# Loading the dataset using Kaggle API

```
!pip install kaggle
```

```
Requirement already satisfied: kaggle in /usr/local/lib/python3.10/dist-packages (1.5.16)
Requirement already satisfied: six>=1.10 in /usr/local/lib/python3.10/dist-packages (from kaggle) (1.16.0)
Requirement already satisfied: certifi in /usr/local/lib/python3.10/dist-packages (from kaggle) (2023.7.22)
Requirement already satisfied: python-dateutil in /usr/local/lib/python3.10/dist-packages (from kaggle) (2.8.2)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from kaggle) (2.31.0)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from kaggle) (4.66.1)
Requirement already satisfied: python-slugify in /usr/local/lib/python3.10/dist-packages (from kaggle) (8.0.1)
Requirement already satisfied: urllib3 in /usr/local/lib/python3.10/dist-packages (from kaggle) (2.0.7)
Requirement already satisfied: bleach in /usr/local/lib/python3.10/dist-packages (from kaggle) (6.1.0)
Requirement already satisfied: webencodings in /usr/local/lib/python3.10/dist-packages (from bleach->kaggle) (0.5.1)
Requirement already satisfied: text-unidecode>=1.3 in /usr/local/lib/python3.10/dist-packages (from python-slugify->kag
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests->kagg
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests->kaggle) (3.4)
```

```
# ! mkdir ~/.kaggle
```

```
mkdir: cannot create directory ‘/root/.kaggle’: File exists
```

```
# ! cp kaggle.json ~/.kaggle/
# ! chmod 600 ~/kaggle/kaggle.json
```

```
chmod: cannot access '/root/kaggle/kaggle.json': No such file or directory
```

```
from google.colab import files
files.upload()
```

```
Choose Files No file chosen  Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving kaggle.json to kaggle.json
{'kaggle.json': b'{"username":"shauryadhoundiyal","key":"79ac760dd8639c19ed9678f7a4553226"}'}
```

```
! chmod 600 ~/.kaggle/kaggle.json
```

```
! kaggle datasets list
```

```
ref                                                            title                                                size
-------------------------------------------------------------  ---------------------------------------------------  -----
carlmcbrideellis/llm-7-prompt-training-dataset                 LLM: 7 prompt training dataset                        41MB
thedrcat/daigt-proper-train-dataset                            DAIGT Proper Train Dataset                           119MB
iamsouravbanerjee/customer-shopping-trends-dataset             Customer Shopping Trends Dataset                     146KB
joebeachcapital/30000-spotify-songs                            30000 Spotify Songs                                    3MB
thedrcat/daigt-v2-train-dataset                                DAIGT V2 Train Dataset                                29MB
prasad22/healthcare-dataset                                    🩺Healthcare Dataset 💉                                 4B
mauryansshivam/list-of-internet-products-of-top-tech-companies List of Internet Products of Top Tech Companies      9KB
dillonmyrick/high-school-student-performance-and-demographics  High School Student Performance & Demographics       24KB
nelgiriyewithana/world-educational-data                        World Educational Data                                9KB
alejopaullier/daigt-external-dataset                           DAIGT | External Dataset                              3MB
joebeachcapital/coronavirus-covid-19-cases-daily-updates       Coronavirus (COVID-19) Cases (Daily Updates)         14MB
jacksondivakarr/online-shopping-dataset                        🛒 Online Shopping Dataset 📊📉📈
anshtanwar/top-200-trending-books-with-reviews                 Top 100 Bestselling Book Reviews on Amazon           422KB
stefancomanita/top-us-songs-from-1950-to-2019-w-lyrics         Top US Songs from 1950 to 2019, w. lyrics            674KB
asaniczka/ufc-fighters-statistics                              UFC Fighters' Statistics Dataset                     148KB
zeesolver/consumer-behavior-and-shopping-habits-dataset        Consumer Behavior and Shopping Habits Dataset:       146KB
ddosad/auto-sales-data                                         Automobile Sales data                                 79KB
jdaustralia/icc-cwc23-all-innings-cleaned                      ICC Cricket World Cup CWC23 All innings               28KB
vikramrn/icc-mens-odi-world-cup-wc-2023                        ICC mens cricket odi world cup wc 2023 - batting     16KB
simonezappatini/body-fat-extended-dataset                      Body Fat Extended Dataset                             12KB
```

```
!kaggle datasets download -d snginh/teethdecay
```

```
teethdecay.zip: Skipping, found more recently modified local copy (use --force to force download)
```

```
!unzip /content/teethdecay.zip
```

```
  inflating: teeth_dataset/train/caries/wc41_8.jpg
  inflating: teeth_dataset/train/caries/wc41_9.jpg
  inflating: teeth_dataset/train/caries/wc42.jpg
  inflating: teeth_dataset/train/caries/wc42_0.jpg
  inflating: teeth_dataset/train/caries/wc42_1.jpg
  inflating: teeth_dataset/train/caries/wc42_10.jpg
  inflating: teeth_dataset/train/caries/wc42_11.jpg
  inflating: teeth_dataset/train/caries/wc42_12.jpg
  inflating: teeth_dataset/train/caries/wc42_13.jpg
  inflating: teeth_dataset/train/caries/wc42_14.jpg
  inflating: teeth_dataset/train/caries/wc42_15.jpg
  inflating: teeth_dataset/train/caries/wc42_16.jpg
  inflating: teeth_dataset/train/caries/wc42_17.jpg
  inflating: teeth_dataset/train/caries/wc42_18.jpg
  inflating: teeth_dataset/train/caries/wc42_19.jpg
  inflating: teeth_dataset/train/caries/wc42_2.jpg
  inflating: teeth_dataset/train/caries/wc42_3.jpg
  inflating: teeth_dataset/train/caries/wc42_4.jpg
  inflating: teeth_dataset/train/caries/wc42_5.jpg
  inflating: teeth_dataset/train/caries/wc42_6.jpg
  inflating: teeth_dataset/train/caries/wc42_7.jpg
  inflating: teeth_dataset/train/caries/wc42_8.jpg
  inflating: teeth_dataset/train/caries/wc42_9.jpg
  inflating: teeth_dataset/train/caries/wc43.jpg
  inflating: teeth_dataset/train/caries/wc43_0.jpg
  inflating: teeth_dataset/train/caries/wc43_1.jpg
  inflating: teeth_dataset/train/caries/wc43_10.jpg
  inflating: teeth_dataset/train/caries/wc43_11.jpg
  inflating: teeth_dataset/train/caries/wc43_12.jpg
  inflating: teeth_dataset/train/caries/wc43_13.jpg
  inflating: teeth_dataset/train/caries/wc43_14.jpg
  inflating: teeth_dataset/train/caries/wc43_15.jpg
  inflating: teeth_dataset/train/caries/wc43_16.jpg
  inflating: teeth_dataset/train/caries/wc43_17.jpg
  inflating: teeth_dataset/train/caries/wc43_18.jpg
  inflating: teeth_dataset/train/caries/wc43_19.jpg
  inflating: teeth_dataset/train/caries/wc43_2.jpg
  inflating: teeth_dataset/train/caries/wc43_3.jpg
  inflating: teeth_dataset/train/caries/wc43_4.jpg
  inflating: teeth_dataset/train/caries/wc43_5.jpg
  inflating: teeth_dataset/train/caries/wc43_6.jpg
  inflating: teeth_dataset/train/caries/wc43_7.jpg
  inflating: teeth_dataset/train/caries/wc43_8.jpg
  inflating: teeth_dataset/train/caries/wc43_9.jpg
  inflating: teeth_dataset/train/caries/wc47.jpg
  inflating: teeth_dataset/train/caries/wc47_0.jpg
  inflating: teeth_dataset/train/caries/wc47_1.jpg
  inflating: teeth_dataset/train/caries/wc47_10.jpg
  inflating: teeth_dataset/train/caries/wc47_11.jpg
  inflating: teeth_dataset/train/caries/wc47_12.jpg
  inflating: teeth_dataset/train/caries/wc47_13.jpg
```

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from tqdm import tqdm
import cv2

import warnings
warnings.filterwarnings('ignore')
import os
train_path = "/content/teeth_dataset/train/"
test_path = "/content/teeth_dataset/test"
```

```
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Activation, Dropout, Flatten, Dense, BatchNormalization
from keras.preprocessing.image import ImageDataGenerator
from keras.utils import img_to_array, load_img
from keras.utils import plot_model
from glob import glob
```

## Getting contents of training data

```
x_data = []
y_data = []

for category in glob(train_path+'/*'):
    for file in tqdm(glob(category+'/*')):
        img_array=cv2.imread(file)
        img_array = cv2.cvtColor(img_array, cv2.COLOR_BGR2RGB)
        x_data.append(img_array)
        y_data.append(category.split("/")[-1])

