import os

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

import tensorflow as tf

from tensorflow.keras.preprocessing.image import ImageDataGenerator

from tensorflow.keras.models import Sequential

from tensorflow.keras.applications import ResNet50

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

from tensorflow.keras.optimizers import Adam

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

**#CONSTs**

IMG\_WIDTH, IMG\_HEIGHT = 150, 150

BATCH\_SIZE = 32

**# Importing data** First I have to distribute the data in two main directories train\_dir and test\_dir after that I create two subdirectories Valid and Invalid in above mentioned directories respectively.

train\_dir = r"D:\asdf\new\_project\Train\_data"

train\_Valid = len(os.listdir(os.path.join(train\_dir, "Valid")))

train\_Invalid = len(os.listdir(os.path.join(train\_dir, "Invalid")))

test\_dir = r"D:\asdf\new\_project\Test\_data"

test\_Valid = len(os.listdir(os.path.join(test\_dir, "Valid")))

test\_Invalid = len(os.listdir(os.path.join(test\_dir, "Invalid")))

**# Simple EDA**

plt.figure(figsize=(20, 10))

sns.barplot(x=["Train", "Test"], y=[train\_Valid + train\_Invalid, test\_Valid + test\_Invalid])

plt.title("Distribution of Imgs")

plt.ylabel("Count")

[plt.show](http://plt.show)()

**# Data Augmentation**

Data augmentation is a technique used in machine learning and deep learning to artificially increase the size of a dataset by applying various transformations to the existing data. It is particularly useful when dealing with limited data, as it can help improve model performance, reduce overfitting, and increase the generalization of machine learning models. Data augmentation techniques are commonly applied to image, text, and audio data, among others.

Here are some common data augmentation techniques for image data:

1. **Rotation**: Rotate the image by a certain degree (e.g., 90 degrees, 180 degrees) while keeping the content the same.
2. **Flip**: Horizontally or vertically flip the image.
3. **Scaling and Zooming**: Resize the image to a different size or crop it to focus on a specific region of interest.
4. **Brightness and Contrast Adjustments**: Modify the brightness and contrast of the image to simulate different lighting conditions.
5. **Color Jitter**: Randomly change the color attributes of the image, such as hue, saturation, and brightness.
6. **Noise Addition**: Add random noise (e.g., Gaussian noise) to the image to simulate noise in real-world scenarios.
7. **Blur and Sharpen**: Apply various blurring or sharpening filters to the image.
8. **Translation**: Shift the image in the horizontal and vertical directions.
9. **Shearing**: Distort the image by changing the angles of its sides.
10. **Combining Transformations**: Apply multiple transformations together, such as rotation, scaling, and cropping, to create more diverse data.

train\_datagen = ImageDataGenerator(

rescale=1.0/255.0,

rotation\_range=30,

width\_shift\_range=0.2,

height\_shift\_range=0.2,

shear\_range=0.2,

zoom\_range=0.2,

horizontal\_flip=True,

fill\_mode='nearest'

)

test\_datagen = ImageDataGenerator(rescale=1.0/255.0)

train\_generator = train\_datagen.flow\_from\_directory(

train\_dir,

target\_size=(IMG\_WIDTH, IMG\_HEIGHT),

batch\_size=BATCH\_SIZE,

class\_mode='binary'

)

test\_generator = test\_datagen.flow\_from\_directory(

test\_dir,

target\_size=(IMG\_WIDTH, IMG\_HEIGHT),

batch\_size=BATCH\_SIZE,

class\_mode='binary'

)

o/p

Found 35 images belonging to 2 classes.

Found 29 images belonging to 2 classes.

# Model here i have first train the sequential model and then R-CNN model.

model = Sequential()

model.add(Conv2D(32, (3, 3), activation='relu', input\_shape=(IMG\_WIDTH, IMG\_HEIGHT, 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(Flatten())

model.add(Dense(128, activation='relu'))

model.add(Dropout(0.5))

model.add(Dense(1, activation='sigmoid'))

model.compile(loss='binary\_crossentropy', optimizer=Adam(learning\_rate=0.0001), metrics=['accuracy'])

history = [model.fit](http://model.fit)(

train\_generator,

steps\_per\_epoch=train\_generator.samples // BATCH\_SIZE,

epochs=30,

)

**Result and Evalution of sequential model**

test\_results = model.evaluate(test\_generator, steps=test\_generator.samples)

print("Test Loss:", test\_results[0])

print("Test Accuracy:", test\_results[1])

predictions = model.predict(test\_generator)

y\_pred = (predictions > 0.5).astype(int)

print(classification\_report(test\_generator.classes, y\_pred, target\_names=["Valid", "Invalid"]))

confusion\_mtx = confusion\_matrix(test\_generator.classes, y\_pred)

plt.figure(figsize=(6, 6))

sns.heatmap(confusion\_mtx, annot=True, fmt='d', cmap='Blues', cbar=False, square=True, xticklabels=["Valid", "Invalid"], yticklabels=["Valid", "Invalid"])

plt.xlabel('Predicted')

plt.ylabel('True')

plt.title('Confusion Matrix')

[plt.show](http://plt.show)()

1/29 [>.............................] - ETA: 11s - loss: 0.6966 - accuracy: 0.4828WARNING:tensorflow:Your input ran out of data; interrupting training. Make sure that your dataset or generator can generate at least `steps\_per\_epoch \* epochs` batches (in this case, 29 batches). You may need to use the repeat() function when building your dataset.

29/29 [==============================] - 0s 695us/step - loss: 0.6966 - accuracy: 0.4828

Test Loss: 0.6965624094009399

Test Accuracy: 0.48275861144065857

1/1 [==============================] - 0s 372ms/step

precision recall f1-score support

Valid 0.00 0.00 0.00 15

Invalid 0.48 1.00 0.65 14

accuracy 0.48 29

macro avg 0.24 0.50 0.33 29

weighted avg 0.23 0.48 0.31 29