297 - learning_rate: 4.0000e-05
Epoch 61/100
239/239             11s 45ms/step -
accuracy: 0.9194 - loss: 0.2472 - precision: 0.9600 - val_accuracy: 0.9170 -
val_loss: 0.5313 - val_precision: 0.9370 - learning_rate: 4.0000e-05
Epoch 62/100
239/239             0s 41ms/step -
accuracy: 0.9112 - loss: 0.2538 - precision: 0.9542
Epoch 62: ReduceLROnPlateau reducing learning rate to 8.000000525498762e-06.
239/239             10s 42ms/step -
accuracy: 0.9113 - loss: 0.2538 - precision: 0.9542 - val_accuracy: 0.9170 -
val_loss: 0.5355 - val_precision: 0.9409 - learning_rate: 4.0000e-05
Epoch 63/100
239/239             10s 42ms/step -
accuracy: 0.9215 - loss: 0.2376 - precision: 0.9610 - val_accuracy: 0.9170 -
val_loss: 0.5327 - val_precision: 0.9409 - learning_rate: 8.0000e-06
Epoch 64/100
239/239             10s 42ms/step -
accuracy: 0.9233 - loss: 0.2302 - precision: 0.9631 - val_accuracy: 0.9170 -
val_loss: 0.5329 - val_precision: 0.9409 - learning_rate: 8.0000e-06
Epoch 65/100
239/239             10s 42ms/step -
accuracy: 0.9261 - loss: 0.2334 - precision: 0.9615 - val_accuracy: 0.9170 -
val_loss: 0.5305 - val_precision: 0.9409 - learning_rate: 8.0000e-06
Epoch 66/100
239/239             10s 42ms/step -
```

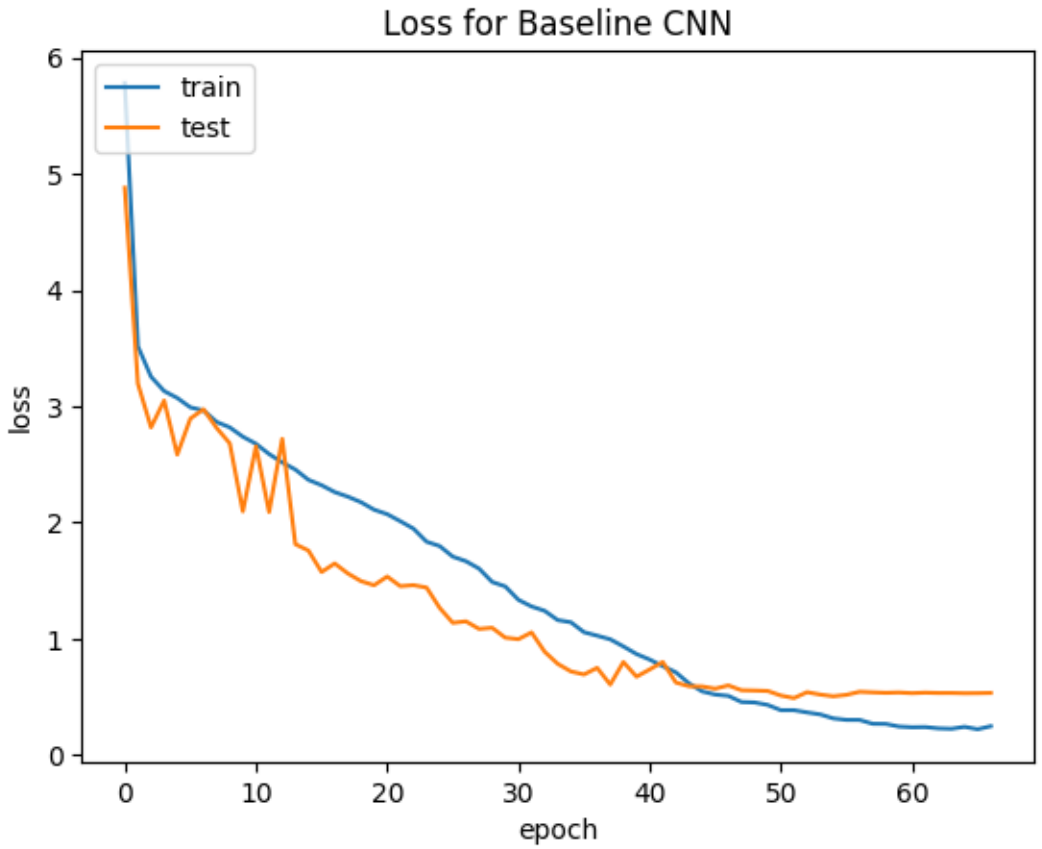
```
accuracy: 0.9255 - loss: 0.2310 - precision: 0.9606 - val_accuracy: 0.9170 -
val_loss: 0.5313 - val_precision: 0.9412 - learning_rate: 8.0000e-06
Epoch 67/100
239/239             0s 41ms/step -
accuracy: 0.9227 - loss: 0.2408 - precision: 0.9600
Epoch 67: ReduceLROnPlateau reducing learning rate to 1.6000001778593287e-06.
