**Resnet Model Implementation+Training in Python for Canesat & Napane**

**Program: -**

import os

import numpy as np

import matplotlib.pyplot as plt

import tensorflow as tf

from tensorflow.keras.models import Sequential, Model

from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, InputLayer, Input, BatchNormalization, Activation, Add, GlobalAveragePooling2D

from tensorflow.keras.preprocessing.image import ImageDataGenerator

from sklearn.metrics import classification\_report, confusion\_matrix

from skimage import io, transform

from sklearn.model\_selection import train\_test\_split

from sklearn.svm import SVC

from sklearn.ensemble import RandomForestClassifier

def residual\_block(x, filters, kernel\_size=3, stride=1):

    shortcut = x

    x = Conv2D(filters, kernel\_size, strides=stride, padding='same')(x)

    x = BatchNormalization()(x)

    x = Activation('relu')(x)

    x = Conv2D(filters, kernel\_size, padding='same')(x)

    x = BatchNormalization()(x)

    if stride != 1 or x.shape[-1] != shortcut.shape[-1]:

        shortcut = Conv2D(filters, (1, 1), strides=stride, padding='same')(shortcut)

        shortcut = BatchNormalization()(shortcut)

    x = Add()([x, shortcut])

    x = Activation('relu')(x)

    return x

def build\_resnet(input\_shape, num\_classes, num\_blocks=[2, 2, 2, 2]):

    inputs = Input(shape=input\_shape)

    x = Conv2D(64, (7, 7), strides=2, padding='same')(inputs)

    x = BatchNormalization()(x)

    x = Activation('relu')(x)

    x = MaxPooling2D((3, 3), strides=2, padding='same')(x)

    for i, block\_num in enumerate(num\_blocks):

        for j in range(block\_num):

            if i == 0 and j == 0:

                pass

            else:

                strides = 2 if j == 0 else 1

                filters = 64 \* 2\*\*i

                x = residual\_block(x, filters, stride=strides)

    x = GlobalAveragePooling2D()(x)

    outputs = Dense(num\_classes, activation='softmax')(x)

    model = Model(inputs=inputs, outputs=outputs)

    return model

input\_shape = (100, 100, 1)

num\_classes = 2

model = build\_resnet(input\_shape, num\_classes)

model.summary()

model.compile(optimizer='adam', loss='sparse\_categorical\_crossentropy', metrics=['accuracy'])

history = model.fit(X\_train, y\_train, epochs=10, validation\_data=(X\_test, y\_test))

test\_loss, test\_acc = model.evaluate(X\_test, y\_test)

print('Test accuracy:', test\_acc)

y\_pred = model.predict(X\_test)

y\_pred\_classes = np.argmax(y\_pred, axis=1)

report = classification\_report(y\_test, y\_pred\_classes)

print(report)

cm = confusion\_matrix(y\_test, y\_pred\_classes)

print(cm)

# Update HTML report with results

html\_report = f'''

<!DOCTYPE html>

<html>

<head>

  <style>

    ... (styling)

  </style>

</head>

<body>

  <h2>Classification Report</h2>

  <table>

    <tr>

      <th>Model</th>

      <th>Accuracy</th>

      <th>Precision (Sparse/Dense)</th>

      <th>Recall (Sparse/Dense)</th>

      <th>F1-score (Sparse/Dense)</th>

      <th>SVM Accuracy as FE</th>

      <th>RF Accuracy as FE</th>

      <th>SVM Precision as FE</th>

      <th>RF Precision as FE</th>

      <th>SVM Recall as FE</th>

      <th>RF Recall as FE</th>

    </tr>

    <tr>

      <td>ResNet</td>

      <td>{test\_acc:.4f}</td>

      <td>{report['1']['precision']:.4f}/{report['0']['precision']:.4f}</td>

      <td>{report['1']['recall']:.4f}/{report['0']['recall']:.4f}</td>

      <td>{report['1']['f1-score']:.4f}/{report['0']['f1-score']:.4f}</td>

      <td></td>  # Add SVM and RF results later

      <td></td>

      <td></td>

      <td></td>

      <td></td>

      <td></td>

    </tr>

  </table>

</body>

</html>

'''

