به نام خدا

دانشگاه تهران

دانشکده مهندسی برق و کامپیوتر

درس شبکه‌های عصبی و یادگیری عمیق

تمرین دوم

نام و نام خانوادگی	**محمد پویا افشاری – علیرضا اسمعیل زاده**
شماره دانشجویی	**810198351-810198577**
تاریخ ارسال گزارش	**1402.09.03**

فهرست

شکل‌ها

پاسخ ۱ – تجزیه و تحلیل احساسات صورت مبتنی بر CNN

۱-۱-۱. وظیفه

در این مساله باید به کمک مدل CNN طراحی شده در مقاله با عنوان:

CNN-based Facial Affect Analysis on Mobile Devices

که مدلی برای تجزیه و تحلیل احساسات صورت در دستگاه های تلفن همراه ارائه می‌کند را پیاده سازی کنیم.

برخلاف رویکردهای سنتی **CNN**، مدل های مستقر در دستگاه های تلفن همراه باید نیازهای ذخیره سازی را به حداقل برسانند و در عین حال عملکرد بالا را حفظ کنند. بنابراین برای دستیابی به این امر ۳ معماری متفاوت **CNN** در این مقاله ارائه شده است. نتایج به دست آمده نشان میدهد که معماری های پیشنهادی عملکرد مشابهی را نسبت به آخرین مدل های پیشرو در این زمینه دارند و این در حالی است که نیازهای ذخیره سازی را به حداقل میرسانند. معماری های مورد استفاده در این مقاله عبارتند از **VGGNet**، **AlexNet** و **MobileNet**

۱-۱-۲. جمع آوری دادگان و پیش پردازش

از اصلی ترین بخش‌های کار روی این تمرین بخش پیش پردازش داده‌ها خواهد. در ابتدا نیاز است داده ها را از روی گوگل درایو به محیط کولب mount کنیم پس از دسترسی به داده ها و و سپس نیاز است تا داده ها به مجموعه های آموزش و تست تقسیم شوند.

۱-۱-۲-۱. اضافه کردن کتابخانه‌ها

برای حل تمرین و ساخت شبکه CNN میتوان هر یک از کتابخانه‌های Pythorch یا Kerberos را استفاده کرد در اینجا ما از Tensorflow و بخصوص Kerberos استفاده کردیم.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import cv2
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from tensorflow import keras
from tensorflow.keras import layers, models
from tensorflow.keras.models import Sequential
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, BatchNormalization, Dropout
import os
import seaborn as sns
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.utils import to_categorical

from google.colab import drive
```

۲-۲-۱-۱. جمع آوری داده ها

برای افزودن داده ها به این روش ابتدا داده‌ها را در درایو ریخته از انجا import می‌کنیم در Colab:

```
drive.flush_and_unmount()
drive.mount('/content/drive')
```

```
Drive not mounted, so nothing to flush and unmount.
Mounted at /content/drive
```

سپس فایل rar داده ها را unrar کنیم

```
drive_path = '/content/drive/MyDrive/HW2Q1-resources/'
os.chdir(drive_path)


!pwd

!unrar x NN_HW2_Face_Emotion.rar
```

```
Streaming output truncated to the last 5000 lines.
Extracting  TRAIN/fear/image0000694.jpg                   24%  OK
Extracting  TRAIN/fear/image0000701.jpg                   24%  OK
Extracting  TRAIN/fear/image0000777.jpg                   24%  OK
Extracting  TRAIN/fear/image0000808.jpg                   24%  OK
Extracting  TRAIN/fear/image0000815.jpg                   24%  OK
Extracting  TRAIN/fear/image0000842.jpg                   24%  OK
Extracting  TRAIN/fear/image0000843.jpg                   24%  OK
Extracting  TRAIN/fear/image0000915.jpg                   24%  OK
Extracting  TRAIN/fear/image0001001.jpg                   24%  OK
Extracting  TRAIN/fear/image0001038.jpg                   24%  OK
Extracting  TRAIN/fear/image0001247.jpg                   24%  OK
Extracting  TRAIN/fear/image0001286.jpg                   24%  OK
Extracting  TRAIN/fear/image0001316.jpg                   24%  OK
Extracting  TRAIN/fear/image0001454.jpg                   24%  OK
Extracting  TRAIN/fear/image0001458.jpg                   24%  OK
Extracting  TRAIN/fear/image0001651.jpg                   24%  OK
Extracting  TRAIN/fear/image0001685.jpg                   24%  OK
Extracting  TRAIN/fear/image0001977.jpg                   24%  OK
Extracting  TRAIN/fear/image0001991.jpg                   24%  OK
Extracting  TRAIN/fear/image0002031.jpg                   24%  OK
Extracting  TRAIN/fear/image0002217.jpg                   24%  OK
Extracting  TRAIN/fear/image0002309.jpg                   24%  OK
Extracting  TRAIN/fear/image0002352.jpg                   24%  OK
Extracting  TRAIN/fear/image0002354.jpg                   24%  OK
...
Extracting  TRAIN/surprise/image0011801.jpg               99%  OK
Extracting  TRAIN/surprise/image0011802.jpg               99%  OK
Extracting  TRAIN/surprise/image0011824.jpg               99%  OK
All OK
```

۲-۲-۱-۳. تکثیر داده ها Data Augment

پس از نرمال سازی بین ۰ و ۱ جهت تعمیم پذیری مدل نیاز است تا نیاز به تولید تصاویر جدید از روی داده ها وجود دارید که با حالت های چرخش تصاویر تا ۲۰ درجه، translation تا 10% و چرخش در جهت x است:

```
dataset_path = '/content/drive/MyDrive/HW2Q1-resources/'
```

```
train_path = os.path.join(dataset_path, 'TRAIN')
tune_path = os.path.join(dataset_path, 'TUNE')
class_mapping = {'anger': 0, 'contempt': 1, 'disgust': 2, 'fear': 3, 'happy': 4, 'neutral': 5, 'sad': 6, 'surprise': 7}

def load_and_preprocess_images(path, num_samples, class_mapping):
    images = []
    labels = []

    datagen = ImageDataGenerator(
        rotation_range=20,
        width_shift_range=0.1,
        height_shift_range=0.1,
        horizontal_flip=True
    )

    class_names = os.listdir(path)

    for class_name in class_names:
        class_path = os.path.join(path, class_name)
        class_images = os.listdir(class_path)[:num_samples]
        print(f"Class: {class_name}, Number of samples loaded: {len(class_images)}")

        for image_name in class_images:
            image_path = os.path.join(class_path, image_name)
            image = cv2.imread(image_path)
            image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
            image = cv2.resize(image, (128, 128))
            image = image / 255.0
            # Apply data augmentation
            img_array = image.reshape((1,) + image.shape)
            for batch in datagen.flow(img_array, batch_size=1):
                augmented_image = batch[0]
                images.append(augmented_image)
                labels.append(class_mapping[class_name])
                break

            images.append(image)
            labels.append(class_mapping[class_name])

    return np.array(images), np.array(labels)
```

سپس به تعداد ۱۲۵ نمونه آموزشی از برای هر کلاس لود می کنیم

```
num_train_samples = 125 # load 125 from each category - > 1000 total samples
train_images, train_labels = load_and_preprocess_images(train_path, num_train_samples, class_mapping)
```

```
Class: contempt, Number of samples loaded: 125
Class: disgust, Number of samples loaded: 125
Class: fear, Number of samples loaded: 125
Class: happy, Number of samples loaded: 125
Class: neutral, Number of samples loaded: 125
Class: sad, Number of samples loaded: 125
```

```
Class: surprise, Number of samples loaded: 125
Class: anger, Number of samples loaded: 125
```

سپس به تعداد ۲۵ نمونه تیون شده از برای هر کلاس لود می کنیم