data=pd.DataFrame({'image': x_data,'label': y_data})
```

```
100%|██████████| 945/945 [00:00<00:00, 1429.97it/s]
100%|██████████| 315/315 [00:00<00:00, 1382.21it/s]
```

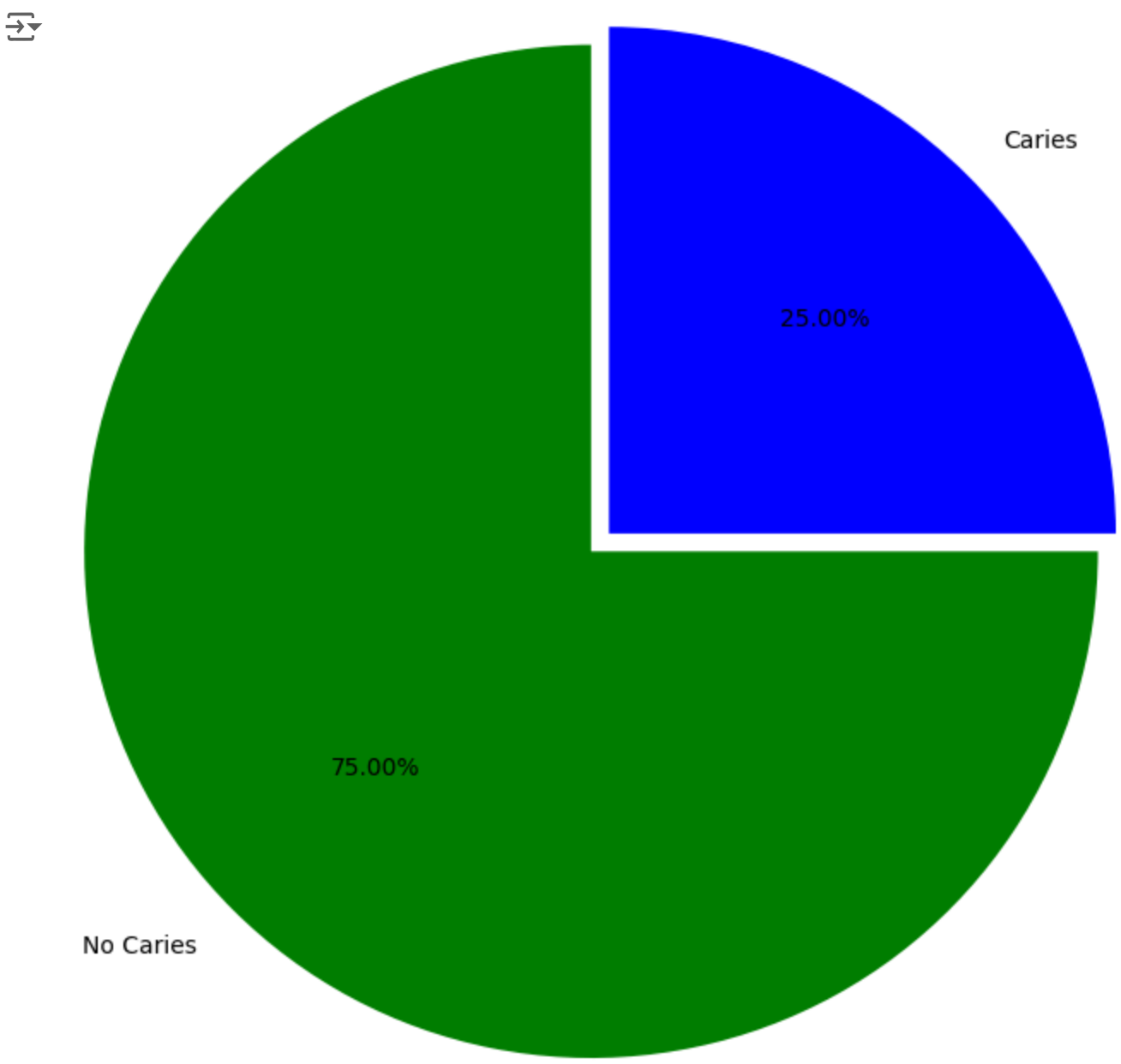
```
data.shape
```

```
(1260, 2)
```

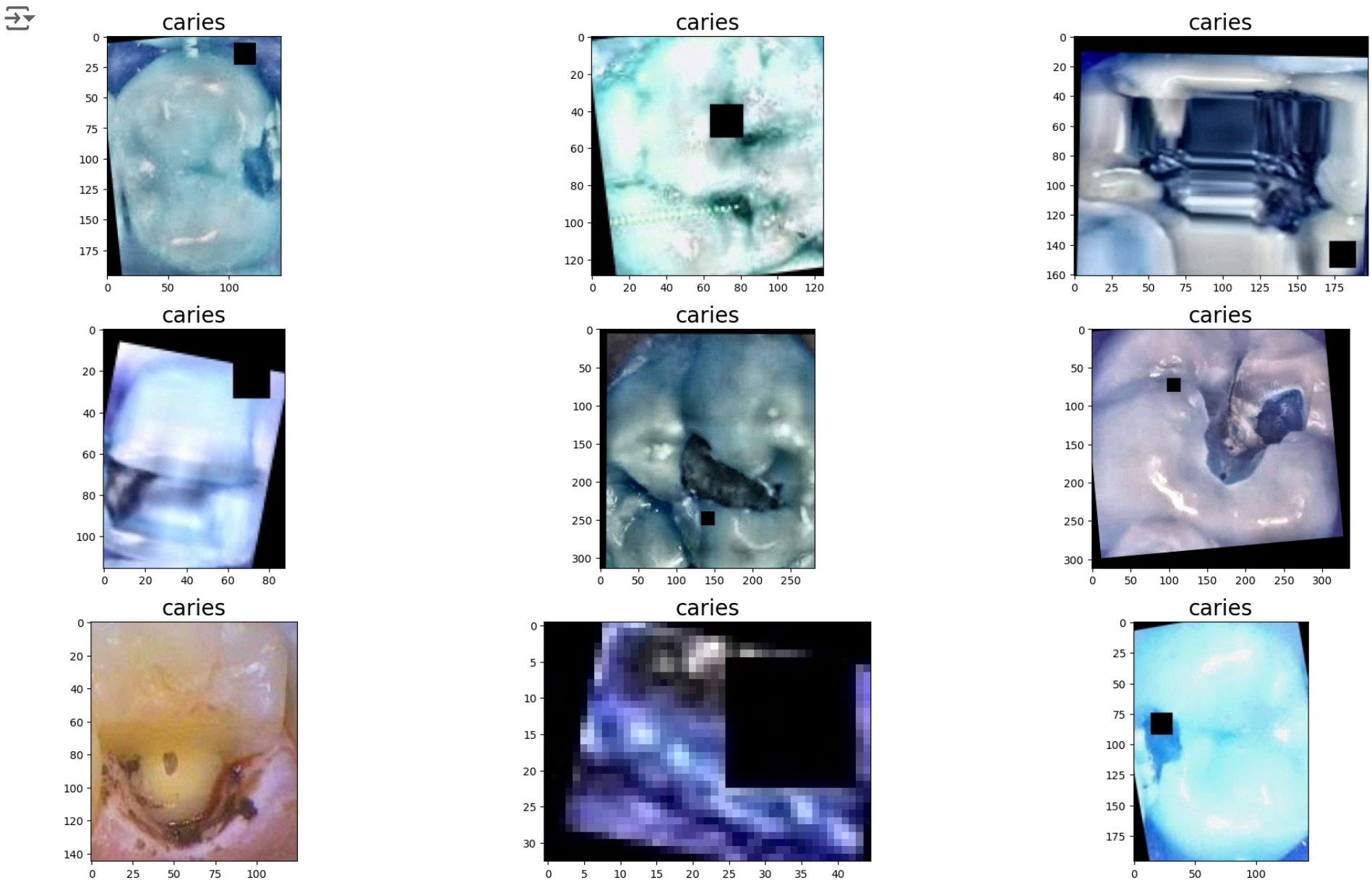
```
from collections import Counter
Counter(y_data)
```

```
Counter({'caries': 945, 'no-caries': 315})
```

```
colors = ['green','blue']
plt.pie(data.label.value_counts(),startangle=90,explode=[0.05,0.05],autopct='%0.2f%%',labels=['No Caries', 'Caries'], color
plt.show()
```




```
plt.figure(figsize=(20,15))
for i in range(9):
    plt.subplot(4,3,(i%12)+1)
    index=np.random.randint(1000)
    plt.title('{0}'.format(data.label[index]),fontdict={'size':20})
    plt.imshow(data.image[index])
    plt.tight_layout()
```




```
className = glob(train_path + '/*' )
numberOfClass = len(className)
print("Number Of Class: ",numberOfClass)
```

```
Number Of Class:  2
```

## Getting contents of testing data

```
x_data = []
y_data = []

for category in glob(test_path+'/*'):
    for file in tqdm(glob(category+'/*')):
        img_array=cv2.imread(file)
        img_array = cv2.cvtColor(img_array, cv2.COLOR_BGR2RGB)
        x_data.append(img_array)
        y_data.append(category.split("/")[-1])

data=pd.DataFrame({'image': x_data,'label': y_data})
```

```
100%|██████████| 228/228 [00:00<00:00, 1530.77it/s]
100%|██████████| 84/84 [00:00<00:00, 4802.18it/s]
```

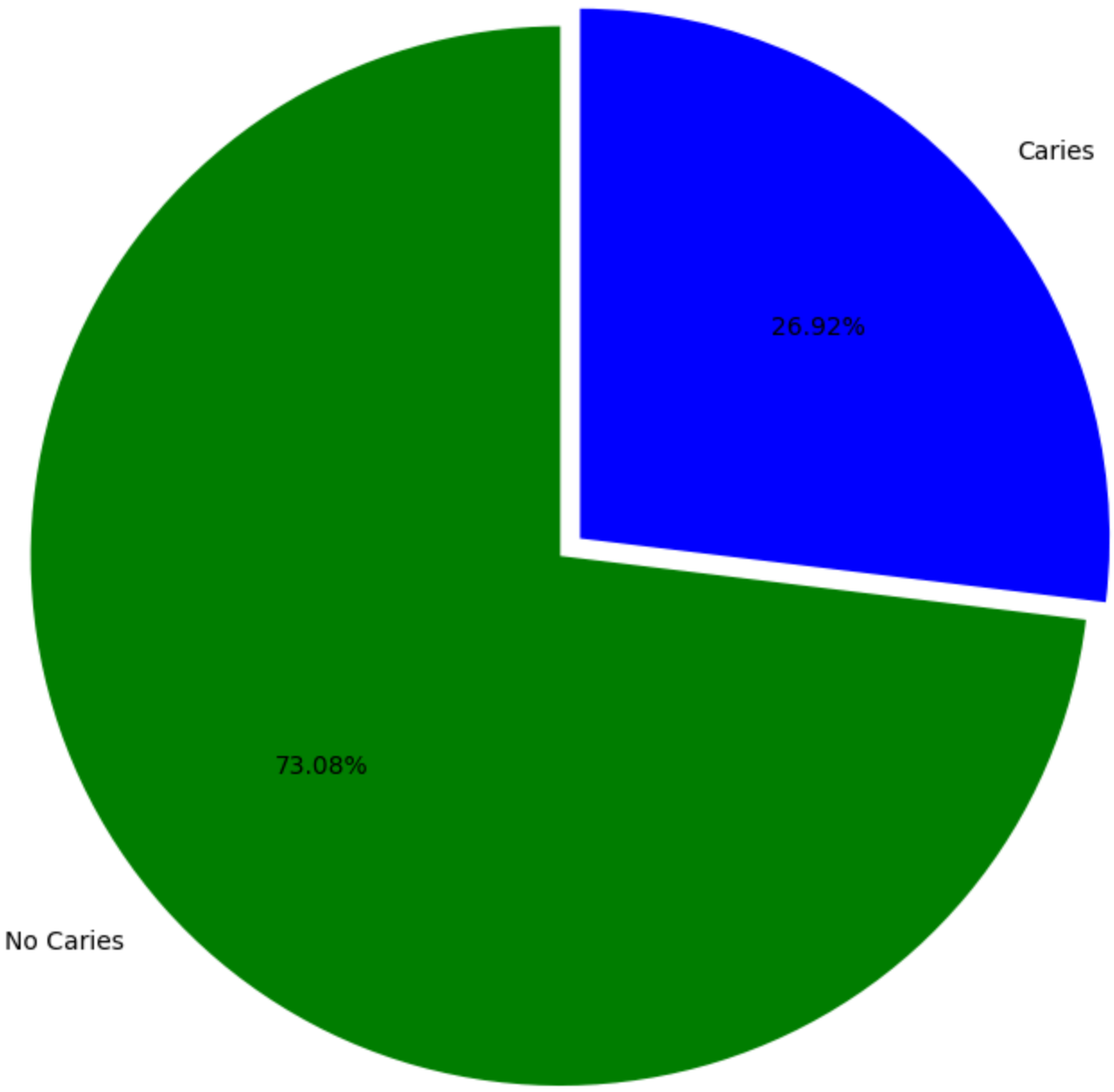
```
data.shape
```