239/239             10s 42ms/step -
accuracy: 0.9227 - loss: 0.2408 - precision: 0.9600 - val_accuracy: 0.9170 -
val_loss: 0.5333 - val_precision: 0.9412 - learning_rate: 8.0000e-06
```

[24]:
```
end1=time.time()
```

6.1.3 Visualising baseline model's training and validation performance :

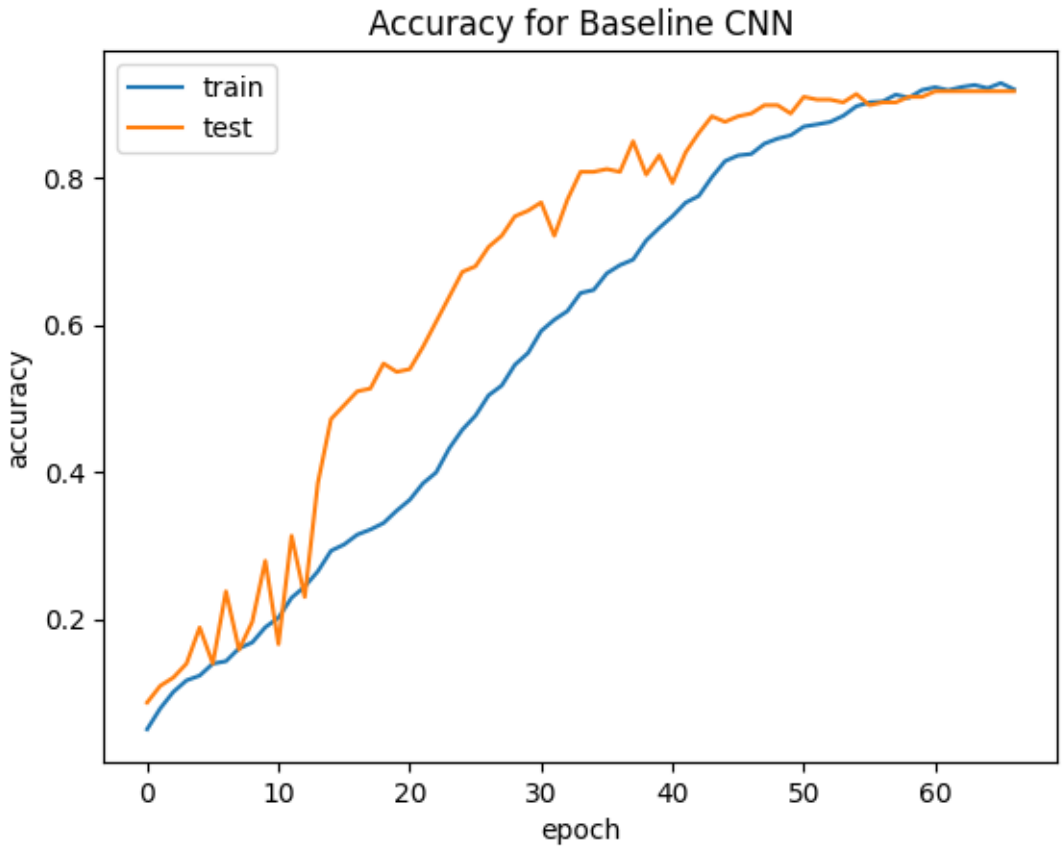
[25]:
```
plt.plot(history1.history['loss'])
plt.plot(history1.history['val_loss'])
plt.title('Loss for Baseline CNN')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
```

[25]:
```
<matplotlib.legend.Legend at 0x79500022bd90>
```



[26]:
```
plt.plot(history1.history['accuracy'])
plt.plot(history1.history['val_accuracy'])
plt.title('Accuracy for Baseline CNN')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
```

[26]:
```
<matplotlib.legend.Legend at 0x795024218520>
```



6.1.4 Evaluating model on testing data and saving its metrics :

[27]:
```
m1_eval=model1.evaluate(test_dataset)
```

```
9/9             0s 13ms/step -
accuracy: 0.9117 - loss: 0.4769 - precision: 0.9382
```

[28]:
```
print(m1_eval)
```

```
[0.6416582465171814, 0.8792452812194824, 0.9169960618019104]
```

[29]:
```
df.loc[len(df)] = ['Baseline_Model',m1_eval[1],m1_eval[2],(end1-start1)]
```

6.2 Building Model 2 : With EfficientNetB1

This model has:

1) Augmentation layer
2) Base layers of EfficientNetB1
3) One dense layer

Downloading the pretrained base :

[31]:
```
efnetb1 = tf.keras.applications.
 ↪EfficientNetB1(weights='imagenet',include_top=False,input_shape=(img_height,␣
 ↪img_width, 3))
efnetb1.training = False
```

```
Downloading data from https://storage.googleapis.com/keras-
applications/efficientnetb1_notop.h5
27018416/27018416              0s
0us/step
```

6.2.1 Building the model :

[32]:
```
model2 = tf.keras.Sequential([
    tf.keras.layers.Input(shape=(img_height,img_width) + (3, ),␣