```
num_tune_samples = 25 # load 125 from each category - > 200 total samples
tune_images, tune_labels = load_and_preprocess_images(tune_path, num_tune_samples, class_mapping)
```

```
Class: anger, Number of samples loaded: 25
Class: contempt, Number of samples loaded: 25
Class: disgust, Number of samples loaded: 25
Class: fear, Number of samples loaded: 25
Class: happy, Number of samples loaded: 25
Class: neutral, Number of samples loaded: 25
Class: sad, Number of samples loaded: 25
Class: surprise, Number of samples loaded: 25
```

سپس داده ها را به داده های آموزشی وتستی تقسیم می کنیم

```
train_images, test_images, train_labels, test_labels = train_test_split(
    train_images, train_labels, test_size=0.4, random_state=42
)
```

```
print("Train Images Shape:", train_images.shape)
print("Train Labels Shape:", train_labels.shape)
print("Test Images Shape:", test_images.shape)
print("Test Labels Shape:", test_labels.shape)
print("Validation Images Shape:", tune_images.shape)
print("Validation Labels Shape:", tune_labels.shape)
```

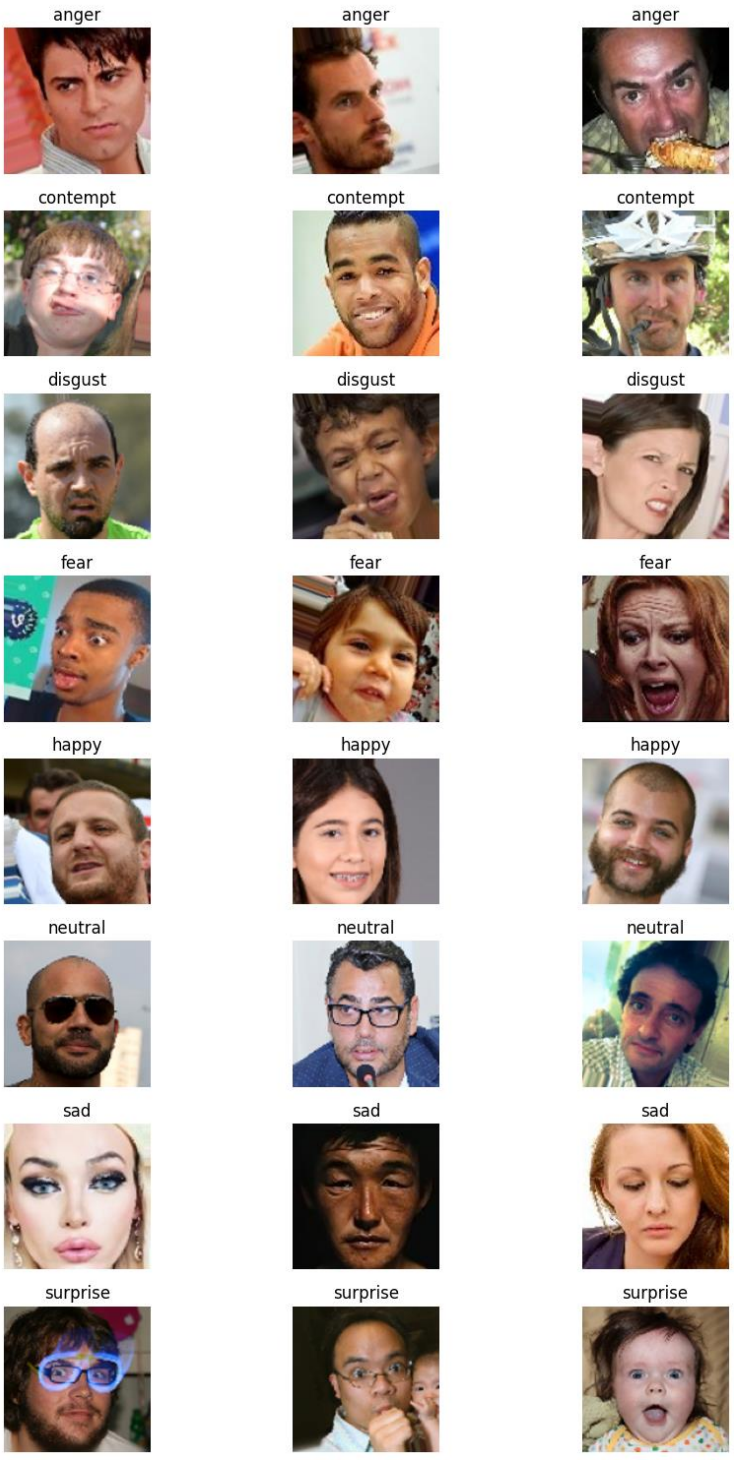
```
Train Images Shape: (1200, 128, 128, 3)
Train Labels Shape: (1200,)
Test Images Shape: (800, 128, 128, 3)
Test Labels Shape: (800,)
Validation Images Shape: (400, 128, 128, 3)
Validation Labels Shape: (400,)
```

۴-۲-۱-۱. نمایش داده‌ها

در این بخش چند عکس رندم از احساسات صورت افراد نمایش داده می‌شود.

```
def display_sample_images(images, labels, class_mapping):
    num_classes = len(class_mapping)
    fig, axes = plt.subplots(nrows=num_classes, ncols=3, figsize=(10, 15))

    for i, class_name in enumerate(class_mapping.keys()):
        class_indices = np.where(labels == class_mapping[class_name])[0]
        sample_indices = np.random.choice(class_indices, size=3, replace=False)

        for j, sample_idx in enumerate(sample_indices):
            axes[i, j].imshow(images[sample_idx])
            axes[i, j].set_title(class_name)
            axes[i, j].axis('off')

    plt.tight_layout()
    plt.show()
```

```
display_sample_images(train_images, train_labels, class_mapping)
```



شکل 1. نمایش نمونه‌های کلاس‌های احساسی مختلف

۳-۱-۱. معماری و یادگیری مدل AlexNet

این معماری شامل مجموعه ای از هسته های کانولوشن با ابعاد کرنل رو به کاهش می باشد که از ۹×۹ شروع میشود و به ۳×۳ کاهش مییابد. جهت منظم سازی و آموزش سریعتر، هر بلوک کانولوشن از یک لایه کانولوشن دو بعدی معمولی و به دنبال آن یک لایه Batch-Normalization و یک لایه فعال سازی ReLU به همراه ۰.۲

Dropout برای MaxPooling و ۰.۵ Dropout برای لایه های Dense ساخته شده است. معماری همچنین کم عمق تر و باریکت ر از طرح اصلی AlexNet تا اندازه مدل را به حداقل برساند.

Type	Shape	Output
Conv	$9 \times 9 \times 16$	$128 \times 128 \times 16$
MaxPool	2×2	$64 \times 64 \times 16$
Conv	$7 \times 7 \times 32$	$64 \times 64 \times 32$
MaxPool	2×2	$32 \times 32 \times 32$
Conv	$5 \times 5 \times 64$	$32 \times 32 \times 64$
MaxPool	2×2	$16 \times 16 \times 64$
Conv	$3 \times 3 \times 128$	$16 \times 16 \times 128$
MaxPool	2×2	$8 \times 8 \times 128$
Conv	$3 \times 3 \times 128$	$8 \times 8 \times 128$
MaxPool	2×2	$4 \times 4 \times 128$
Flatten	2048	–
2×Dense	1024	–
Dense	8 or 2	1 label or 2 floats

شکل 2. معماری **AlexNet**

```
def create_affectnet_cnn(input_shape=(128, 128, 3), num_classes=8):
    model = Sequential()

