**بدون وزن های ImageNet**

import tensorflow as tf

import os

import cv2

from tqdm import tqdm

import numpy as np

from sklearn import preprocessing

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

from tensorflow.keras.models import Sequential

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

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score, confusion\_matrix

from matplotlib.backends.backend\_pdf import PdfPages

# Define the AlexNet model

def AlexNet(input\_shape, num\_classes):

    model = Sequential([

        # Layer 1: Convolutional Layer

        tf.keras.layers.Conv2D(96, (11, 11), strides=(4, 4), activation='relu', input\_shape=input\_shape),

        tf.keras.layers.MaxPooling2D((3, 3), strides=(2, 2)),

        # Layer 2: Convolutional Layer

        tf.keras.layers.Conv2D(256, (5, 5), padding="same", activation='relu'),

        tf.keras.layers.MaxPooling2D((3, 3), strides=(2, 2)),

        # Layer 3: Convolutional Layer

        tf.keras.layers.Conv2D(384, (3, 3), padding="same", activation='relu'),

        # Layer 4: Convolutional Layer

        tf.keras.layers.Conv2D(384, (3, 3), padding="same", activation='relu'),

        # Layer 5: Convolutional Layer

        tf.keras.layers.Conv2D(256, (3, 3), padding="same", activation='relu'),

        tf.keras.layers.MaxPooling2D((3, 3), strides=(2, 2)),

        # Flatten the output of the previous layer

        tf.keras.layers.Flatten(),

        # Layer 6: Fully Connected Layer

        tf.keras.layers.Dense(4096, activation='relu'),

        # Layer 7: Fully Connected Layer

        tf.keras.layers.Dense(4096, activation='relu'),

        # Layer 8: Output Layer for Binary Classification

        tf.keras.layers.Dense(num\_classes, activation='sigmoid')  # Use 'sigmoid' for binary classification

    ])

    return model

# Download the data

#!gdown 13ndWKp\_qe5aD1LCOEkdVf\_tkvLCMBPRJ       # For 3316 samples

#!gdown 1FiEdQcfa3W\_8ZrLExftnrVNGY4e-GC8I       # For 3316 non\_emoj samples

#!gdown 1qbaV7ZW6ccZDnK6lN\_qHQYgkr8tTn1w0       # For 167290 samples Compressed     Dataset\_C

#!gdown 14ZLbnCYAmEnK\_0xPlVvKL\_-adilqxoX9       # For 83644 samples Compressed      Dataset\_C\_H

#!gdown 11YiYReHcSeyVoe6I8fIHK-\_RJwYp2cgK       # For 41822 samples Compressed      Dataset\_C\_Q

#!gdown 1\_SLbu\_FqCD5GVsFjPGaMm-54Li3npSgV       # For 41822 samples Compressed      Dataset\_C\_E

!gdown 1VhpWHMQaYqecBmd2QTKMFIY0m\_9hbBy1       # For 41822 samples Compressed      Dataset\_C\_E\_non\_emoj

import zipfile

file\_name = "/content/Dataset\_C\_E\_non\_emoj.zip"

with zipfile.ZipFile(file\_name, 'r') as zip\_ref:

    zip\_ref.extractall()

    print('Done')

data\_dir\_class1 = '/content/0'  # Path to the directory containing class 1 images

data\_dir\_class2 = '/content/1'  # Path to the directory containing class 2 images

X = []

Y = []

# Load images from class 1 directory

for i in tqdm(os.listdir(data\_dir\_class1)):

    img = cv2.imread(os.path.join(data\_dir\_class1, i), cv2.IMREAD\_GRAYSCALE)  # Load as grayscale

    img = cv2.resize(img, (224, 224))

    X.append(img)

    Y.append(0)  # Assign label 0 for class 1

# Load images from class 2 directory

for i in tqdm(os.listdir(data\_dir\_class2)):

    img = cv2.imread(os.path.join(data\_dir\_class2, i), cv2.IMREAD\_GRAYSCALE)  # Load as grayscale

    img = cv2.resize(img, (224, 224))

    X.append(img)

    Y.append(1)  # Assign label 1 for class 2

le = preprocessing.LabelEncoder()

Y = le.fit\_transform(Y)

Y = tf.keras.utils.to\_categorical(Y, num\_classes=2)

X = np.array(X).reshape(-1, 224, 224, 1)  # Reshape to (num\_samples, height, width, channels)

Y = np.array(Y)

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X, Y, test\_size=0.25, random\_state=42)

# Define the input shape and number of classes for binary classification (2 classes).

input\_shape = (224, 224, 1)  # Grayscale image has one channel

num\_classes = 2

# Create the model

model = AlexNet(input\_shape, num\_classes)