```
(312, 2)
```

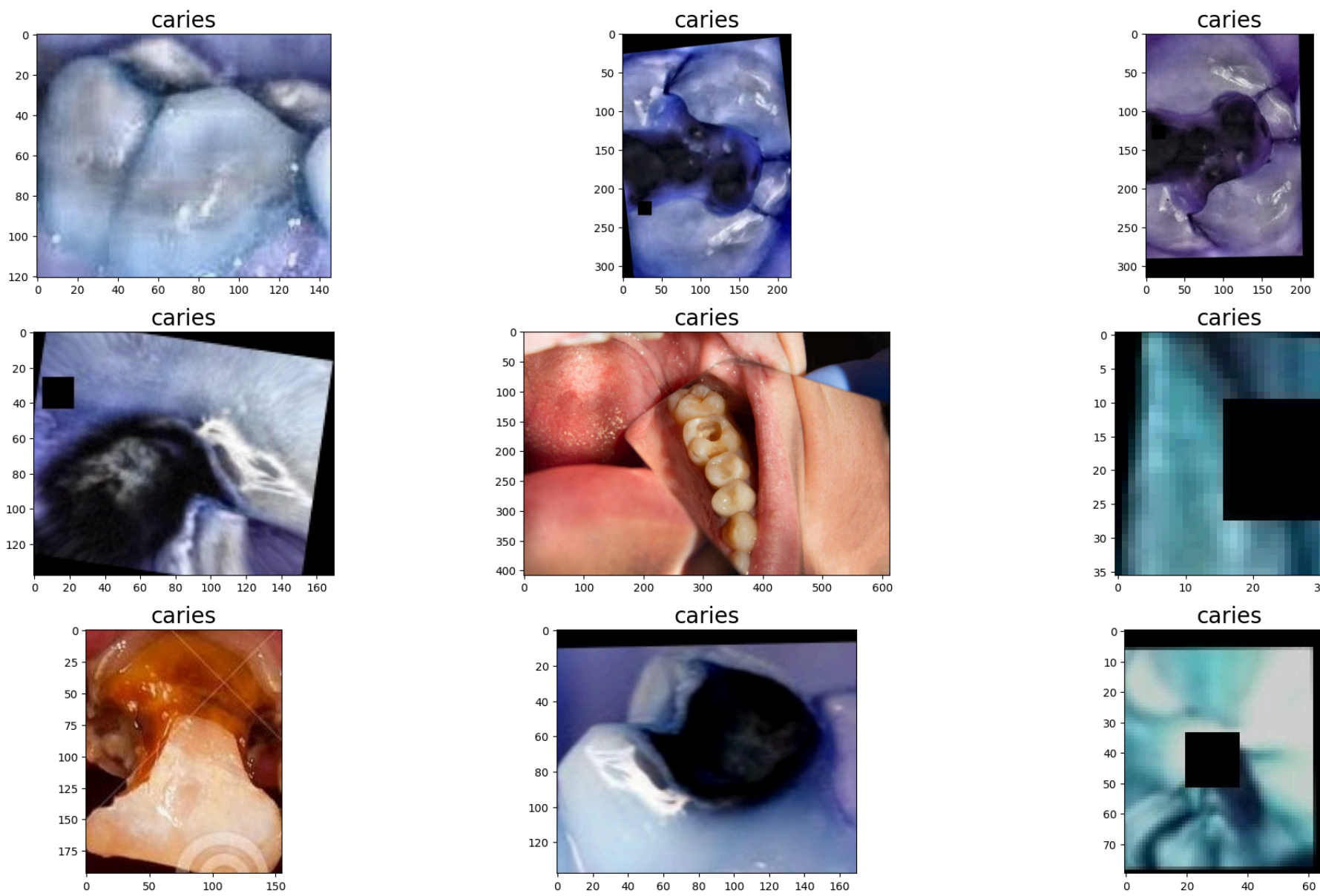
```
from collections import Counter
Counter(y_data)
```

```
Counter({'caries': 228, 'no-caries': 84})
```

```
colors = ['green','blue']
plt.pie(data.label.value_counts(),startangle=90,explode=[0.05,0.05],autopct='%0.2f%%',labels=['No Caries', 'Caries'], color
plt.show()
```




```
plt.figure(figsize=(20,15))
for i in range(9):
    plt.subplot(4,3,(i%12)+1)
    index=np.random.randint(294)
    plt.title('{0}'.format(data.label[index]),fontdict={'size':20})
    plt.imshow(data.image[index])
    plt.tight_layout()
```




## Model Building

```
train_generator = ImageDataGenerator(
    rescale = 1.0/255.,
    rotation_range=30,
    width_shift_range=0.1,
    height_shift_range=0.1,
    shear_range=0.1,
    zoom_range=0.1,
    horizontal_flip=True,
    vertical_flip=True,
    fill_mode='nearest',
    brightness_range=(0.5, 1.5),
)
test_generator = ImageDataGenerator(rescale=1./255.)
```

```
train_datagen = train_generator.flow_from_directory(
    train_path,
    batch_size=10,
    target_size=(224, 224)
)
test_datagen = test_generator.flow_from_directory(
    test_path,
    batch_size=10,
    target_size=(224, 224)
)
```

```
Found 1260 images belonging to 2 classes.
Found 312 images belonging to 2 classes.
```

## Proposed CNN Model

```
model = Sequential()

model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(224, 224, 3)))
model.add(MaxPooling2D((2, 2)))

model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))

model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))

model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))

model.add(Flatten())
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(2, activation='softmax'))

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

```
print(model.summary())
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 222, 222, 32)      896

 max_pooling2d (MaxPooling2  (None, 111, 111, 32)      0
 D)

 conv2d_1 (Conv2D)           (None, 109, 109, 64)      18496

 max_pooling2d_1 (MaxPoolin  (None, 54, 54, 64)        0
 g2D)

 conv2d_2 (Conv2D)           (None, 52, 52, 128)       73856

 max_pooling2d_2 (MaxPoolin  (None, 26, 26, 128)       0
 g2D)

 conv2d_3 (Conv2D)           (None, 24, 24, 128)       147584

 max_pooling2d_3 (MaxPoolin  (None, 12, 12, 128)       0
 g2D)
```

```
 flatten (Flatten)           (None, 18432)             0

 dense (Dense)               (None, 512)               9437696

 dropout (Dropout)           (None, 512)               0

 dense_1 (Dense)             (None, 2)                 1026

=================================================================
Total params: 9679554 (36.92 MB)
Trainable params: 9679554 (36.92 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
None
```

```
hist_cnn = model.fit_generator(generator = train_datagen,epochs=20,validation_data = test_datagen)
```

```
Epoch 1/20
126/126 [==============================] - 36s 182ms/step - loss: 0.5945 - accuracy: 0.7468 - val_loss: 0.5579 - val_ac
Epoch 2/20
126/126 [==============================] - 22s 177ms/step - loss: 0.5404 - accuracy: 0.7516 - val_loss: 0.3908 - val_ac
Epoch 3/20
126/126 [==============================] - 22s 170ms/step - loss: 0.5094 - accuracy: 0.7643 - val_loss: 0.4236 - val_ac
Epoch 4/20
126/126 [==============================] - 22s 173ms/step - loss: 0.5021 - accuracy: 0.7698 - val_loss: 0.4083 - val_ac
Epoch 5/20
126/126 [==============================] - 20s 162ms/step - loss: 0.4899 - accuracy: 0.7706 - val_loss: 0.3811 - val_ac
Epoch 6/20
126/126 [==============================] - 22s 170ms/step - loss: 0.4978 - accuracy: 0.7706 - val_loss: 0.3968 - val_ac
Epoch 7/20
126/126 [==============================] - 20s 162ms/step - loss: 0.4514 - accuracy: 0.7960 - val_loss: 0.3228 - val_ac
Epoch 8/20
126/126 [==============================] - 22s 171ms/step - loss: 0.4467 - accuracy: 0.7897 - val_loss: 0.3037 - val_ac
Epoch 9/20
126/126 [==============================] - 22s 172ms/step - loss: 0.4295 - accuracy: 0.7921 - val_loss: 0.3349 - val_ac
Epoch 10/20
126/126 [==============================] - 20s 160ms/step - loss: 0.4247 - accuracy: 0.7960 - val_loss: 0.3092 - val_ac
Epoch 11/20
126/126 [==============================] - 22s 173ms/step - loss: 0.3938 - accuracy: 0.8032 - val_loss: 0.4001 - val_ac
Epoch 12/20
126/126 [==============================] - 21s 168ms/step - loss: 0.4048 - accuracy: 0.8238 - val_loss: 0.3729 - val_ac
Epoch 13/20
126/126 [==============================] - 20s 159ms/step - loss: 0.4033 - accuracy: 0.8103 - val_loss: 0.3210 - val_ac
Epoch 14/20
126/126 [==============================] - 21s 167ms/step - loss: 0.3743 - accuracy: 0.8230 - val_loss: 0.2927 - val_ac
Epoch 15/20
126/126 [==============================] - 20s 158ms/step - loss: 0.4101 - accuracy: 0.8071 - val_loss: 0.3475 - val_ac
Epoch 16/20
126/126 [==============================] - 22s 174ms/step - loss: 0.3721 - accuracy: 0.8270 - val_loss: 0.4127 - val_ac
Epoch 17/20
126/126 [==============================] - 22s 177ms/step - loss: 0.3906 - accuracy: 0.8214 - val_loss: 0.3309 - val_ac
Epoch 18/20
126/126 [==============================] - 20s 162ms/step - loss: 0.3764 - accuracy: 0.8190 - val_loss: 0.3634 - val_ac
Epoch 19/20
126/126 [==============================] - 21s 169ms/step - loss: 0.3696 - accuracy: 0.8357 - val_loss: 0.3431 - val_ac
Epoch 20/20
126/126 [==============================] - 20s 155ms/step - loss: 0.3797 - accuracy: 0.8214 - val_loss: 0.3381 - val_ac
```