    # Convolutional Block 1
    model.add(Conv2D(16, (9, 9), activation='relu', input_shape=input_shape, name='conv2d_input'))
    model.add(BatchNormalization())
    model.add(MaxPooling2D((2, 2)))
    model.add(Dropout(0.2))

    # Convolutional Block 2
    model.add(Conv2D(32, (7, 7), activation='relu'))
    model.add(BatchNormalization())
    model.add(MaxPooling2D((2, 2)))
    model.add(Dropout(0.2))

    # Convolutional Block 3
    model.add(Conv2D(64, (5, 5), activation='relu'))
    model.add(BatchNormalization())
    model.add(MaxPooling2D((2, 2)))
    model.add(Dropout(0.2))

    # Convolutional Block 4
    model.add(Conv2D(128, (3, 3), activation='relu'))
    model.add(BatchNormalization())
    model.add(MaxPooling2D((2, 2)))
    model.add(Dropout(0.2))

    # Convolutional Block 5
```

```
    model.add(Conv2D(128, (3, 3), activation='relu'))
    model.add(BatchNormalization())
    model.add(MaxPooling2D((2, 2)))
    model.add(Dropout(0.2))

    # Flatten
    model.add(Flatten())

    # Fully Connected (Dense) Layers
    model.add(Dense(1024, activation='relu'))
    model.add(Dropout(0.5))

    # Fully Connected (Dense) Layers
    model.add(Dense(1024, activation='relu'))
    model.add(Dropout(0.5))

    # Output Layer for Emotion Classification
    model.add(Dense(num_classes, activation='softmax', name='emotion_output'))

    return model
```

۱-۳-۱-۱. یادگیری مدل AlexNet

ابتدا لیبل ها را به صورت one_hot تبدیل میکنیم سپس مدل را با استفاده از optimizer Adam و نرخ یادگیری 0.001 با ۲۴ ایپاک و batch_size ۳۲ آموزش میدهیم.

```
train_labels_one_hot = to_categorical(train_labels, num_classes=8)
test_labels_one_hot = to_categorical(test_labels,num_classes=8)
```

```
model = create_affectnet_cnn(input_shape=(128, 128, 3), num_classes=8)
```

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

```
history = model.fit(train_images, train_labels_one_hot,
                    epochs=24,
                    batch_size=32,
                    validation_split=0.2)
```

```
Epoch 1/24
30/30 [==============================] - 51s 2s/step - loss: 1.8785 - accuracy: 0.2979 - val_loss:
2.1042 - val_accuracy: 0.2083
Epoch 2/24
30/30 [==============================] - 52s 2s/step - loss: 1.8509 - accuracy: 0.2937 - val_loss:
2.0671 - val_accuracy: 0.1958
Epoch 3/24
30/30 [==============================] - 51s 2s/step - loss: 1.7488 - accuracy: 0.3292 - val_loss:
2.0323 - val_accuracy: 0.2333
Epoch 4/24
30/30 [==============================] - 51s 2s/step - loss: 1.6598 - accuracy: 0.3490 - val_loss:
1.9497 - val_accuracy: 0.2167
```

```
Epoch 5/24
30/30 [==============================] - 53s 2s/step - loss: 1.6301 - accuracy: 0.3604 - val_loss:
1.8715 - val_accuracy: 0.2542
Epoch 6/24
30/30 [==============================] - 54s 2s/step - loss: 1.5643 - accuracy: 0.3823 - val_loss:
2.2361 - val_accuracy: 0.2500
Epoch 7/24
30/30 [==============================] - 51s 2s/step - loss: 1.4873 - accuracy: 0.4354 - val_loss:
1.7679 - val_accuracy: 0.3083
Epoch 8/24
30/30 [==============================] - 52s 2s/step - loss: 1.4086 - accuracy: 0.4625 - val_loss:
2.0949 - val_accuracy: 0.3000
Epoch 9/24
30/30 [==============================] - 58s 2s/step - loss: 1.3994 - accuracy: 0.4698 - val_loss:
1.8646 - val_accuracy: 0.2542
Epoch 10/24
30/30 [==============================] - 51s 2s/step - loss: 1.3721 - accuracy: 0.4833 - val_loss:
1.7424 - val_accuracy: 0.3417
Epoch 11/24
30/30 [==============================] - 52s 2s/step - loss: 1.2384 - accuracy: 0.5323 - val_loss:
2.1958 - val_accuracy: 0.2500
Epoch 12/24
30/30 [==============================] - 51s 2s/step - loss: 1.1936 - accuracy: 0.5562 - val_loss:
1.7993 - val_accuracy: 0.3042
Epoch 13/24
...
Epoch 23/24
30/30 [==============================] - 52s 2s/step - loss: 0.6275 - accuracy: 0.7896 - val_loss:
2.3294 - val_accuracy: 0.3417
Epoch 24/24
30/30 [==============================] - 51s 2s/step - loss: 0.5541 - accuracy: 0.7990 - val_loss:
2.2889 - val_accuracy: 0.3458
```

ذخیره سازی و لود کردن مدل در فایل h5

```
model.save('variantAlexNet.h5')
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3079 UserWarning: You are saving your model as an HDF5 file via `model.save()`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my_model.keras')`.
  saving_api.save_model(
```

```
from keras.models import load_model

loaded_model = load_model('variantAlexNet.h5')
```

۴-۱-۱. ارزیابی مدل

۱-۴-۱-۱. نمودار Loss و Accuracy

ترسیم نمودار Loss و Accuracy

```
# Plot training loss
plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epoch')
```

```
plt.ylabel('Loss')
plt.legend()

# Plot training accuracy
plt.subplot(1, 2, 2)
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

plt.show()
```

شكل 3. نمودار دقت و خطای مدل **AlexNet**

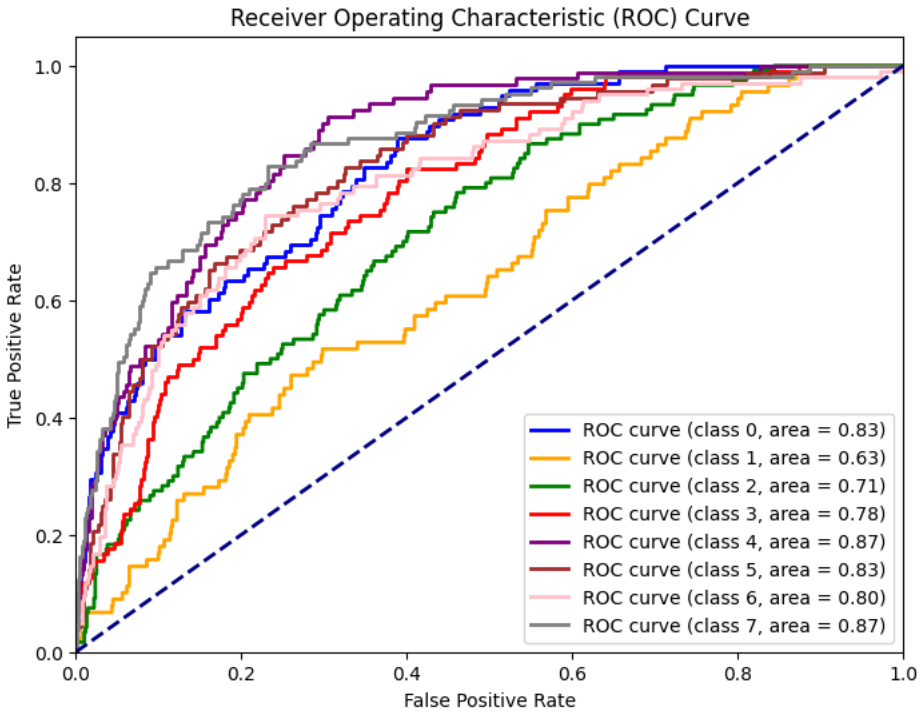
بر طبق نمودارهاي Loss و Accuracy مي‌بينيم كه داده نوسان زيادي روي داده‌هاي Validation دارد و دقت آن روي حدود ايپاک 20 تا حد Maximum خود رسيده و Loss آن كمينه شده است اين داده به ما كمك ميكند كه نسل مناسب مدل خود را براي داده تست انتخاب كنيم.

۲-۴-۱-۱. نمودار ROC

ترسيم نمودار ROC