# Write the HTML report to a file

with open("classification\_report.html", "w") as file:

  file.write(html\_report)

# Display the HTML report as output

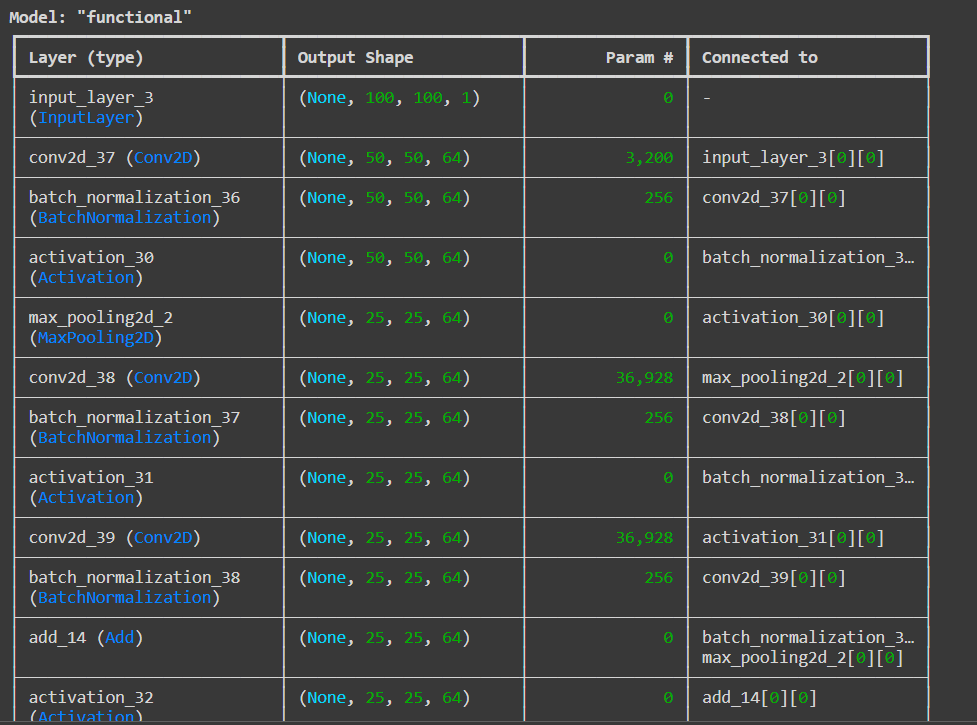
from IPython.display import display, HTML