# Compile the model

model.compile(optimizer=tf.keras.optimizers.SGD(learning\_rate=1e-6, momentum=0.9), loss='binary\_crossentropy', metrics=['accuracy'])

# Print the model summary

model.summary()

# Train the model

history = model.fit(X\_train, Y\_train, validation\_data=(X\_test, Y\_test),

                    batch\_size=16, epochs=100, verbose=1)

# Calculate predictions

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

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

# Convert one-hot encoded labels back to integers

Y\_test = np.argmax(Y\_test, axis=1)

# Calculate evaluation metrics

accuracy = accuracy\_score(Y\_test, y\_pred)

precision = precision\_score(Y\_test, y\_pred)

recall = recall\_score(Y\_test, y\_pred)

f1 = f1\_score(Y\_test, y\_pred)

confusion\_mat = confusion\_matrix(Y\_test, y\_pred)

print("Accuracy:", accuracy)

print("Precision:", precision)

print("Recall:", recall)

print("F1 score:", f1)

print("Confusion Matrix:")

print(confusion\_mat)

# Create a PDF file for saving the plots and the confusion matrix

pdf\_file = "/content/output\_plots.pdf"

pdf\_pages = PdfPages(pdf\_file)

# Plot Confusion Matrix

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

sns.heatmap(confusion\_mat, annot=True, fmt=".0f", linewidths=0.5, square=True, cmap="Blues")

plt.xlabel('Predicted Label')

plt.ylabel('True Label')

plt.title('Confusion Matrix')

pdf\_pages.savefig()  # Save the Confusion Matrix plot to PDF

plt.close()

# Plot training and validation accuracy

plt.plot(history.history['accuracy'])

plt.plot(history.history['val\_accuracy'])

plt.title('Model Accuracy')

plt.xlabel('Epoch')

plt.ylabel('Accuracy')

plt.legend(['Train', 'Validation'], loc='upper left')

pdf\_pages.savefig()  # Save the Accuracy Plot to PDF

plt.close()

# Plot training and validation loss

plt.plot(history.history['loss'])

plt.plot(history.history['val\_loss'])

plt.title('Model Loss')

plt.xlabel('Epoch')

plt.ylabel('Loss')

plt.legend(['Train', 'Validation'], loc='upper left')

pdf\_pages.savefig()  # Save the Loss Plot to PDF

plt.close()

# Save the model weights

# model.save\_weights("/content/model\_weights.h5")

# Close the PDF file

pdf\_pages.close()

**با وزن های ImageNet در pytorch مشکل دارد**

import zipfile

import os

import torch

import torch.nn as nn

import torch.optim as optim

import torchvision.models as models

import torchvision.transforms as transforms

from torch.utils.data import DataLoader, Dataset, Subset

from PIL import Image

import numpy as np

import matplotlib.pyplot as plt

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score, confusion\_matrix

# Download and extract the dataset zip file

!gdown 1VhpWHMQaYqecBmd2QTKMFIY0m\_9hbBy1  # Dataset\_C\_E\_non\_emoj.zip

file\_name = "/content/Dataset\_C\_E\_non\_emoj.zip"

with zipfile.ZipFile(file\_name, 'r') as zip\_ref:

    zip\_ref.extractall()

    print('Done')

# Define the image transformations (resize and convert to tensor)

data\_transform = transforms.Compose([

    transforms.Resize((224, 224)),  # Resize images to the input size expected by AlexNet (224x224)

    transforms.ToTensor(),  # Convert PIL images to tensors

])

# Define a custom dataset class

class CustomDataset(Dataset):

    def \_\_init\_\_(self, root, transform=None):

        self.root = root

        self.transform = transform

        self.image\_names = os.listdir(self.root)

    def \_\_len\_\_(self):

        return len(self.image\_names)

    def \_\_getitem\_\_(self, idx):

        image\_name = self.image\_names[idx]

        image\_path = os.path.join(self.root, image\_name)

        image = Image.open(image\_path)

        # Get the class label from the first character of the image name

        class\_label = int(image\_name[0])

        if self.transform:

            image = self.transform(image)

        return image, class\_label

# Load the dataset for class 1 using the custom dataset class

class1\_folder = '/content/0'  # Path to the directory containing class 1 images

class1\_dataset = CustomDataset(root=class1\_folder, transform=data\_transform)

# Use the previously loaded class 2 dataset

class2\_dataset = CustomDataset(root='/content/1', transform=data\_transform)

# Combine the datasets into a single dataset

dataset = torch.utils.data.ConcatDataset([class1\_dataset, class2\_dataset])