 ↪name="input_layer"),
    augmentation_layer,
    efnetb1,
    tf.keras.layers.GlobalAveragePooling2D(name="global_average_pooling_layer"),
    tf.keras.layers.Dense(256, activation=tf.keras.activations.relu),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Dense(len(class_names), activation=tf.keras.activations.
 ↪softmax)
])
```

6.2.2 Compiling the model and checking its structure :

[33]:
```
model2.compile(optimizer='Adam',
               loss=tf.keras.losses.CategoricalCrossentropy(from_logits=True),
               metrics=['accuracy','precision'])
```

[34]:
```
model2.summary()
```

```
Model: "sequential_1"
```

Layer (type)	Output Shape	Param #

augmentation_layer (Sequential)	(None, None, None, 3)	0
efficientnetb1 (Functional)	(None, None, None, 1280)	6,575,239
global_average_pooling_layer (GlobalAveragePooling2D)	(None, 1280)	0
dense_2 (Dense)	(None, 256)	327,936
batch_normalization_4 (BatchNormalization)	(None, 256)	1,024
dense_3 (Dense)	(None, 53)	13,621

Total params: 6,917,820 (26.39 MB)

Trainable params: 6,855,253 (26.15 MB)

Non-trainable params: 62,567 (244.41 KB)

6.2.3 Fitting the model for 100 epochs :

[35]:
```
model2_checkpoint = tf.keras.callbacks.ModelCheckpoint('best_efnetb0_model.
↪keras', monitor='val_accuracy', save_best_only=True, mode='max')
```

[36]:
```
start2=time.time()
history2 = model2.fit(
  train_dataset,
  validation_data=valid_dataset,
 ␣
↪epochs=100,verbose=1,callbacks=[early_stopping_callback,model2_checkpoint,reduce_lr_callbac
```

```
Epoch 1/100
/opt/conda/lib/python3.10/site-packages/keras/src/backend/tensorflow/nn.py:560:
UserWarning: "`categorical_crossentropy` received `from_logits=True`, but the
`output` argument was produced by a Softmax activation and thus does not
represent logits. Was this intended?