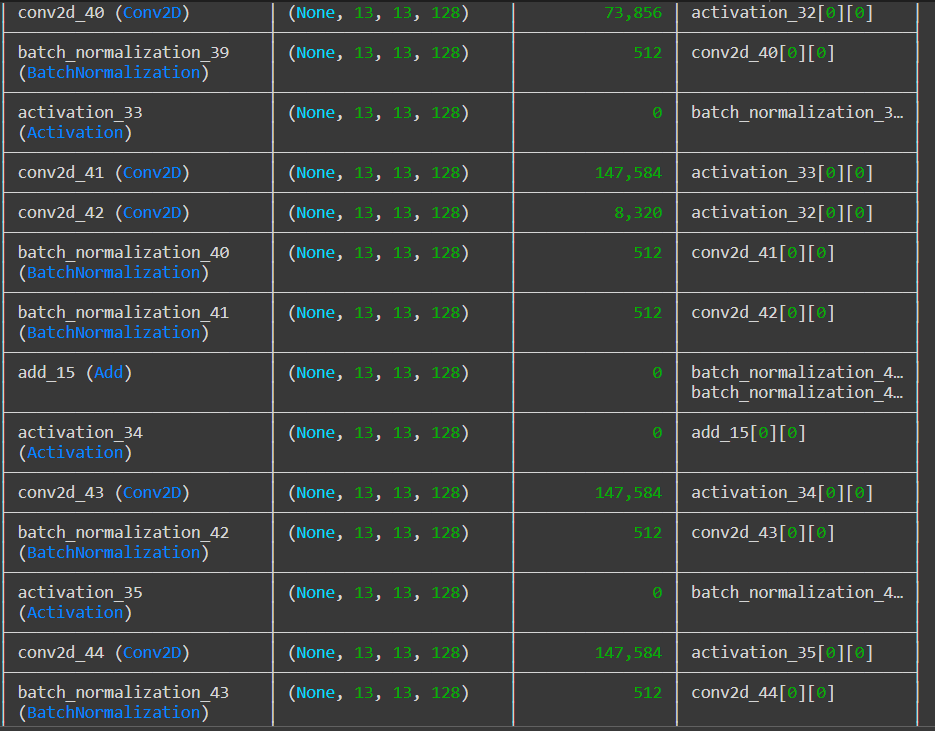
display(HTML(html\_report))