## Transfer Learning

## ResNet50

```
from tensorflow.keras.layers import Dense, Flatten, Dropout, GlobalAveragePooling2D
from tensorflow.keras.models import Model
from tensorflow.keras.applications.resnet50 import ResNet50

base_model = ResNet50(weights='imagenet', include_top=False, input_shape=(224, 224, 3))

# Freeze pre-trained layers
for layer in base_model.layers:
    layer.trainable = False

x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(512, activation='relu')(x)
x = BatchNormalization()(x)
x = Dropout(0.5)(x)
x = Dense(256, activation='relu')(x)
x = BatchNormalization()(x)
x = Dropout(0.5)(x)
x = Dense(128, activation='relu')(x)
x = BatchNormalization()(x)
x = Dropout(0.5)(x)
predictions = Dense(2, activation='softmax')(x)

model_res = Model(inputs=base_model.input, outputs=predictions)

model_res.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
hist_res = model_res.fit(train_datagen, epochs=25, validation_data=test_datagen)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/resnet/resnet50_weights_tf_dim_order
94765736/94765736 [==============================] - 3s 0us/step
Epoch 1/25
126/126 [==============================] - 32s 196ms/step - loss: 0.8482 - accuracy: 0.5897 - val_loss: 0.5780 - val_ac
Epoch 2/25
126/126 [==============================] - 22s 175ms/step - loss: 0.7010 - accuracy: 0.6730 - val_loss: 0.5908 - val_ac
Epoch 3/25
126/126 [==============================] - 23s 186ms/step - loss: 0.6163 - accuracy: 0.7151 - val_loss: 0.5290 - val_ac
Epoch 4/25
126/126 [==============================] - 24s 188ms/step - loss: 0.6213 - accuracy: 0.7286 - val_loss: 0.6505 - val_ac
Epoch 5/25
126/126 [==============================] - 23s 186ms/step - loss: 0.5865 - accuracy: 0.7198 - val_loss: 0.5348 - val_ac
Epoch 6/25
126/126 [==============================] - 23s 183ms/step - loss: 0.5570 - accuracy: 0.7167 - val_loss: 0.4607 - val_ac
Epoch 7/25
126/126 [==============================] - 24s 190ms/step - loss: 0.5559 - accuracy: 0.7238 - val_loss: 0.4736 - val_ac
Epoch 8/25
126/126 [==============================] - 23s 185ms/step - loss: 0.5494 - accuracy: 0.7294 - val_loss: 0.4942 - val_ac
Epoch 9/25
126/126 [==============================] - 23s 179ms/step - loss: 0.5342 - accuracy: 0.7373 - val_loss: 0.8879 - val_ac
Epoch 10/25
126/126 [==============================] - 23s 185ms/step - loss: 0.5159 - accuracy: 0.7476 - val_loss: 0.5141 - val_ac
Epoch 11/25
126/126 [==============================] - 24s 187ms/step - loss: 0.5026 - accuracy: 0.7540 - val_loss: 0.9520 - val_ac
Epoch 12/25
126/126 [==============================] - 24s 191ms/step - loss: 0.5235 - accuracy: 0.7500 - val_loss: 0.4913 - val_ac
Epoch 13/25
126/126 [==============================] - 23s 184ms/step - loss: 0.5110 - accuracy: 0.7357 - val_loss: 0.8938 - val_ac
Epoch 14/25
126/126 [==============================] - 22s 175ms/step - loss: 0.5093 - accuracy: 0.7500 - val_loss: 0.6056 - val_ac
Epoch 15/25
126/126 [==============================] - 23s 185ms/step - loss: 0.5086 - accuracy: 0.7460 - val_loss: 0.5095 - val_ac
Epoch 16/25
126/126 [==============================] - 23s 185ms/step - loss: 0.5168 - accuracy: 0.7484 - val_loss: 0.5191 - val_ac
Epoch 17/25
126/126 [==============================] - 23s 185ms/step - loss: 0.4984 - accuracy: 0.7484 - val_loss: 0.4967 - val_ac
Epoch 18/25
126/126 [==============================] - 23s 181ms/step - loss: 0.4946 - accuracy: 0.7587 - val_loss: 0.5452 - val_ac
Epoch 19/25
126/126 [==============================] - 24s 187ms/step - loss: 0.4948 - accuracy: 0.7429 - val_loss: 0.5021 - val_ac
Epoch 20/25
126/126 [==============================] - 23s 185ms/step - loss: 0.4823 - accuracy: 0.7571 - val_loss: 1.1181 - val_ac
Epoch 21/25
126/126 [==============================] - 24s 192ms/step - loss: 0.4992 - accuracy: 0.7444 - val_loss: 0.4686 - val_ac
Epoch 22/25
126/126 [==============================] - 22s 176ms/step - loss: 0.4802 - accuracy: 0.7595 - val_loss: 0.6353 - val_ac
Epoch 23/25
126/126 [==============================] - 23s 184ms/step - loss: 0.5014 - accuracy: 0.7437 - val_loss: 0.4121 - val_ac
Epoch 24/25
126/126 [==============================] - 23s 184ms/step - loss: 0.4899 - accuracy: 0.7532 - val_loss: 0.4362 - val_ac
Epoch 25/25
126/126 [==============================] - 23s 187ms/step - loss: 0.4718 - accuracy: 0.7619 - val_loss: 1.0615 - val_ac
```

## InceptionV3

```
from tensorflow.keras.layers import Dense, Flatten, Dropout, GlobalAveragePooling2D, MaxPooling2D
from tensorflow.keras.models import Model
from tensorflow.keras.applications.inception_v3 import InceptionV3

base_model = InceptionV3(weights='imagenet', include_top=False, input_shape=(224, 224, 3))

# Freeze pre-trained layers
for layer in base_model.layers:
    layer.trainable = False


x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(512, activation='relu')(x)
x = BatchNormalization()(x)
x = Dropout(0.5)(x)
x = Dense(256, activation='relu')(x)
x = BatchNormalization()(x)
x = Dropout(0.5)(x)
x = Dense(128, activation='relu')(x)
x = BatchNormalization()(x)
x = Dropout(0.5)(x)
predictions = Dense(2, activation='softmax')(x)

model_inc = Model(inputs=base_model.input, outputs=predictions)

model_inc.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
hist_inc = model_inc.fit(train_datagen, epochs=10, validation_data=test_datagen)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/inception_v3/inception_v3_weights_tf
87910968/87910968 [==============================] - 3s 0us/step
Epoch 1/10
126/126 [==============================] - 32s 192ms/step - loss: 0.7902 - accuracy: 0.6556 - val_loss: 0.4694 - val_ac
Epoch 2/10
126/126 [==============================] - 23s 182ms/step - loss: 0.5961 - accuracy: 0.7484 - val_loss: 0.4000 - val_ac
Epoch 3/10
126/126 [==============================] - 22s 174ms/step - loss: 0.5108 - accuracy: 0.7952 - val_loss: 0.3656 - val_ac
Epoch 4/10
126/126 [==============================] - 22s 171ms/step - loss: 0.4440 - accuracy: 0.8190 - val_loss: 0.3710 - val_ac
Epoch 5/10
126/126 [==============================] - 23s 186ms/step - loss: 0.4232 - accuracy: 0.8302 - val_loss: 0.3627 - val_ac
Epoch 6/10
126/126 [==============================] - 23s 185ms/step - loss: 0.3946 - accuracy: 0.8341 - val_loss: 0.4063 - val_ac
Epoch 7/10
126/126 [==============================] - 23s 181ms/step - loss: 0.3953 - accuracy: 0.8357 - val_loss: 0.3490 - val_ac
Epoch 8/10
126/126 [==============================] - 22s 172ms/step - loss: 0.3575 - accuracy: 0.8563 - val_loss: 0.3665 - val_ac
Epoch 9/10
126/126 [==============================] - 23s 182ms/step - loss: 0.3865 - accuracy: 0.8365 - val_loss: 0.3392 - val_ac
Epoch 10/10
126/126 [==============================] - 23s 180ms/step - loss: 0.3763 - accuracy: 0.8389 - val_loss: 0.3205 - val_ac
```