```
from sklearn.metrics import roc_curve, auc

y_score = model.predict(test_images)

fpr = dict()
tpr = dict()
roc_auc = dict()

num_classes = test_labels_one_hot.shape[1]

for i in range(num_classes):
    fpr[i], tpr[i], _ = roc_curve(test_labels_one_hot[:, i], y_score[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

plt.figure(figsize=(8, 6))
colors = ['blue', 'orange', 'green', 'red', 'purple', 'brown', 'pink', 'gray']
```

```
for i in range(num_classes):
    plt.plot(fpr[i], tpr[i], color=colors[i], lw=2,
             label=f'ROC curve (class {i}, area = {roc_auc[i]:.2f})')

plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc="lower right")
plt.show()
```



شکل 5. نمودار **ROC** مدل **AlexNet** برای کلاس‌های مختلف

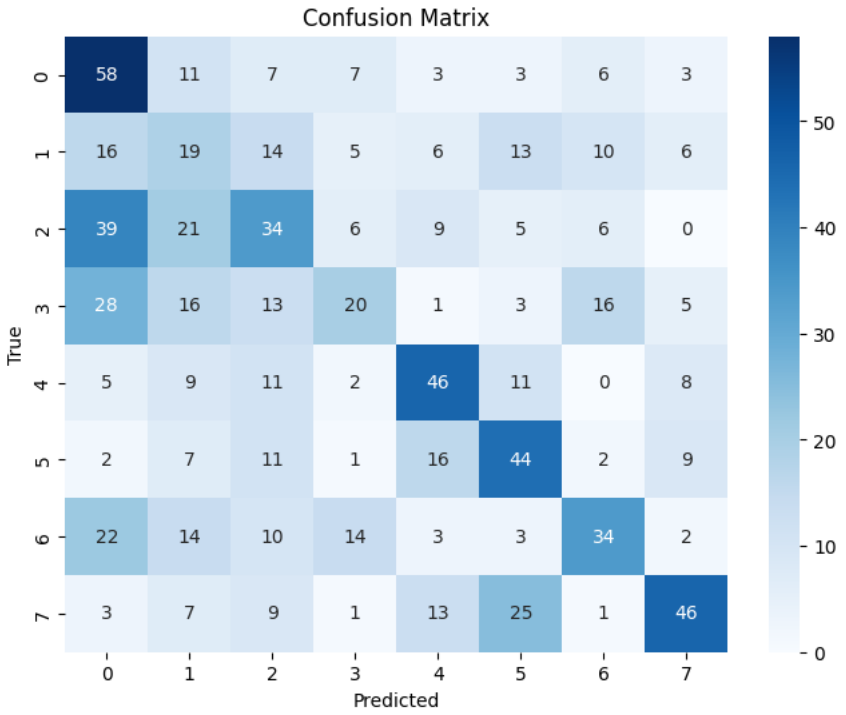
منحنی ROC (منحنی مشخصه عملکرد گیرنده) نموداری است که عملکرد یک مدل classification را در تمام آستانه‌های classification نشان می‌دهد. این منحنی دو پارامتر را ترسیم می کند: نرخ مثبت واقعی. نرخ مثبت کاذب هر چه این منحنی به محور چپ و بالا نزدیک تر باشد عملکرد بهتری دارد در اینجا بهترین عملکرد برای happy و surprise و بدترین عملکرد برای contempt بوده است.

۳-۴-۱-۱. مقادیر Recall، Precision، F1 و ماتریس Confusion

```
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score

y_pred = model.predict(test_images)
y_pred_classes = np.argmax(y_pred, axis=1)
y_true = np.argmax(test_labels_one_hot, axis=1)

conf_mat = confusion_matrix(y_true, y_pred_classes)
```

```
plt.figure(figsize=(8, 6))
sns.heatmap(conf_mat, annot=True, fmt='d', cmap='Blues',
            xticklabels=range(num_classes),
            yticklabels=range(num_classes))
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.show()

print(classification_report(y_true, y_pred_classes, target_names=[f'Class {i}' for i in range(num_classes)]))

accuracy = accuracy_score(y_true, y_pred_classes)
print(f'Accuracy: {accuracy:.2f}')
```

```
25/25 [==============================] - 8s 302ms/step
```



شکل 6. نمودار آشفتگی مدل **AlexNet**

```
                precision    recall  f1-score   support

     Class 0       0.34      0.59      0.43        98
     Class 1       0.18      0.21      0.20        89
     Class 2       0.31      0.28      0.30       120
     Class 3       0.36      0.20      0.25       102
     Class 4       0.47      0.50      0.49        92
     Class 5       0.41      0.48      0.44        92
     Class 6       0.45      0.33      0.38       102
     Class 7       0.58      0.44      0.50       105

    accuracy                           0.38       800
   macro avg       0.39      0.38      0.37       800
weighted avg       0.39      0.38      0.37       800

Accuracy: 0.38
```

پاسخ ۲ – پیاده سازی مدل VGGNet

۱-۲-۱. معماری و یادگیری مدل VGGNet

این معماری تقریباً شبیه طراحی الهام گرفته شده از AlexNet در قسمت قبل است، اگرچه از اصل پشت VGG16 از هسته‌های کانولوشن 3×3 انباشته برای گرفتن ساختار تصویر بزرگ‌تر استفاده می‌کند. بلوک های کانولوشن که برای مدل قبل توضیح داده شده است. بالا دوباره استفاده می شود، به لایه های max-pooling متصل شده است و به دنبال آن دو لایه fully connected قبل از لایه خروجی قرار می گیرند. همانطور که در بالا، هر لایه pooling با یک dropout گاوسی 0.2 متصل می شود و بعد از هر لایه dense یک 0.5 dropout وجود دارد ،همچنین باریک تر و کم عمق تر از پیاده سازی های معمولی VGGNet به منظور حفظ فضا ذخیره سازی است.

Type	Shape	Output
2×Conv	3 × 3 × 16	128 × 128 × 16
MaxPool	2 × 2	64 × 64 × 16
2×Conv	3 × 3 × 32	64 × 64 × 32
MaxPool	2 × 2	32 × 32 × 32
2×Conv	3 × 3 × 64	32 × 32 × 64
MaxPool	2 × 2	16 × 16 × 64
2×Conv	3 × 3 × 128	16 × 16 × 128
MaxPool	2 × 2	8 × 8 × 128
2×Conv	3 × 3 × 128	8 × 8 × 128
MaxPool	2 × 2	4 × 4 × 128
Flatten	2048	–
2×Dense	1024	–
Dense	8 or 2	1 label or 2 floats

شکل 7. معماری مدل **VGGNet**

```
def create_vggnet_variant(input_shape=(128, 128, 3), num_classes=8):
    model = Sequential()
     # Convolutional Block 1
    model.add(Conv2D(16, (3, 3), activation='relu', input_shape=input_shape, name='conv2d_input'))
    model.add(BatchNormalization())
    model.add(Conv2D(16, (3, 3), activation='relu', padding='same'))
    model.add(BatchNormalization())
    model.add(MaxPooling2D((2, 2)))
    model.add(Dropout(0.2))

    # Convolutional Block 2
```

```
model.add(Conv2D(32, (3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(Conv2D(32, (3, 3), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D((2, 2)))
model.add(Dropout(0.2))

# Convolutional Block 3
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(Conv2D(64, (3, 3), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D((2, 2)))
model.add(Dropout(0.2))

# Convolutional Block 4
model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(Conv2D(128, (3, 3), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D((2, 2)))
model.add(Dropout(0.2))

# Convolutional Block 5
model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(Conv2D(128, (3, 3), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D((2, 2)))
model.add(Dropout(0.2))

# Flatten
model.add(Flatten())

# Fully Connected (Dense) Layers
model.add(Dense(1024, activation='relu'))
model.add(Dropout(0.5))

# Fully Connected (Dense) Layers
model.add(Dense(1024, activation='relu'))
model.add(Dropout(0.5))

# Output Layer for Emotion Classification
model.add(Dense(num_classes, activation='softmax', name='emotion_output'))

return model
```

۱-۲-۲-۱. یادگیری مدل VGGNet

مدل را با استفاده از optimizer Adam و نرخ یادگیری 0.001 با ۲۴ ایپاک و batch_size ۳۲ آموزش می دهیم.