  output, from_logits = _get_logits(
2024-06-30 10:13:48.177804: E
tensorflow/core/grappler/optimizers/meta_optimizer.cc:961] layout failed:
INVALID_ARGUMENT: Size of values 0 does not match size of permutation 4 @ fanin
shape inStatefulPartitionedCall/sequential_1_1/efficientnetb1_1/block1b_drop_1/s
tateless_dropout/SelectV2-2-TransposeNHWCToNCHW-LayoutOptimizer
```

```
239/239             154s 405ms/step -
accuracy: 0.2776 - loss: 2.8406 - precision: 0.4986 - val_accuracy: 0.0189 -
val_loss: 4.5471 - val_precision: 0.0000e+00 - learning_rate: 0.0010
Epoch 2/100
239/239             88s 366ms/step -
accuracy: 0.6808 - loss: 1.1365 - precision: 0.8443 - val_accuracy: 0.5962 -
val_loss: 1.4252 - val_precision: 0.8170 - learning_rate: 0.0010
Epoch 3/100
239/239             87s 363ms/step -
accuracy: 0.7623 - loss: 0.8715 - precision: 0.8837 - val_accuracy: 0.8679 -
val_loss: 0.4572 - val_precision: 0.9237 - learning_rate: 0.0010
Epoch 4/100
239/239             87s 365ms/step -
accuracy: 0.7953 - loss: 0.7337 - precision: 0.8995 - val_accuracy: 0.9208 -
val_loss: 0.3229 - val_precision: 0.9365 - learning_rate: 0.0010
Epoch 5/100
239/239             85s 354ms/step -
accuracy: 0.8207 - loss: 0.6135 - precision: 0.9146 - val_accuracy: 0.8755 -
val_loss: 0.3852 - val_precision: 0.9076 - learning_rate: 0.0010
Epoch 6/100
239/239             84s 352ms/step -
accuracy: 0.8352 - loss: 0.5799 - precision: 0.9243 - val_accuracy: 0.8981 -
val_loss: 0.3583 - val_precision: 0.9431 - learning_rate: 0.0010
Epoch 7/100
239/239             86s 359ms/step -
accuracy: 0.8585 - loss: 0.4859 - precision: 0.9252 - val_accuracy: 0.8604 -
val_loss: 0.4915 - val_precision: 0.8911 - learning_rate: 0.0010
Epoch 8/100
239/239             86s 361ms/step -
accuracy: 0.8657 - loss: 0.4607 - precision: 0.9265 - val_accuracy: 0.8830 -
val_loss: 0.4135 - val_precision: 0.9059 - learning_rate: 0.0010
Epoch 9/100
239/239             87s 365ms/step -
accuracy: 0.8791 - loss: 0.4487 - precision: 0.9288 - val_accuracy: 0.9283 -
val_loss: 0.2082 - val_precision: 0.9453 - learning_rate: 0.0010
Epoch 10/100
239/239             85s 355ms/step -
accuracy: 0.8957 - loss: 0.3632 - precision: 0.9413 - val_accuracy: 0.9170 -
val_loss: 0.2837 - val_precision: 0.9336 - learning_rate: 0.0010
Epoch 11/100
239/239             85s 357ms/step -
accuracy: 0.8867 - loss: 0.3794 - precision: 0.9306 - val_accuracy: 0.9208 -
val_loss: 0.2371 - val_precision: 0.9407 - learning_rate: 0.0010
Epoch 12/100
239/239             85s 355ms/step -
accuracy: 0.9064 - loss: 0.3083 - precision: 0.9424 - val_accuracy: 0.9170 -
val_loss: 0.3308 - val_precision: 0.9269 - learning_rate: 0.0010
Epoch 13/100
```

```
239/239             85s 355ms/step -
accuracy: 0.9103 - loss: 0.3138 - precision: 0.9462 - val_accuracy: 0.9019 -
val_loss: 0.4038 - val_precision: 0.9186 - learning_rate: 0.0010
Epoch 14/100
239/239             0s 347ms/step -
accuracy: 0.9076 - loss: 0.3287 - precision: 0.9438
Epoch 14: ReduceLROnPlateau reducing learning rate to 0.00020000000949949026.