## MobileNetV2

```
from tensorflow.keras.layers import Dense, Flatten, Dropout
from tensorflow.keras.models import Model
from tensorflow.keras.applications.mobilenet_v2 import MobileNetV2

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

# Freeze pre-trained layers
for layer in base_model.layers:
    layer.trainable = False

x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(512, activation='relu')(x)
x = BatchNormalization()(x)
x = Dropout(0.5)(x)
x = Dense(256, activation='relu')(x)
x = BatchNormalization()(x)
x = Dropout(0.5)(x)
x = Dense(128, activation='relu')(x)
x = BatchNormalization()(x)
x = Dropout(0.5)(x)
predictions = Dense(2, activation='softmax')(x)

model_mob = Model(inputs=base_model.input, outputs=predictions)

model_mob.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
hist_mob = model_mob.fit(train_datagen, epochs=20, validation_data=test_datagen)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/mobilenet_v2/mobilenet_v2_weights_tf
9406464/9406464 [==============================] - 1s 0us/step
Epoch 1/20
126/126 [==============================] - 27s 169ms/step - loss: 0.7409 - accuracy: 0.6778 - val_loss: 0.5836 - val_ac
Epoch 2/20
126/126 [==============================] - 21s 165ms/step - loss: 0.5323 - accuracy: 0.7889 - val_loss: 0.6449 - val_ac
Epoch 3/20
126/126 [==============================] - 22s 174ms/step - loss: 0.4804 - accuracy: 0.8024 - val_loss: 0.5564 - val_ac
Epoch 4/20
126/126 [==============================] - 21s 167ms/step - loss: 0.4293 - accuracy: 0.8206 - val_loss: 0.4567 - val_ac
Epoch 5/20
126/126 [==============================] - 20s 158ms/step - loss: 0.4025 - accuracy: 0.8278 - val_loss: 0.5284 - val_ac
Epoch 6/20
126/126 [==============================] - 21s 166ms/step - loss: 0.3741 - accuracy: 0.8413 - val_loss: 0.4920 - val_ac
Epoch 7/20
126/126 [==============================] - 20s 160ms/step - loss: 0.3810 - accuracy: 0.8278 - val_loss: 0.4104 - val_ac
Epoch 8/20
126/126 [==============================] - 21s 163ms/step - loss: 0.3838 - accuracy: 0.8294 - val_loss: 0.4593 - val_ac
Epoch 9/20
126/126 [==============================] - 21s 164ms/step - loss: 0.3690 - accuracy: 0.8365 - val_loss: 0.4396 - val_ac
Epoch 10/20
126/126 [==============================] - 19s 152ms/step - loss: 0.3577 - accuracy: 0.8389 - val_loss: 0.4291 - val_ac
Epoch 11/20
126/126 [==============================] - 21s 166ms/step - loss: 0.3434 - accuracy: 0.8444 - val_loss: 0.3694 - val_ac
Epoch 12/20
126/126 [==============================] - 20s 156ms/step - loss: 0.3310 - accuracy: 0.8579 - val_loss: 0.4025 - val_ac
Epoch 13/20
126/126 [==============================] - 21s 165ms/step - loss: 0.3393 - accuracy: 0.8524 - val_loss: 0.4365 - val_ac
Epoch 14/20
126/126 [==============================] - 20s 158ms/step - loss: 0.3334 - accuracy: 0.8579 - val_loss: 0.5060 - val_ac
Epoch 15/20
126/126 [==============================] - 21s 165ms/step - loss: 0.3041 - accuracy: 0.8667 - val_loss: 0.4830 - val_ac
Epoch 16/20
126/126 [==============================] - 21s 169ms/step - loss: 0.3186 - accuracy: 0.8651 - val_loss: 0.4749 - val_ac
Epoch 17/20
126/126 [==============================] - 21s 168ms/step - loss: 0.3232 - accuracy: 0.8683 - val_loss: 0.4169 - val_ac
Epoch 18/20
126/126 [==============================] - 20s 162ms/step - loss: 0.3048 - accuracy: 0.8786 - val_loss: 0.3896 - val_ac
Epoch 19/20
126/126 [==============================] - 21s 170ms/step - loss: 0.3340 - accuracy: 0.8444 - val_loss: 0.4108 - val_ac
Epoch 20/20
126/126 [==============================] - 22s 171ms/step - loss: 0.3179 - accuracy: 0.8643 - val_loss: 0.4076 - val_ac
```

## VGG16

```
from tensorflow.keras.layers import Dense, Flatten, Dropout
from tensorflow.keras.models import Model
from tensorflow.keras.applications.vgg16 import VGG16

base_model = VGG16(weights='imagenet', include_top=False, input_shape=(48, 48, 3))

# Freeze pre-trained layers
for layer in base_model.layers:
    layer.trainable = False

x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(512, activation='relu')(x)
x = BatchNormalization()(x)
x = Dropout(0.5)(x)
x = Dense(256, activation='relu')(x)
x = BatchNormalization()(x)
x = Dropout(0.5)(x)
x = Dense(128, activation='relu')(x)
x = BatchNormalization()(x)
x = Dropout(0.5)(x)
predictions = Dense(2, activation='softmax')(x)

model_vgg = Model(inputs=base_model.input, outputs=predictions)

model_vgg.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
hist_vgg = model_vgg.fit(train_datagen, epochs=20, validation_data=test_datagen)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_
58889256/58889256 [==============================] - 2s 0us/step
Epoch 1/20
126/126 [==============================] - 28s 188ms/step - loss: 0.7371 - accuracy: 0.6762 - val_loss: 0.5275 - val_ac
Epoch 2/20
126/126 [==============================] - 23s 184ms/step - loss: 0.5607 - accuracy: 0.7571 - val_loss: 0.4779 - val_ac
Epoch 3/20
126/126 [==============================] - 23s 185ms/step - loss: 0.5006 - accuracy: 0.7913 - val_loss: 0.5728 - val_ac
Epoch 4/20
126/126 [==============================] - 23s 184ms/step - loss: 0.4444 - accuracy: 0.8119 - val_loss: 0.4403 - val_ac
Epoch 5/20
126/126 [==============================] - 23s 185ms/step - loss: 0.4375 - accuracy: 0.8040 - val_loss: 0.3854 - val_ac
Epoch 6/20
126/126 [==============================] - 23s 186ms/step - loss: 0.4093 - accuracy: 0.8230 - val_loss: 0.4081 - val_ac
Epoch 7/20
126/126 [==============================] - 22s 175ms/step - loss: 0.4139 - accuracy: 0.8159 - val_loss: 0.4466 - val_ac
Epoch 8/20
126/126 [==============================] - 23s 183ms/step - loss: 0.3751 - accuracy: 0.8381 - val_loss: 0.3822 - val_ac
Epoch 9/20
126/126 [==============================] - 23s 182ms/step - loss: 0.4043 - accuracy: 0.8246 - val_loss: 0.2917 - val_ac
Epoch 10/20
126/126 [==============================] - 23s 184ms/step - loss: 0.3527 - accuracy: 0.8437 - val_loss: 0.3810 - val_ac
Epoch 11/20
126/126 [==============================] - 23s 184ms/step - loss: 0.3905 - accuracy: 0.8294 - val_loss: 0.3375 - val_ac
Epoch 12/20
126/126 [==============================] - 25s 199ms/step - loss: 0.3585 - accuracy: 0.8405 - val_loss: 0.3634 - val_ac
Epoch 13/20
126/126 [==============================] - 25s 199ms/step - loss: 0.3489 - accuracy: 0.8524 - val_loss: 0.3904 - val_ac
Epoch 14/20
126/126 [==============================] - 25s 200ms/step - loss: 0.3585 - accuracy: 0.8373 - val_loss: 0.4102 - val_ac
Epoch 15/20
126/126 [==============================] - 25s 200ms/step - loss: 0.3710 - accuracy: 0.8357 - val_loss: 0.3459 - val_ac
Epoch 16/20
126/126 [==============================] - 26s 203ms/step - loss: 0.3669 - accuracy: 0.8429 - val_loss: 0.3558 - val_ac
Epoch 17/20
126/126 [==============================] - 25s 197ms/step - loss: 0.3590 - accuracy: 0.8349 - val_loss: 0.3482 - val_ac
Epoch 18/20
126/126 [==============================] - 25s 198ms/step - loss: 0.3342 - accuracy: 0.8524 - val_loss: 0.3615 - val_ac
Epoch 19/20
126/126 [==============================] - 24s 193ms/step - loss: 0.3584 - accuracy: 0.8444 - val_loss: 0.3832 - val_ac
Epoch 20/20
126/126 [==============================] - 23s 183ms/step - loss: 0.3318 - accuracy: 0.8571 - val_loss: 0.3754 - val_ac
```

# Model Evaluation

```
# Creating holders to store the model performance results
Deep_Learning_Model = []
Training_accuracy = []
Validation_accuracy = []
Training_loss = []
Validation_loss = []

#function to call for storing the results
def storeResults(model, a,b,c,d):
  Deep_Learning_Model.append(model)
  Training_accuracy.append(round(a, 3))
  Validation_accuracy.append(round(b, 3))
  Training_loss.append(round(c, 3))
  Validation_loss.append(round(d, 3))
```