```
model_vggnet= create_vggnet_variant(input_shape=(128, 128, 3), num_classes=8)
```

```
model_vggnet.compile(optimizer='adam',
            loss='categorical_crossentropy',
            metrics=['accuracy'])
```

```
history_vggnet= model_vggnet.fit(train_images, train_labels_one_hot,
                  epochs=24,
                  batch_size=32,
                  validation_split=0.2)
```

```
Epoch 1/24
30/30 [==============================] - 57s 2s/step - loss: 3.0270 - accuracy: 0.1187 - val_loss:
2.0925 - val_accuracy: 0.1042
Epoch 2/24
30/30 [==============================] - 53s 2s/step - loss: 2.3616 - accuracy: 0.1521 - val_loss:
2.0923 - val_accuracy: 0.1083
Epoch 3/24
30/30 [==============================] - 56s 2s/step - loss: 2.1633 - accuracy: 0.1667 - val_loss:
2.0809 - val_accuracy: 0.1333
Epoch 4/24
30/30 [==============================] - 52s 2s/step - loss: 2.0479 - accuracy: 0.2156 - val_loss:
2.1791 - val_accuracy: 0.1500
Epoch 5/24
30/30 [==============================] - 53s 2s/step - loss: 1.9657 - accuracy: 0.2719 - val_loss:
2.1602 - val_accuracy: 0.1167
Epoch 6/24
30/30 [==============================] - 53s 2s/step - loss: 1.9020 - accuracy: 0.2750 - val_loss:
2.2771 - val_accuracy: 0.1458
Epoch 7/24
30/30 [==============================] - 59s 2s/step - loss: 1.8341 - accuracy: 0.2802 - val_loss:
2.6385 - val_accuracy: 0.1500
Epoch 8/24
30/30 [==============================] - 52s 2s/step - loss: 1.8080 - accuracy: 0.3156 - val_loss:
2.2335 - val_accuracy: 0.1333
Epoch 9/24
30/30 [==============================] - 53s 2s/step - loss: 1.6929 - accuracy: 0.3333 - val_loss:
2.1839 - val_accuracy: 0.1542
Epoch 10/24
30/30 [==============================] - 53s 2s/step - loss: 1.6854 - accuracy: 0.3469 - val_loss:
2.7083 - val_accuracy: 0.1500
Epoch 11/24
30/30 [==============================] - 53s 2s/step - loss: 1.6592 - accuracy: 0.3625 - val_loss:
2.8475 - val_accuracy: 0.1500
Epoch 12/24
30/30 [==============================] - 55s 2s/step - loss: 1.5490 - accuracy: 0.3969 - val_loss:
2.5716 - val_accuracy: 0.1792
Epoch 13/24
...
Epoch 23/24
30/30 [==============================] - 52s 2s/step - loss: 0.8638 - accuracy: 0.6698 - val_loss:
2.1099 - val_accuracy: 0.3250
Epoch 24/24
30/30 [==============================] - 52s 2s/step - loss: 0.7211 - accuracy: 0.7427 - val_loss:
2.6793 - val_accuracy: 0.2875
```

ذخیره سازی و لود کردن مدل در فایل h5

```
model.save('variantVGGNet.h5')
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3079: UserWarning: You are saving your model as an HDF5 file via `model.save()`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my_model.keras')`.
```

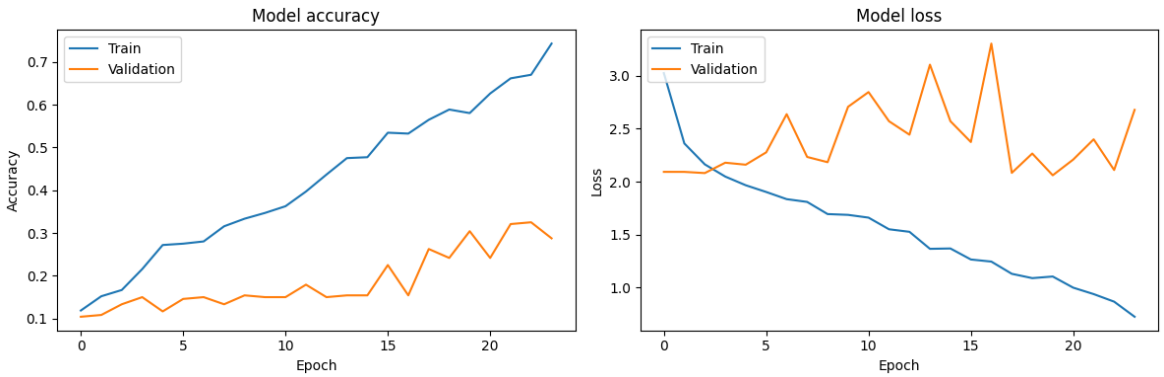
```
loaded_model = load_model('variantVGGNet.h5')
```

۳-۲-۱. ارزیابی مدل

۱-۳-۲-۱. نمودار Loss و Accuracy

ترسیم نمودار Loss و Accuracy

```
plt.figure(figsize=(12, 4))

plt.subplot(1, 2, 1)
plt.plot(history_vggnet.history['accuracy'])
plt.plot(history_vggnet.history['val_accuracy'])
plt.title('Model accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(['Train', 'Validation'], loc='upper left')

plt.subplot(1, 2, 2)
plt.plot(history_vggnet.history['loss'])
plt.plot(history_vggnet.history['val_loss'])
plt.title('Model loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend(['Train', 'Validation'], loc='upper left')