239/239             84s 350ms/step -
accuracy: 0.9076 - loss: 0.3286 - precision: 0.9438 - val_accuracy: 0.9170 -
val_loss: 0.2908 - val_precision: 0.9300 - learning_rate: 0.0010
Epoch 15/100
239/239             86s 360ms/step -
accuracy: 0.9292 - loss: 0.2285 - precision: 0.9535 - val_accuracy: 0.9774 -
val_loss: 0.0842 - val_precision: 0.9808 - learning_rate: 2.0000e-04
Epoch 16/100
239/239             85s 354ms/step -
accuracy: 0.9631 - loss: 0.1202 - precision: 0.9779 - val_accuracy: 0.9698 -
val_loss: 0.0735 - val_precision: 0.9770 - learning_rate: 2.0000e-04
Epoch 17/100
239/239             84s 353ms/step -
accuracy: 0.9733 - loss: 0.0914 - precision: 0.9829 - val_accuracy: 0.9736 -
val_loss: 0.0719 - val_precision: 0.9734 - learning_rate: 2.0000e-04
Epoch 18/100
239/239             142s 352ms/step -
accuracy: 0.9754 - loss: 0.0792 - precision: 0.9848 - val_accuracy: 0.9736 -
val_loss: 0.0719 - val_precision: 0.9771 - learning_rate: 2.0000e-04
Epoch 19/100
239/239             85s 354ms/step -
accuracy: 0.9820 - loss: 0.0661 - precision: 0.9900 - val_accuracy: 0.9736 -
val_loss: 0.0703 - val_precision: 0.9809 - learning_rate: 2.0000e-04
Epoch 20/100
239/239             143s 358ms/step -
accuracy: 0.9862 - loss: 0.0522 - precision: 0.9899 - val_accuracy: 0.9774 -
val_loss: 0.0562 - val_precision: 0.9773 - learning_rate: 2.0000e-04
Epoch 21/100
239/239             142s 358ms/step -
accuracy: 0.9863 - loss: 0.0499 - precision: 0.9899 - val_accuracy: 0.9774 -
val_loss: 0.0807 - val_precision: 0.9774 - learning_rate: 2.0000e-04
Epoch 22/100
239/239             85s 354ms/step -
accuracy: 0.9845 - loss: 0.0484 - precision: 0.9896 - val_accuracy: 0.9736 -
val_loss: 0.0893 - val_precision: 0.9773 - learning_rate: 2.0000e-04
Epoch 23/100