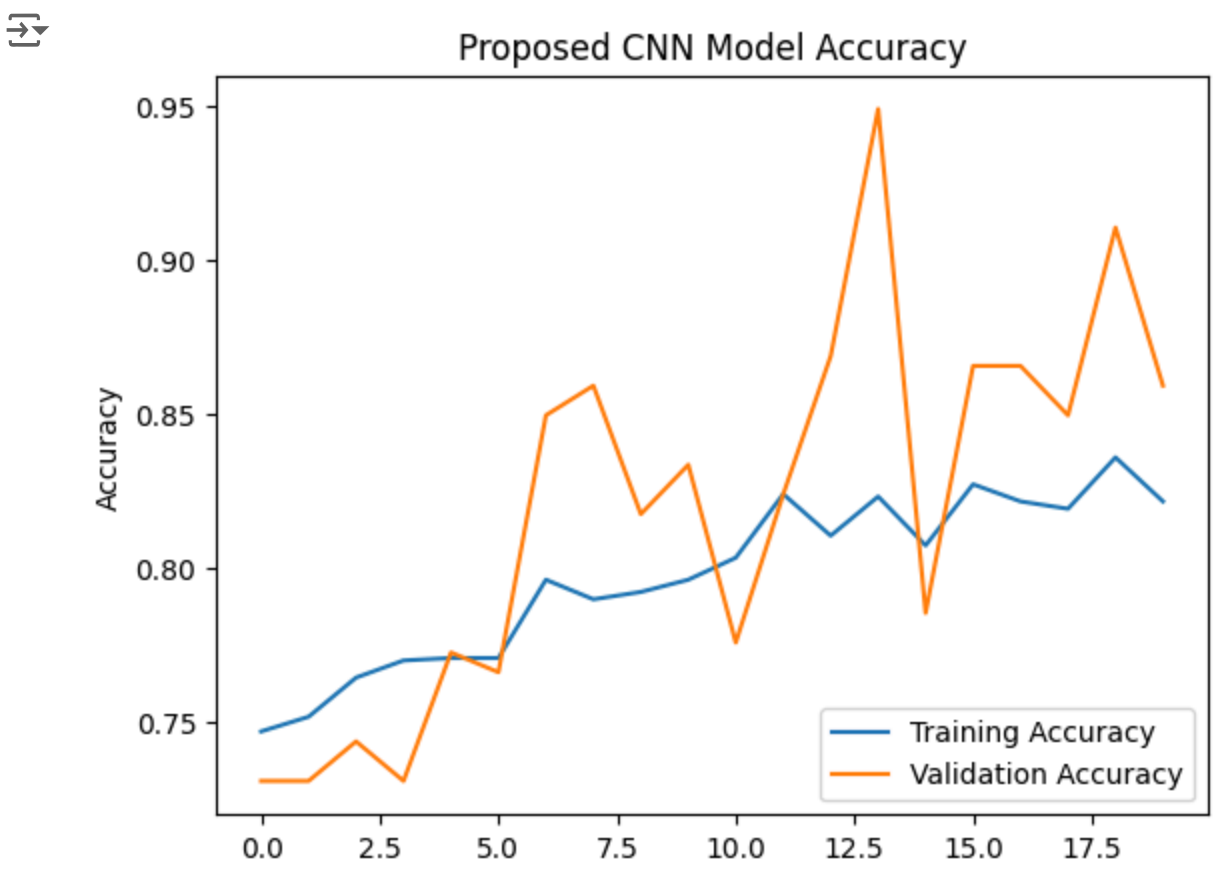
# Proposed CNN Model

## Accuracy Curve

```
acc = hist_cnn.history['accuracy']
val_acc = hist_cnn.history['val_accuracy']

acc1 = hist_cnn.history['accuracy'][-1]
val_acc1 = hist_cnn.history['val_accuracy'][-1]

plt.plot(acc, label='Training Accuracy')
plt.plot(val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.ylabel('Accuracy')
plt.ylim([min(plt.ylim()),max(plt.ylim())])
plt.title('Proposed CNN Model Accuracy')
plt.show()
```




## Loss Curve

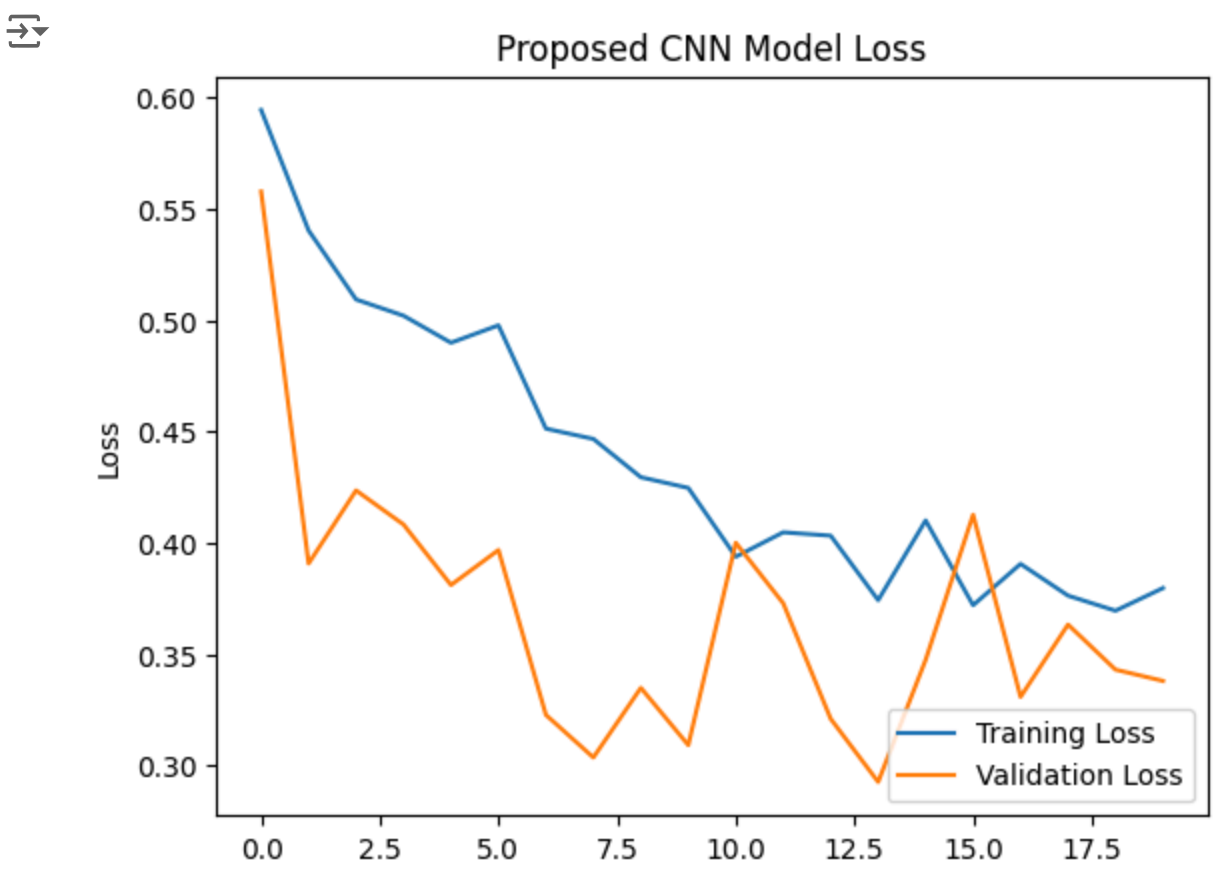
```
loss = hist_cnn.history['loss']
val_loss = hist_cnn.history['val_loss']

loss1 = hist_cnn.history['loss'][-1]
val_loss1 = hist_cnn.history['val_loss'][-1]

plt.plot(loss, label='Training Loss')
plt.plot(val_loss, label='Validation Loss')
plt.legend(loc='lower right')
plt.ylabel('Loss')
plt.ylim([min(plt.ylim()),max(plt.ylim())])
plt.title('Proposed CNN Model Loss')
plt.show()

#storing the results. The below mentioned order of parameter passing is important.

storeResults('Proposed CNN Model',acc1,val_acc1,
             loss1,val_loss1)
```



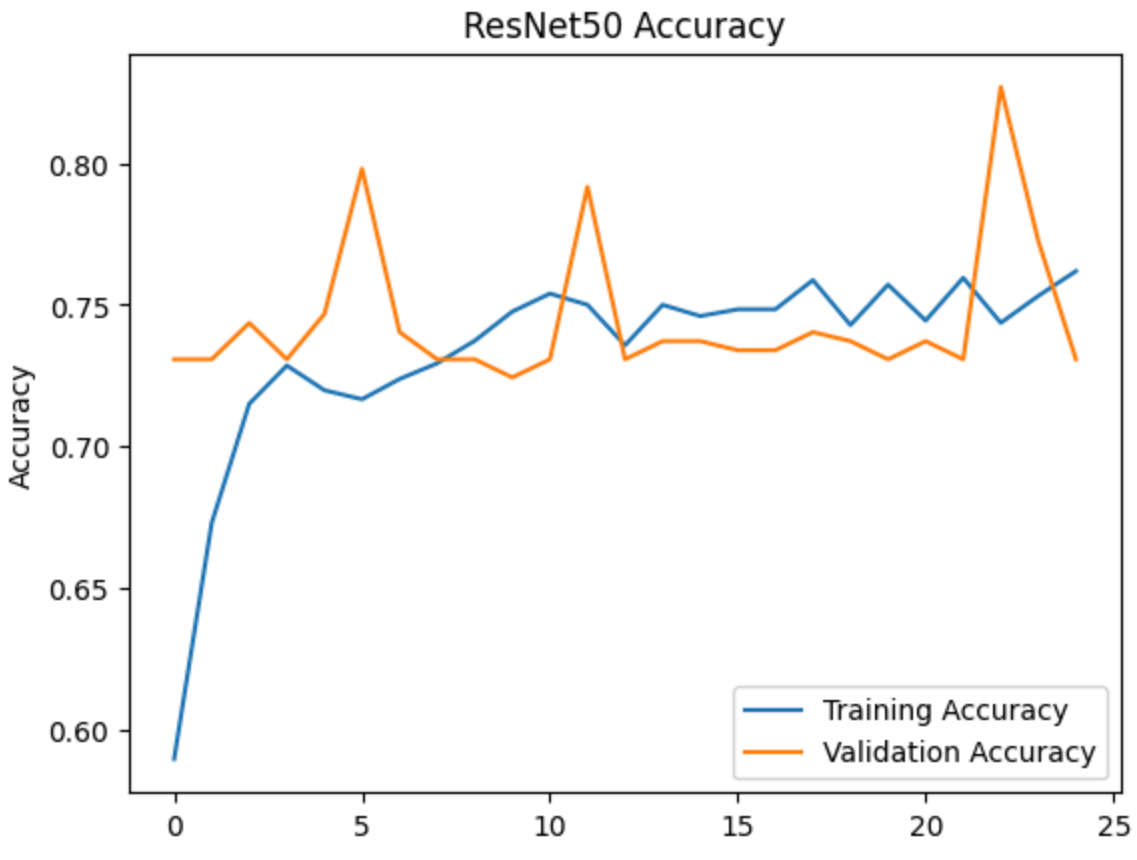

## ResNet50

### Accuracy Curve

```
acc = hist_res.history['accuracy']
val_acc = hist_res.history['val_accuracy']

acc1 = hist_res.history['accuracy'][-1]
val_acc1 = hist_res.history['val_accuracy'][-1]

plt.plot(acc, label='Training Accuracy')
plt.plot(val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.ylabel('Accuracy')
plt.ylim([min(plt.ylim()),max(plt.ylim())])
plt.title('ResNet50 Accuracy')
plt.show()
```