# Calculate the train-test split

train\_ratio = 0.8

num\_samples = len(dataset)

num\_samples\_class1 = len(class1\_dataset)

num\_samples\_class2 = len(class2\_dataset)

train\_size\_class1 = int(train\_ratio \* num\_samples\_class1)

train\_size\_class2 = int(train\_ratio \* num\_samples\_class2)

# Create Subsets for train and test datasets

train\_dataset = Subset(dataset, list(range(train\_size\_class1)) + list(range(num\_samples\_class1, num\_samples\_class1 + train\_size\_class2)))

test\_dataset = Subset(dataset, list(range(train\_size\_class1, num\_samples\_class1)) + list(range(num\_samples\_class1 + train\_size\_class2, num\_samples)))

# Create DataLoaders for training and test sets

batch\_size = 32

train\_loader = DataLoader(

    train\_dataset,

    batch\_size=batch\_size,

    shuffle=True

)

test\_loader = DataLoader(

    test\_dataset,

    batch\_size=batch\_size,

    shuffle=False

)

# Load the pre-trained AlexNet model

alexnet = models.alexnet(pretrained=True)

# Freeze the feature extractor layers so that they are not trained

for param in alexnet.parameters():

    param.requires\_grad = False

# Modify the classifier for binary classification (two classes)

num\_classes = 2

alexnet.classifier[6] = nn.Linear(4096, num\_classes)

# Move the model to the GPU if available

device = torch.device("cuda" if torch.cuda.is\_available() else "cpu")

alexnet.to(device)

# Define loss function and optimizer

criterion = nn.CrossEntropyLoss()

optimizer = optim.Adam(alexnet.parameters(), lr=0.001)

# Train the model

num\_epochs = 10

for epoch in range(num\_epochs):

    total\_train\_correct = 0

    total\_train\_loss = 0.0

    for inputs, labels in train\_loader:

        inputs, labels = inputs.to(device), labels.to(device)

        optimizer.zero\_grad()

        outputs = alexnet(inputs)

        loss = criterion(outputs, labels)

        loss.backward(retain\_graph=True)  # Add retain\_graph=True to retain the computation graph

        optimizer.step()

        total\_train\_correct += (torch.argmax(outputs, dim=1) == labels).sum().item()

        total\_train\_loss += loss.item() \* inputs.size(0)

        loss.backward()

        optimizer.step()

    train\_accuracy = total\_train\_correct / len(train\_dataset)

    train\_loss = total\_train\_loss / len(train\_dataset)

    print(f"Epoch [{epoch + 1}/{num\_epochs}], Train Loss: {train\_loss:.4f}, Train Accuracy: {train\_accuracy:.4f}")

    train\_accuracy\_list.append(train\_accuracy)

    train\_loss\_list.append(train\_loss)

    # Calculate the final accuracy and loss after training

final\_train\_accuracy = total\_train\_correct / len(train\_dataset)

final\_train\_loss = total\_train\_loss / len(train\_dataset)

print(f"Final Train Accuracy: {final\_train\_accuracy:.4f}")

print(f"Final Train Loss: {final\_train\_loss:.4f}")

# Test the model

alexnet.eval()

test\_correct = 0

predicted\_labels = []

true\_labels = []

with torch.no\_grad():

    for inputs, labels in test\_loader:

        inputs, labels = inputs.to(device), labels.to(device)

        outputs = alexnet(inputs)

        \_, predicted = torch.max(outputs, 1)

        test\_correct += (predicted == labels).sum().item()

        predicted\_labels.extend(predicted.cpu().numpy())

        true\_labels.extend(labels.cpu().numpy())

test\_accuracy = test\_correct / len(test\_dataset)

# Calculate other performance metrics

precision = precision\_score(true\_labels, predicted\_labels)

recall = recall\_score(true\_labels, predicted\_labels)

f1 = f1\_score(true\_labels, predicted\_labels)

# Print performance metrics

print(f"Test Accuracy: {test\_accuracy:.4f}")

print(f"Precision: {precision:.4f}, Recall: {recall:.4f}, F1-score: {f1:.4f}")

# Confusion matrix

conf\_matrix = confusion\_matrix(true\_labels, predicted\_labels)

print("Confusion Matrix:")

print(conf\_matrix)

# Plot and save the accuracy and loss charts

plt.figure()

plt.plot(train\_accuracy\_list, label='Train Accuracy')

plt.plot(train\_loss\_list, label='Train Loss')

plt.xlabel('Epoch')

plt.legend()

plt.savefig('accuracy\_loss.pdf')

plt.show()

# Plot and save the confusion matrix

plt.figure()

plt.imshow(conf\_matrix, cmap='Blues', interpolation='nearest')

plt.colorbar()

plt.xlabel('Predicted Label')

plt.ylabel('True Label')

plt.title('Confusion Matrix')

plt.savefig('confusion\_matrix.pdf')

plt.show()