239/239             84s 353ms/step -
accuracy: 0.9875 - loss: 0.0388 - precision: 0.9897 - val_accuracy: 0.9736 -
val_loss: 0.0834 - val_precision: 0.9736 - learning_rate: 2.0000e-04
Epoch 24/100
239/239             84s 349ms/step -
```

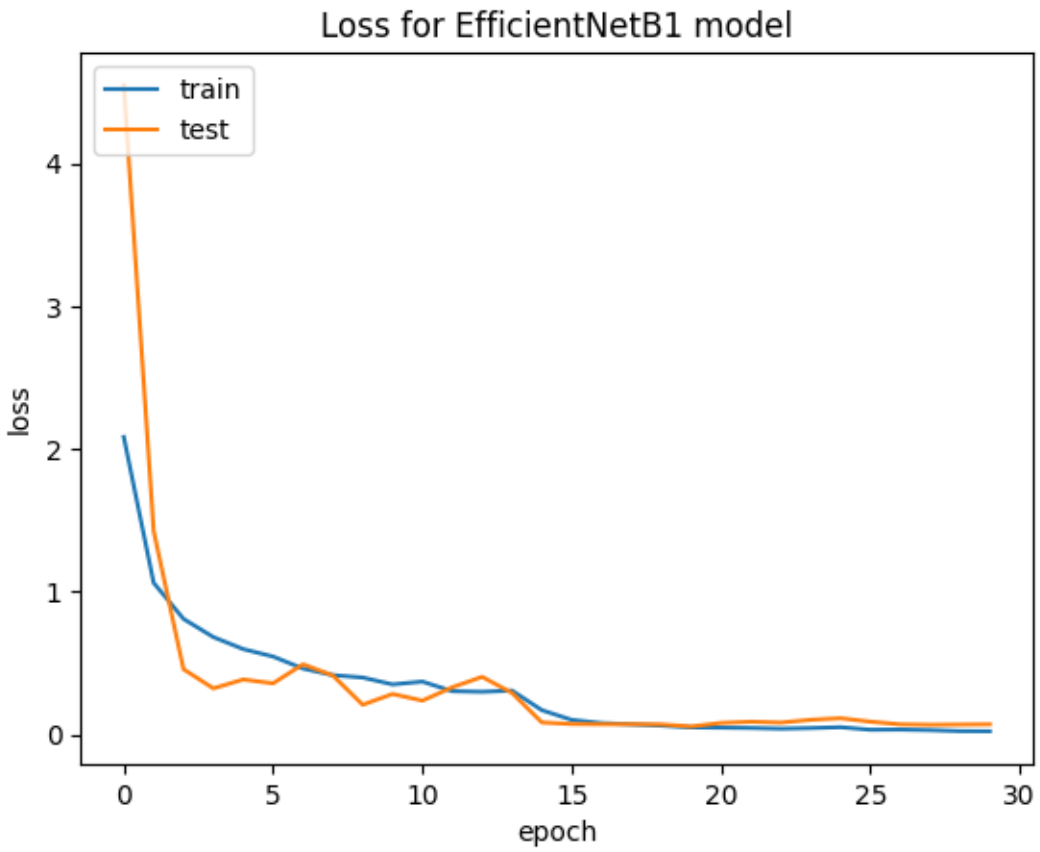
```
accuracy: 0.9844 - loss: 0.0504 - precision: 0.9873 - val_accuracy: 0.9774 -
val_loss: 0.1030 - val_precision: 0.9773 - learning_rate: 2.0000e-04
Epoch 25/100
239/239             0s 352ms/step -
accuracy: 0.9797 - loss: 0.0608 - precision: 0.9838
Epoch 25: ReduceLROnPlateau reducing learning rate to 4.0000001899898055e-05.
239/239             85s 355ms/step -
accuracy: 0.9797 - loss: 0.0608 - precision: 0.9838 - val_accuracy: 0.9623 -
val_loss: 0.1139 - val_precision: 0.9659 - learning_rate: 2.0000e-04
Epoch 26/100
239/239             85s 357ms/step -
accuracy: 0.9899 - loss: 0.0347 - precision: 0.9918 - val_accuracy: 0.9660 -
val_loss: 0.0896 - val_precision: 0.9659 - learning_rate: 4.0000e-05
Epoch 27/100
239/239             85s 357ms/step -
accuracy: 0.9849 - loss: 0.0463 - precision: 0.9877 - val_accuracy: 0.9736 -
val_loss: 0.0711 - val_precision: 0.9773 - learning_rate: 4.0000e-05
Epoch 28/100
239/239             86s 359ms/step -
accuracy: 0.9880 - loss: 0.0349 - precision: 0.9899 - val_accuracy: 0.9736 -
val_loss: 0.0681 - val_precision: 0.9736 - learning_rate: 4.0000e-05
Epoch 29/100
239/239             87s 362ms/step -
accuracy: 0.9920 - loss: 0.0248 - precision: 0.9939 - val_accuracy: 0.9698 -
val_loss: 0.0697 - val_precision: 0.9698 - learning_rate: 4.0000e-05
Epoch 30/100
239/239             0s 358ms/step -
accuracy: 0.9936 - loss: 0.0245 - precision: 0.9955
Epoch 30: ReduceLROnPlateau reducing learning rate to 8.000000525498762e-06.