plt.tight_layout()
plt.show()
```



شکل 7. نمودارهای خطا و دقت مدل **VGGNet**

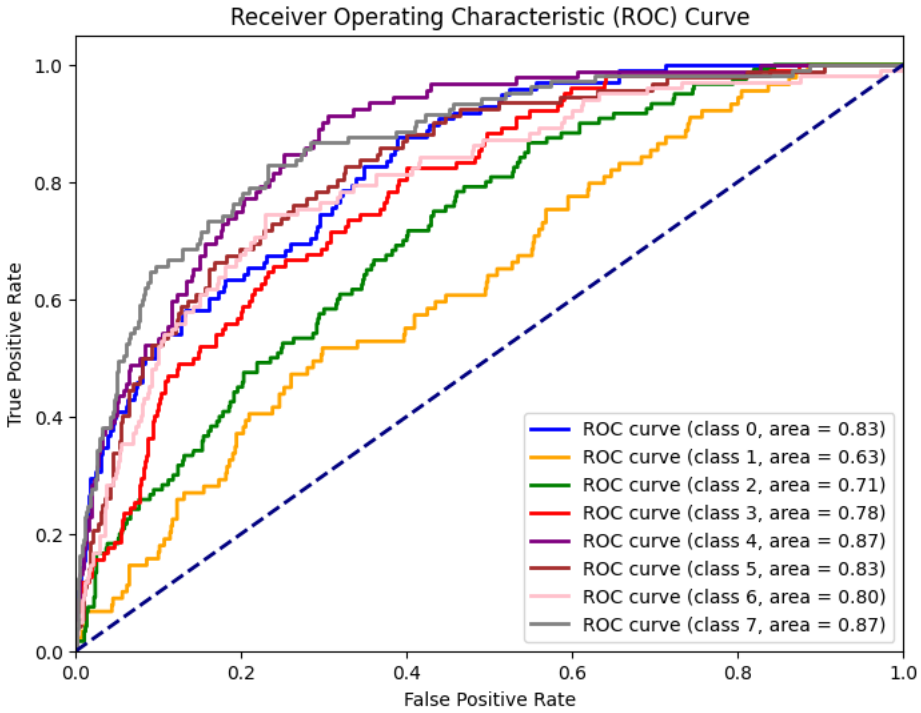
بر طبق نمودارهاي Loss و Accuracy مي‌بينيم كه داده نوسان زيادي روي داده‌هاي Validation دارد و دقت آن روي حدود ايپاک 22 تا حد Maximum خود رسيده و Loss آن كمينه شده است اين داده به ما كمك ميكند كه نسل مناسب مدل خود را براي داده تست انتخاب كنيم.

۲-۳-۲-۱. نمودار ROC

ترسیم نمودار ROC

```
y_score = loaded_model.predict(test_images)

num_classes = y_score.shape[1]

fpr = dict()
tpr = dict()
roc_auc = dict()

for i in range(num_classes):
    fpr[i], tpr[i], _ = roc_curve(test_labels_one_hot[:, i], y_score[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

plt.figure(figsize=(8, 6))
colors = ['blue', 'orange', 'green', 'red', 'purple', 'brown', 'pink', 'gray']

for i in range(num_classes):
    plt.plot(fpr[i], tpr[i], color=colors[i], lw=2,
             label=f'ROC curve (class {i}, area = {roc_auc[i]:.2f})')

plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc="lower right")
plt.show()
```

```
25/25 [==============================] - 10s 400ms/step
```



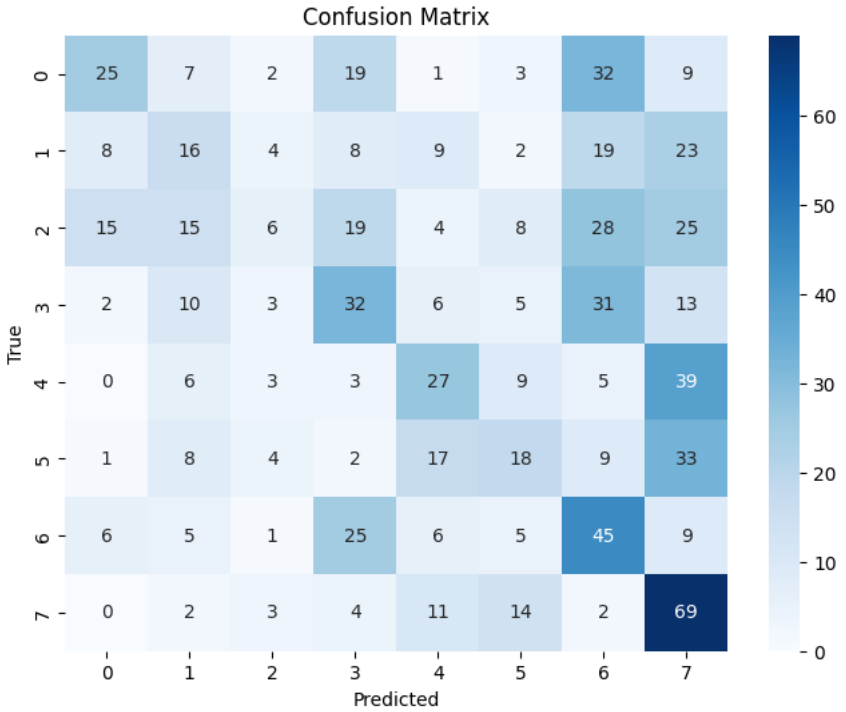
شکل 8. نمودار **ROC** به ازای کلاس‌های مختلف مدل **VGGNet**

منحنی ROC (منحنی مشخصه عملکرد گیرنده) نموداری است که عملکرد یک مدل classification را در تمام آستانه‌های classification نشان می‌دهد. این منحنی دو پارامتر را ترسیم می کند: نرخ مثبت واقعی. نرخ مثبت کاذب هر چه این منحنی به محور چپ و بالا نزدیک تر باشد عملکرد بهتری دارد در اینجا بهترین عملکرد برای happy و surprise و بدترین عملکرد برای contempt بوده است.

۳-۳-۲-۱. مقادیر Recall، Precision، F1 و ماتریس Confusion

```
y_pred = model_vggnet.predict(test_images)
y_pred_classes = np.argmax(y_pred, axis=1)
y_true = np.argmax(test_labels_one_hot, axis=1)

conf_mat = confusion_matrix(y_true, y_pred_classes)

plt.figure(figsize=(8, 6))
sns.heatmap(conf_mat, annot=True, fmt='d', cmap='Blues',
            xticklabels=range(num_classes),
            yticklabels=range(num_classes))
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.show()

class_report = classification_report(y_true, y_pred_classes, target_names=[f'Class {i}' for i in
range(num_classes)])
print("Classification Report:\n", class_report)

# Accuracy
accuracy = accuracy_score(y_true, y_pred_classes)
print(f'Accuracy: {accuracy:.2f}')
```

25/25 [==============================] - 11s 452ms/step



شکل 9. ماتریس آشفتگی معماری **VGGNet**

Classification Report:

	precision	recall	f1-score	support
Class 0	0.44	0.26	0.32	98
Class 1	0.23	0.18	0.20	89
Class 2	0.23	0.05	0.08	120
Class 3	0.29	0.31	0.30	102
Class 4	0.33	0.29	0.31	92
Class 5	0.28	0.20	0.23	92
Class 6	0.26	0.44	0.33	102
Class 7	0.31	0.66	0.42	105
accuracy			0.30	800
macro avg	0.30	0.30	0.28	800
weighted avg	0.30	0.30	0.27	800

Accuracy: 0.30

۴-۳-۲-۱. مقایسه دو مدل AlexNet و VGGNet

AlexNet با دقت کلی بالاتر از VGGNet بهتر عمل می کند.
AlexNetعموماً precision, recall, and F1-score بهتری را در اکثر کلاس‌ها در مقایسه با VGGNet دارد.
تجزیه و تحلیل عملکرد بهتری را در بهبود true positives و کاهش false positives و false negatives برای AlexNet نشان می دهد.
هر دو مدل دارای میانگین های macro و وزنی مشابه هستند، اما مقادیر کمی بالاتر AlexNet نشان دهنده عملکرد کلی بهتر در بین کلاس ها است.
دلیل: انواع مختلف پشته 3 در 3 در VGGNet استفاده می شود.

۴-۲-۱. معماری و یادگیری مدل MobileNet

از 3 × 3 لایه های کانولوشن قابل جداسازی عمقی و به دنبال آن لایه های کانولوشن معمولی 1 × 1 برای حفظ عملکرد بالا و در عین حال به حداقل رساندن پیچیدگی معماری استفاده می کند. این منجر به معماری های شبکه بسیار کوچکتر و قابل تنظیم و مناسب برای استقرار در دستگاه های تلفن همراه می شود. بلوک های کانولوشن قابل تفکیک به صورت عمقی (DConv) همانطور که در توضیح داده شده است استفاده می شود. کاهش پیچیدگی از نظر لایه را از مدل بسیار عمیق‌تری را فراهم می‌کند

Type	Shape	Stride	Output
Conv	3 × 3 × 32	2	64 × 64 × 32
DConv	3 × 3 × 64	1	64 × 64 × 64
DConv	3 × 3 × 128	2	32 × 32 × 128
DConv	3 × 3 × 128	1	32 × 32 × 128
DConv	3 × 3 × 256	2	16 × 16 × 256
DConv	3 × 3 × 256	1	16 × 16 × 256
DConv	3 × 3 × 512	2	8 × 8 × 512
5×DConv	3 × 3 × 512	1	8 × 8 × 512
DConv	3 × 3 × 1024	2	4 × 4 × 1024
DConv	3 × 3 × 1024	1	4 × 4 × 1024
GlobalAvePool	1024	–	–
Dense	8 or 2	–	1 label or 2 floats

شکل 10. معماری **MobileNet**

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, DepthwiseConv2D, GlobalAveragePooling2D, Dense

def create_dconv_model(input_shape=(128, 128, 3), num_classes=8):
    model = Sequential()

    # Convolutional Block 1
    model.add(Conv2D(32, (3, 3), strides=2, input_shape=input_shape))
    # Depth-wise Separable Convolution Block 1
    model.add(DepthwiseConv2D((3, 3), strides=1))
    model.add(DepthwiseConv2D((3, 3), strides=2))
    model.add(DepthwiseConv2D((3, 3), strides=1))

    # Depth-wise Separable Convolution Block 2
    model.add(DepthwiseConv2D((3, 3), strides=2, padding='same'))
```

```
    model.add(DepthwiseConv2D((3, 3), strides=1, padding='same'))

    # Depth-wise Separable Convolution Block 3
    model.add(DepthwiseConv2D((3, 3), strides=2, padding='same'))
    for _ in range(5):  # Repeat the block 5 times
        model.add(DepthwiseConv2D((3, 3), strides=1, padding='same'))

    # Depth-wise Separable Convolution Block 4
    model.add(DepthwiseConv2D((3, 3), strides=2))
    model.add(DepthwiseConv2D((3, 3), strides=1))

    # Global Average Pooling
    model.add(GlobalAveragePooling2D())

    # Fully Connected (Dense) Layer
    model.add(Dense(num_classes, activation='softmax'))

    return model
```