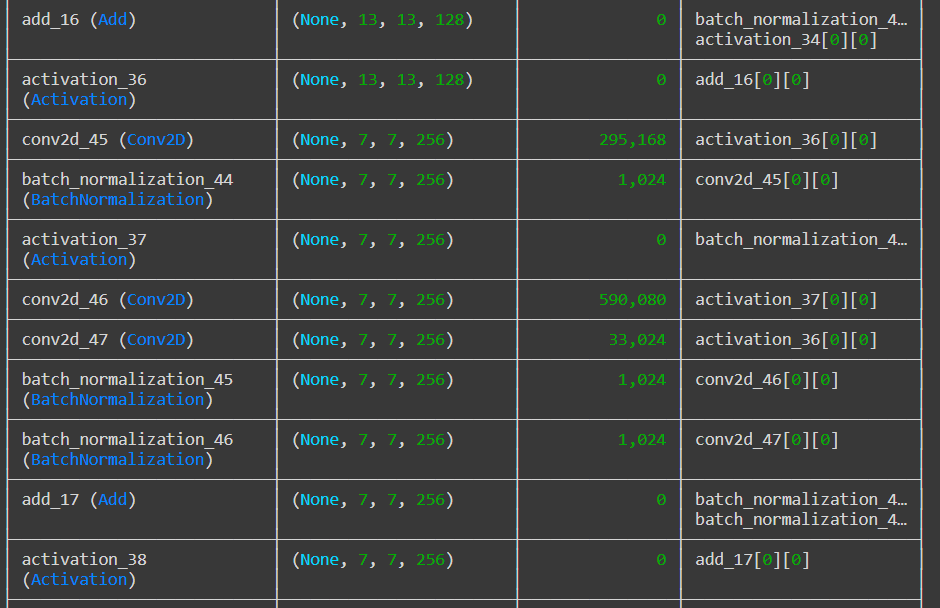
# Save the trained model

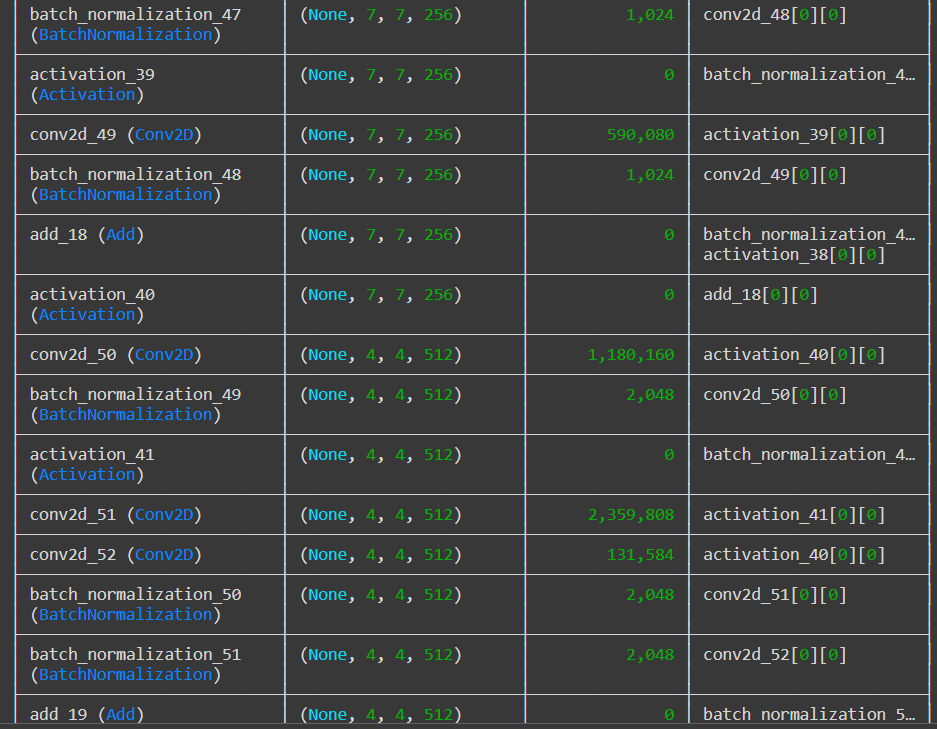
model.save('resnet\_model.h5')

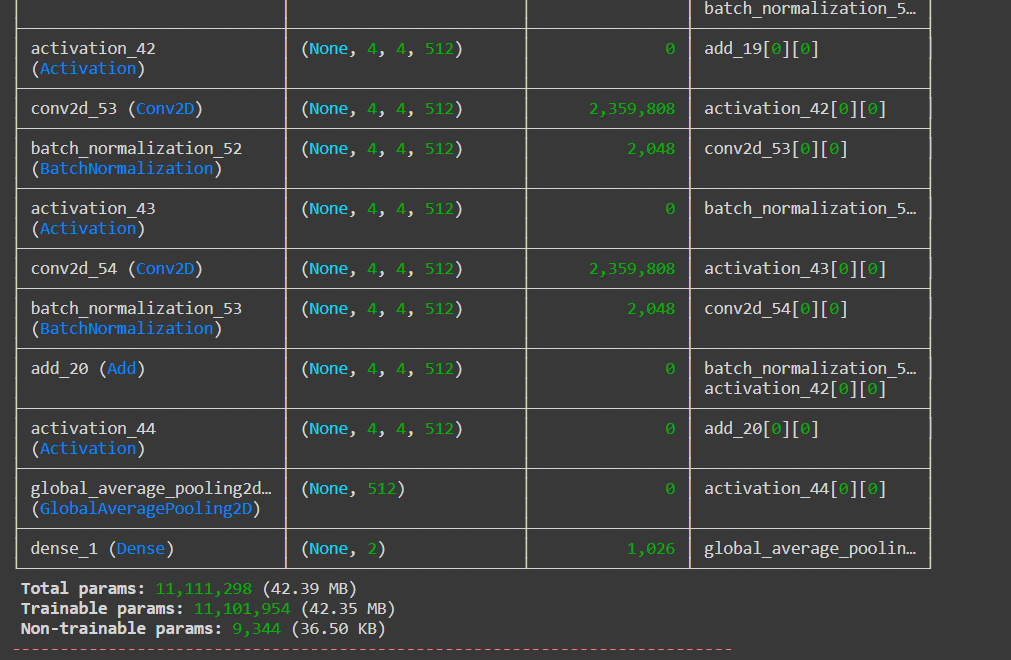
**Output: -**

****





****

****