239/239             86s 360ms/step -
accuracy: 0.9936 - loss: 0.0245 - precision: 0.9955 - val_accuracy: 0.9660 -
val_loss: 0.0714 - val_precision: 0.9660 - learning_rate: 4.0000e-05
```

[37]:
```
end2=time.time()
```

6.2.4 Visualising model performance :

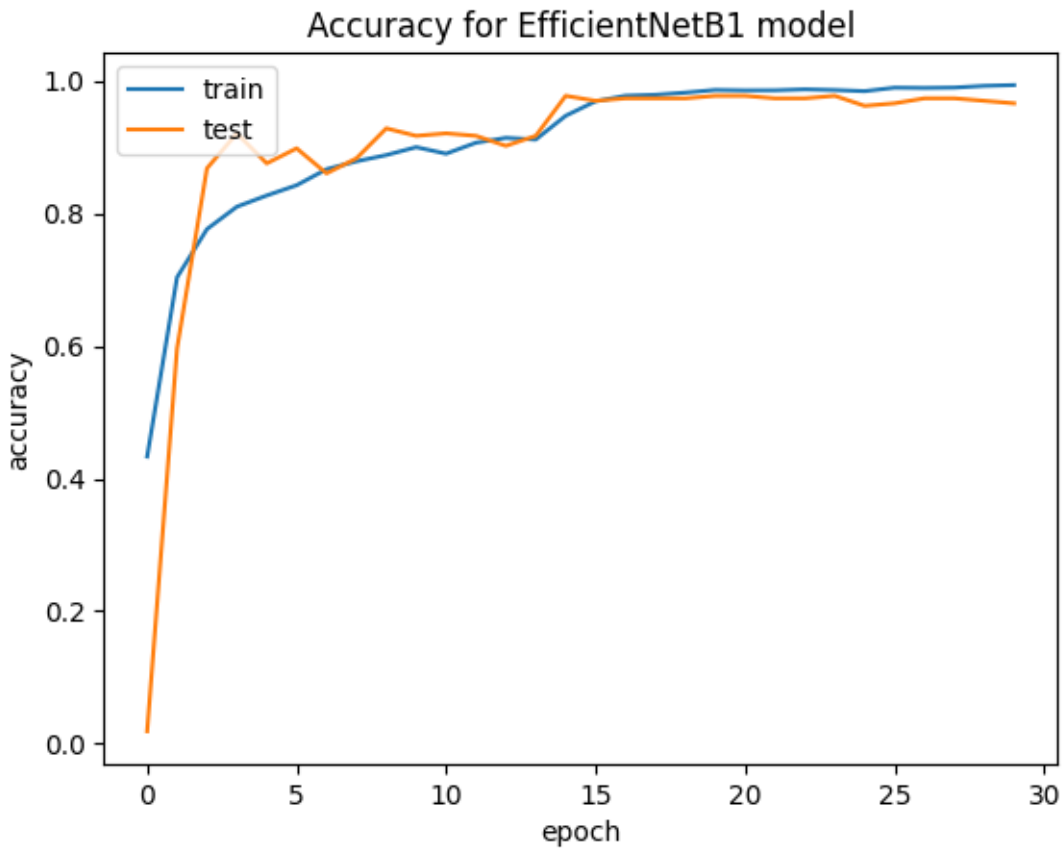
[38]:
```
plt.plot(history2.history['loss'])
plt.plot(history2.history['val_loss'])
plt.title('Loss for EfficientNetB1 model')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
```

[38]:
```
<matplotlib.legend.Legend at 0x794b81c53be0>
```



[39]:
```
plt.plot(history2.history['accuracy'])
plt.plot(history2.history['val_accuracy'])
plt.title('Accuracy for EfficientNetB1 model')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
```

[39]: <matplotlib.legend.Legend at 0x794b334e98a0>



The EfficientNetB1 transfer learning model visibly converged to higher validation accuracy values faster than the baseline model, crossing 90% within the 10 epochs, even though individual epochs took longer to train.

6.2.5 Evaluating model on the testing data and saving its performance metrics:

[40]:
```
m2_eval=model2.evaluate(test_dataset)
```

```
9/9             1s 61ms/step -
accuracy: 0.9606 - loss: 0.1418 - precision: 0.9720
```

[41]:
```
df.loc[len(df)] = ['EfficientNetB1',m2_eval[1],m2_eval[2],(end2-start2)]
```

---

7. Comparing performance of both models :

[42]:
```
print(df)
```

	Model_Name	Accuracy	Precision	Training_time(s)
0	Baseline_Model	0.879245	0.916996	747.207493
1	EfficientNetB1	0.943396	0.957854	2800.077686

Thank You !

If you found this article helpful, please do leave a like and comment any feedback you feel I could use.

While I’m working on the next project, I’ll be over at :

www.linkedin.com/in/shivang-kainthola-2835151b9/

shivang.kainthola64@gmail.com

https://shivangkainthola28.medium.com/

https://github.com/HeadHunter28