۱-۴-۲-۱. یادگیری مدل VGGNet

مدل را با استفاده از optimizer Adam و نرخ یادگیری 0.001 با ۲۴ ایپاک و batch_size ۳۲ آموزش می دهیم.

```
# Create the model
dconv_model = create_dconv_model()

# Display the model summary
dconv_model.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_31 (Conv2D)          (None, 63, 63, 32)        896

 depthwise_conv2d (Depthwis  (None, 61, 61, 32)        320
 eConv2D)

 depthwise_conv2d_1 (Depthw  (None, 30, 30, 32)        320
 iseConv2D)

 depthwise_conv2d_2 (Depthw  (None, 28, 28, 32)        320
 iseConv2D)

 depthwise_conv2d_3 (Depthw  (None, 14, 14, 32)        320
 iseConv2D)

 depthwise_conv2d_4 (Depthw  (None, 14, 14, 32)        320
 iseConv2D)
```

```
 depthwise_conv2d_5 (Depthw  (None, 7, 7, 32)          320
 iseConv2D)

 depthwise_conv2d_6 (Depthw  (None, 7, 7, 32)          320
...
Total params: 5320 (20.78 KB)
Trainable params: 5320 (20.78 KB)
Non-trainable params: 0 (0.00 Byte)
```

```
dconv_model= create_dconv_model(input_shape=(128, 128, 3), num_classes=8)
```

```
optimizer = Adam(lr=0.001)
dconv_model.compile(optimizer=optimizer, loss='categorical_crossentropy', metrics=['accuracy'])
```

```
history_dconv = dconv_model.fit(
    train_images,
    to_categorical(train_labels, num_classes=8),
    epochs=24,
    batch_size=32,
    verbose=1,
    validation_split=0.2
)
```

```
Epoch 1/24
30/30 [==============================] - 8s 151ms/step - loss: 2.0798 - accuracy: 0.1302 - val_loss:
2.0811 - val_accuracy: 0.1042
Epoch 2/24
30/30 [==============================] - 4s 124ms/step - loss: 2.0780 - accuracy: 0.1312 - val_loss:
2.0819 - val_accuracy: 0.1042
Epoch 3/24
30/30 [==============================] - 6s 198ms/step - loss: 2.0775 - accuracy: 0.1417 - val_loss:
2.0830 - val_accuracy: 0.1042
Epoch 4/24
30/30 [==============================] - 5s 165ms/step - loss: 2.0774 - accuracy: 0.1417 - val_loss:
2.0839 - val_accuracy: 0.1042
Epoch 5/24
30/30 [==============================] - 4s 124ms/step - loss: 2.0768 - accuracy: 0.1417 - val_loss:
2.0846 - val_accuracy: 0.1042
Epoch 6/24
30/30 [==============================] - 4s 124ms/step - loss: 2.0768 - accuracy: 0.1417 - val_loss:
2.0856 - val_accuracy: 0.1042
Epoch 7/24
30/30 [==============================] - 6s 211ms/step - loss: 2.0769 - accuracy: 0.1417 - val_loss:
2.0844 - val_accuracy: 0.1042
Epoch 8/24
30/30 [==============================] - 5s 149ms/step - loss: 2.0773 - accuracy: 0.1417 - val_loss:
2.0863 - val_accuracy: 0.1042
Epoch 9/24
30/30 [==============================] - 4s 122ms/step - loss: 2.0768 - accuracy: 0.1417 - val_loss:
2.0856 - val_accuracy: 0.1042
Epoch 10/24
30/30 [==============================] - 4s 123ms/step - loss: 2.0770 - accuracy: 0.1417 - val_loss:
2.0859 - val_accuracy: 0.1042
Epoch 11/24
30/30 [==============================] - 6s 218ms/step - loss: 2.0766 - accuracy: 0.1417 - val_loss:
2.0845 - val_accuracy: 0.1042
```

```
Epoch 12/24
30/30 [==============================] - 4s 141ms/step - loss: 2.0769 - accuracy: 0.1417 - val_loss:
2.0848 - val_accuracy: 0.1042
Epoch 13/24
...
Epoch 23/24
30/30 [==============================] - 6s 189ms/step - loss: 2.0768 - accuracy: 0.1417 - val_loss:
2.0856 - val_accuracy: 0.1042
Epoch 24/24
30/30 [==============================] - 4s 123ms/step - loss: 2.0769 - accuracy: 0.1417 - val_loss:
2.0847 - val_accuracy: 0.1042
```

ذخیره سازی و لود کردن مدل در فایل h5

```
dconv_model.save('dconv_model.h5')
```

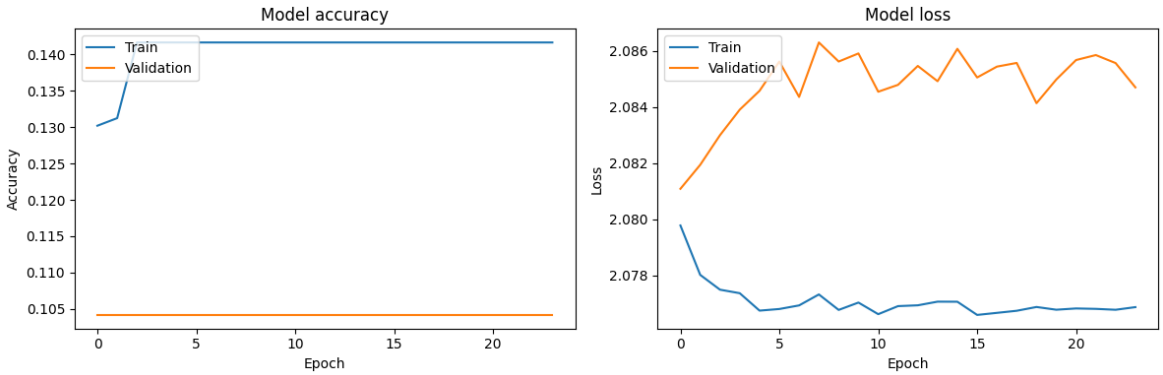
```
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3079 UserWarning: You are saving your
model as an HDF5 file via `model.save()`. This file format is considered legacy. We recommend using instead
the native Keras format, e.g. `model.save('my_model.keras')`.
```

```
loaded_dconv_model = load_model('dconv_model.h5')
```