## Loss Curve

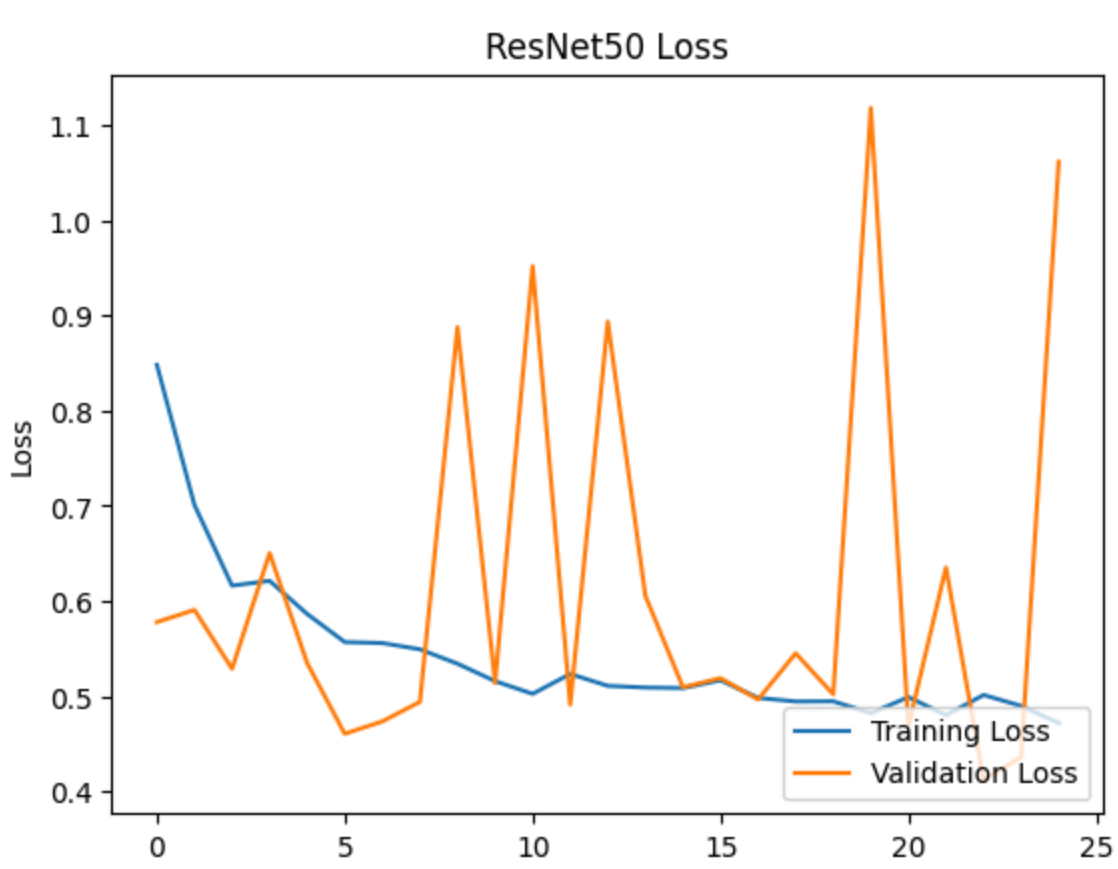
```
loss = hist_res.history['loss']
val_loss = hist_res.history['val_loss']

loss1 = hist_res.history['loss'][-1]
val_loss1 = hist_res.history['val_loss'][-1]

plt.plot(loss, label='Training Loss')
plt.plot(val_loss, label='Validation Loss')
plt.legend(loc='lower right')
plt.ylabel('Loss')
plt.ylim([min(plt.ylim()),max(plt.ylim())])
plt.title('ResNet50 Loss')
plt.show()

#storing the results. The below mentioned order of parameter passing is important.

storeResults('ResNet50',acc1,val_acc1,
             loss1,val_loss1)
```



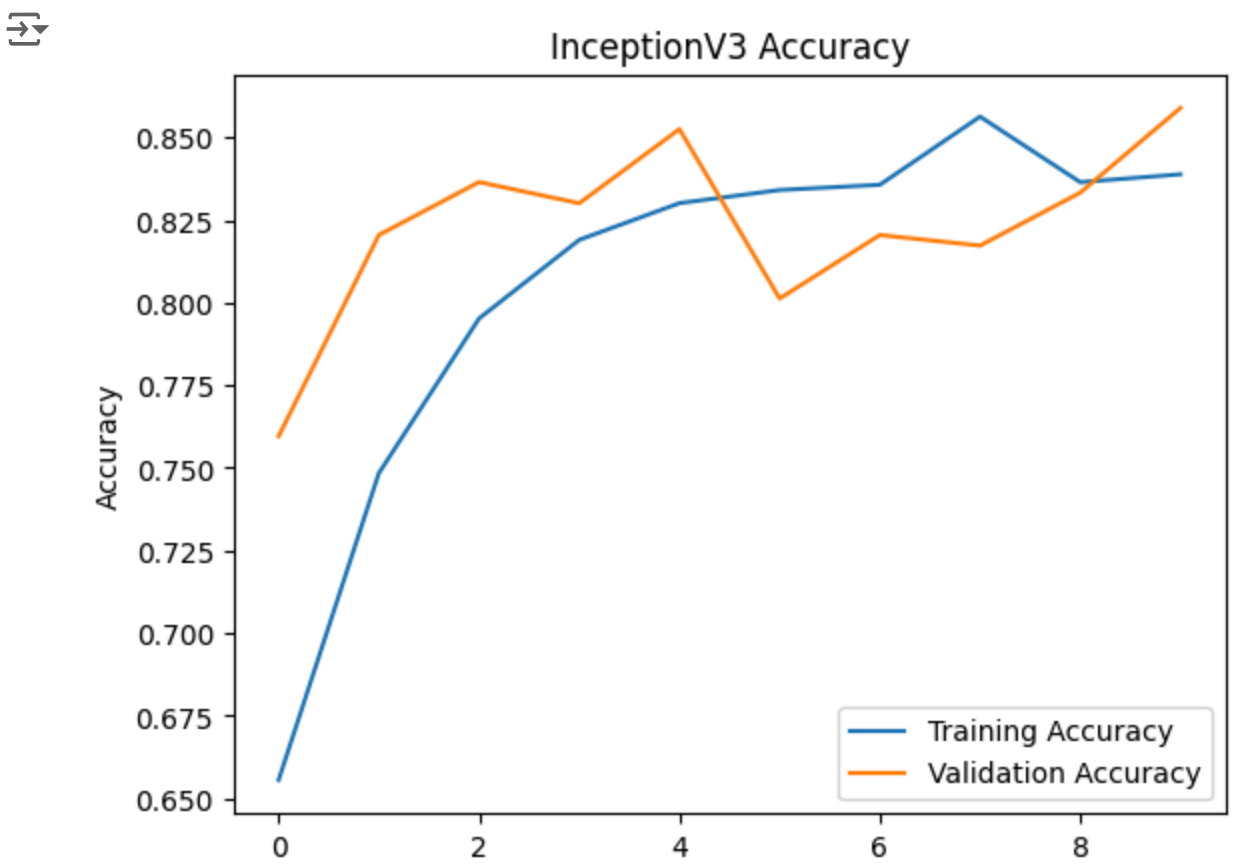

## InceptionV3

## Accuracy Curve

```
acc = hist_inc.history['accuracy']
val_acc = hist_inc.history['val_accuracy']

acc1 = hist_inc.history['accuracy'][-1]
val_acc1 = hist_inc.history['val_accuracy'][-1]

plt.plot(acc, label='Training Accuracy')
plt.plot(val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.ylabel('Accuracy')
plt.ylim([min(plt.ylim()),max(plt.ylim())])
plt.title('InceptionV3 Accuracy')
plt.show()
```




## Loss Curve

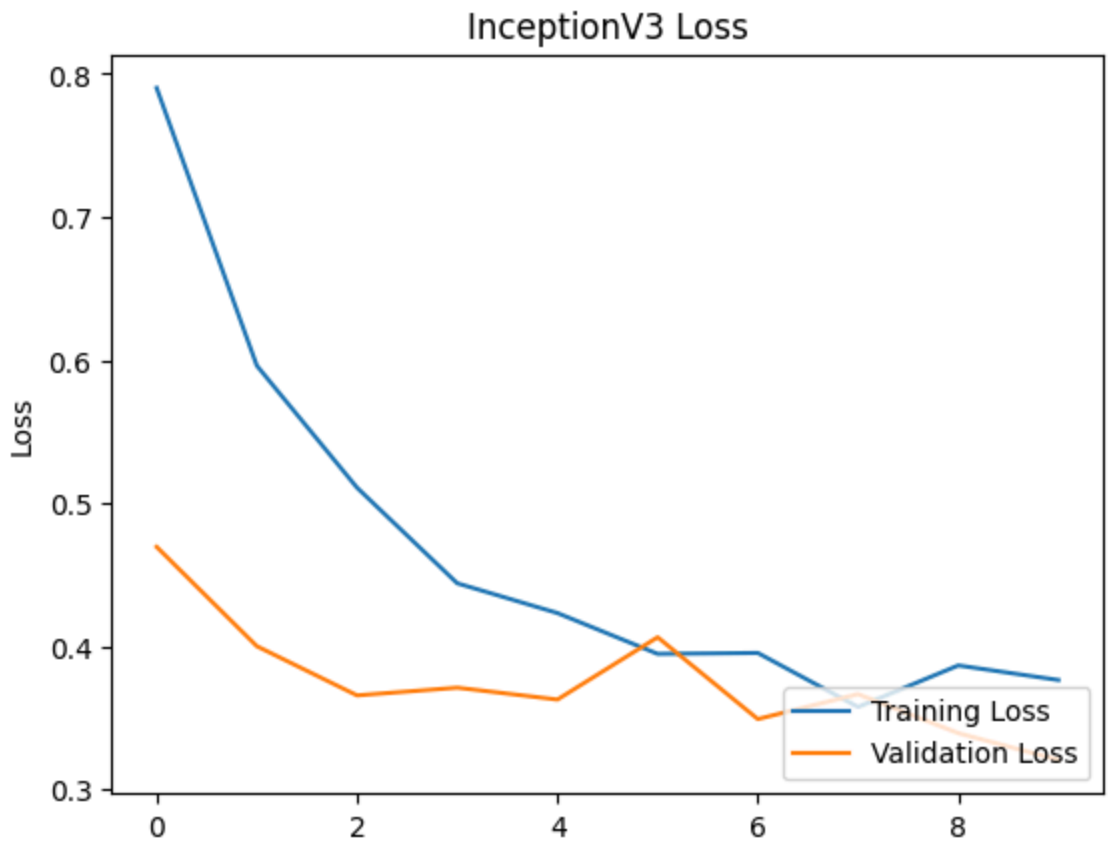
```
loss = hist_inc.history['loss']
val_loss = hist_inc.history['val_loss']

loss1 = hist_inc.history['loss'][-1]
val_loss1 = hist_inc.history['val_loss'][-1]

plt.plot(loss, label='Training Loss')
plt.plot(val_loss, label='Validation Loss')
plt.legend(loc='lower right')
plt.ylabel('Loss')
plt.ylim([min(plt.ylim()),max(plt.ylim())])
plt.title('InceptionV3 Loss')
plt.show()
```

```
#storing the results. The below mentioned order of parameter passing is important.

storeResults('InceptionV3',acc1,val_acc1,
             loss1,val_loss1)
```



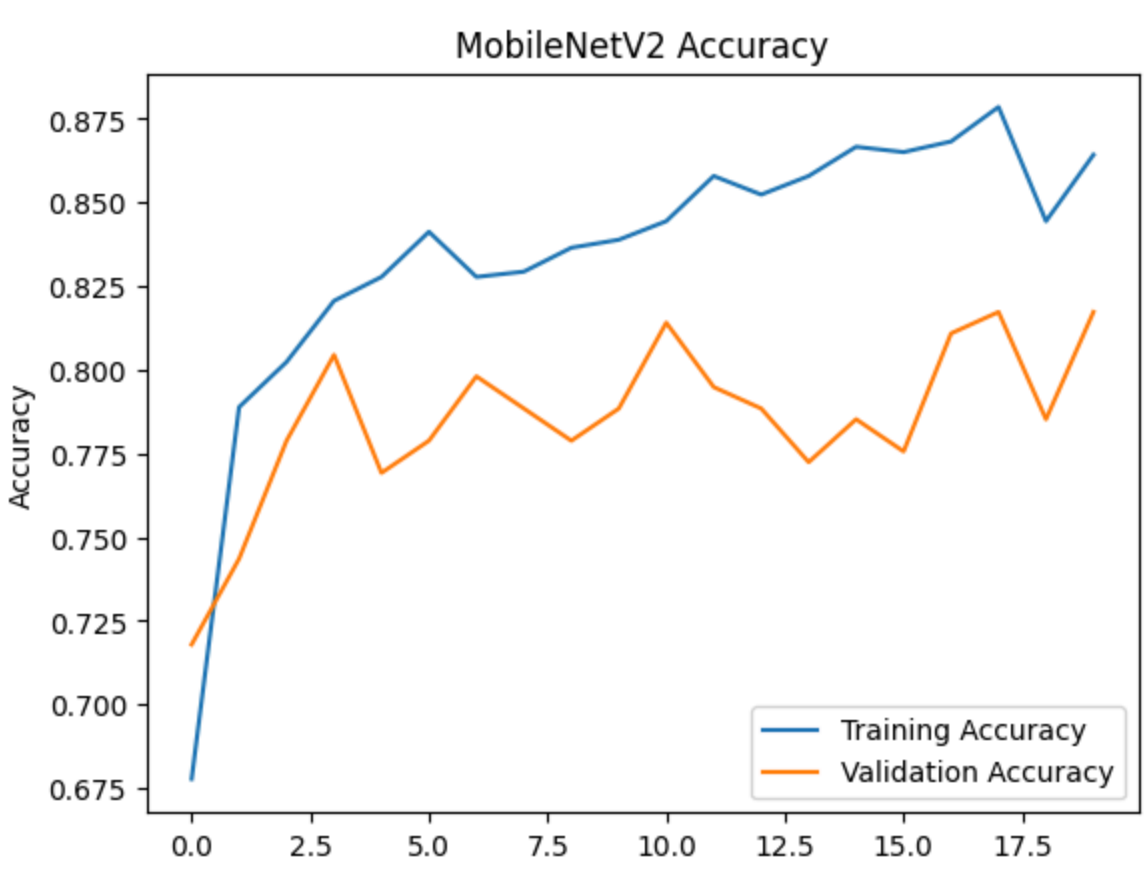

## MobileNetV2

### Accuracy Curve

```
acc = hist_mob.history['accuracy']
val_acc = hist_mob.history['val_accuracy']

acc1 = hist_mob.history['accuracy'][-1]
val_acc1 = hist_mob.history['val_accuracy'][-1]

plt.plot(acc, label='Training Accuracy')
plt.plot(val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.ylabel('Accuracy')
plt.ylim([min(plt.ylim()),max(plt.ylim())])
plt.title('MobileNetV2 Accuracy')
plt.show()
```




### Loss Curve

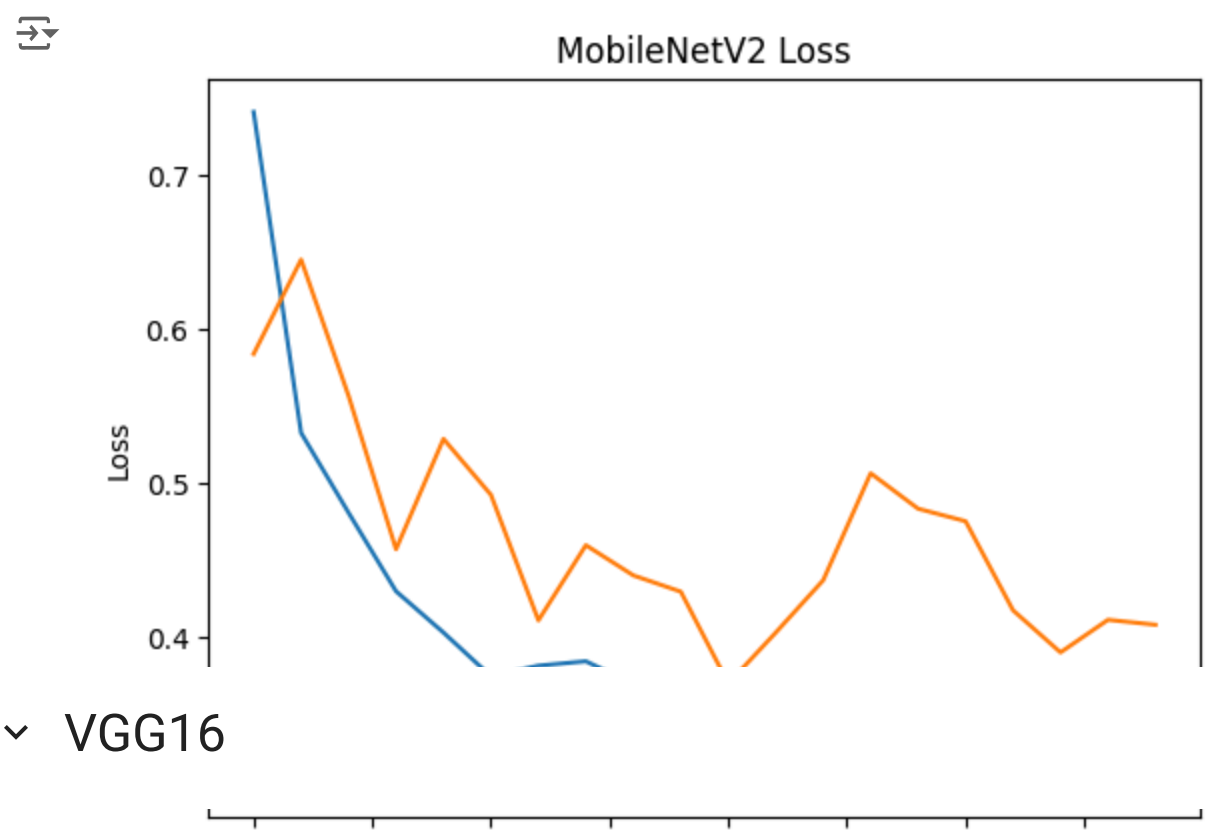
```
loss = hist_mob.history['loss']
val_loss = hist_mob.history['val_loss']

loss1 = hist_mob.history['loss'][-1]
val_loss1 = hist_mob.history['val_loss'][-1]

plt.plot(loss, label='Training Loss')
plt.plot(val_loss, label='Validation Loss')
plt.legend(loc='lower right')
plt.ylabel('Loss')
plt.ylim([min(plt.ylim()),max(plt.ylim())])
plt.title('MobileNetV2 Loss')
plt.show()

#storing the results. The below mentioned order of parameter passing is important.

storeResults('MobileNetV2',acc1,val_acc1,
             loss1,val_loss1)
```




## VGG16

## Accuracy Curve

```
acc = hist_vgg.history['accuracy']
val_acc = hist_vgg.history['val_accuracy']

acc1 = hist_vgg.history['accuracy'][-1]
val_acc1 = hist_vgg.history['val_accuracy'][-1]

plt.plot(acc, label='Training Accuracy')
plt.plot(val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.ylabel('Accuracy')
plt.ylim([min(plt.ylim()),max(plt.ylim())])
```