۲-۴-۲-۱. نمودار Loss و Accuracy

ترسیم نمودار Loss و Accuracy

```
plt.figure(figsize=(12, 4))

plt.subplot(1, 2, 1)
plt.plot(history_dconv.history['accuracy'])
plt.plot(history_dconv.history['val_accuracy'])
plt.title('Model accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(['Train', 'Validation'], loc='upper left')

plt.subplot(1, 2, 2)
plt.plot(history_dconv.history['loss'])
plt.plot(history_dconv.history['val_loss'])
plt.title('Model loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend(['Train', 'Validation'], loc='upper left')

plt.tight_layout()
plt.show()
```



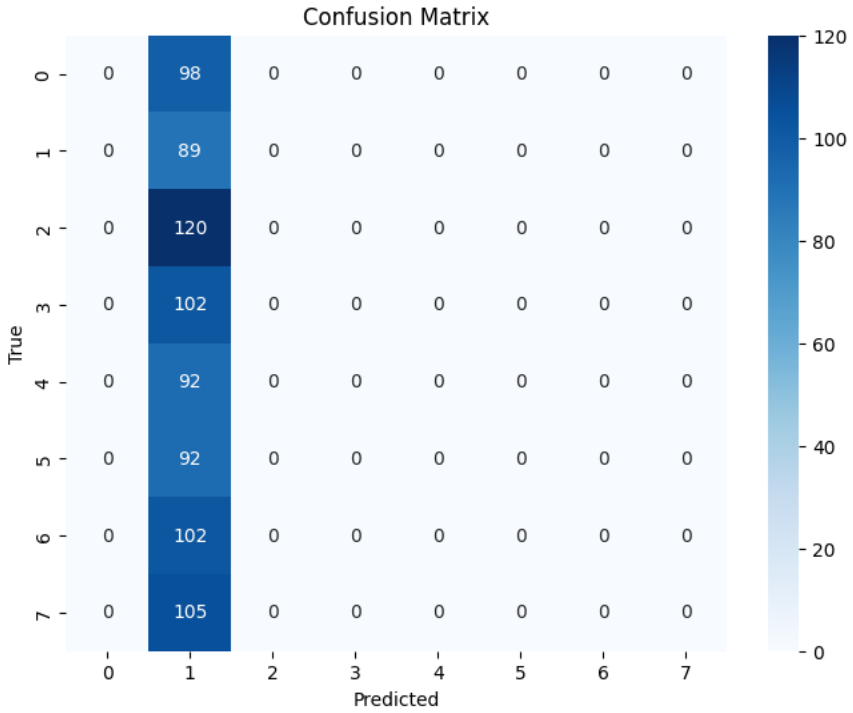
شکل 11. نمودار خطا و دقت مدل **MobileNet**

بر طبق نمودارهاي Loss و Accuracy مي‌بينيم كه داده نوسان زيادي روي داده‌هاي Validation دارد. ولی داده های Validation دقت مناسبی ندارند.

۳-۴-۲-۱. مقادیر Recall، Precision، F1 و ماتریس Confusion

```
test_pred = loaded_dconv_model.predict(test_images)
test_pred_classes = np.argmax(test_pred, axis=1)
test_true_classes = np.argmax(test_labels_one_hot, axis=1)

conf_mat = confusion_matrix(test_true_classes, test_pred_classes)

plt.figure(figsize=(8, 6))
sns.heatmap(conf_mat, annot=True, fmt='d', cmap='Blues', xticklabels=range(num_classes),
yticklabels=range(num_classes))
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.show()

print(classification_report(test_true_classes, test_pred_classes, target_names=[f'Class {i}' for i in
range(num_classes)]))

accuracy = accuracy_score(test_true_classes, test_pred_classes)
print(f'Accuracy: {accuracy:.2f}')
```

```
25/25 [==============================] - 2s 79ms/step
```



شکل 12. نمودار آشفتگی مدل **MobileNet**

	precision	recall	f1-score	support
Class 0	0.00	0.00	0.00	98
Class 1	0.11	1.00	0.20	89
Class 2	0.00	0.00	0.00	120
Class 3	0.00	0.00	0.00	102
Class 4	0.00	0.00	0.00	92
Class 5	0.00	0.00	0.00	92
Class 6	0.00	0.00	0.00	102
Class 7	0.00	0.00	0.00	105
accuracy			0.11	800
macro avg	0.01	0.12	0.03	800
weighted avg	0.01	0.11	0.02	800

Accuracy: 0.11

۴-۴-۲-۱. تفاوت با دو مدل قبل

بلوک‌های DConv از نظر عمقی قابل تفکیک هستند، به این معنی که از یک لایه پیچیدگی عمیق و به دنبال آن یک لایه پیچشی نقطه‌ای تشکیل شده‌اند. پیچیدگی معرفی شده توسط این بلوک ها ممکن است به تنظیم و هایپرپارامترهای متفاوتی نیاز داشته باشند و همچنین بلوک‌های DConv معمولاً پارامترهای کمتری نسبت به لایه‌های کانولوشنال معمولی دارند که ممکن است بر توانایی مدل برای گرفتن الگوهای پیچیده در داده‌ها تأثیر بگذارد.

پاسخ ۳ - تشخیص بیماران مبتال به کووید با استفاده از عکس ریه

۱-۳-۱. وظیفه

در این مساله باید به کمک مدل CNN طراحی شده در مقاله با عنوان:

An Efficient CNNModel for COVID-19 Disease Detection Based on X-Ray Image Classification

که مدلی با دقت بالا برای تشخیص بیماران سالم از بیماران کرونایی ارائه می‌کند را پیاده سازی کنیم.
علت دارای اهمیت بودن این عنوان بر آن است که در حوزه‌های پزشکی ، چون با انسان سر و کار داریم پیاده کردن مدل با دقت بالا بسیار مهم تلقی می‌شود. به طوری که اگر در نظر بگیریم مدلی داریم که با دقت 70 درصد به درستی دو بیمار کرونایی و عادی را کلاسه بندی کند ، در صورتی که 100 نمونه بگیرد یعنی 30 نمونه را اشتباه Label می‌زند و این به این معنا خواهد بود که در عمل 30 بیمار کرونایی وارد اجتماع خواهد شد بی آنکه شناسایی شده باشند. بنابر این در این تمرین مدل ما بر روی داده های تصاویر Xray از سه مجموعه دادگان مختلف تغزیه می‌شود. دو تای آنها برای یادگیری و Fine tune و یکی آنها برای تست استفاده می‌شود. وظیفه اصلی آن بالا بردن دقت تشخیص تا حد قابل قبول خواهد بود.

۲-۳-۱. جمع آوری دادگان و پیش پردازش

از اصلی ترین بخش‌های کار روی این تمرین بخش پیش پردازش داده‌ها خواهد بود زیرا در این تمرین داده‌ها از منابع مختلف جمع آوری می‌شود. عموما در کار با داده‌های پزشکی به دلیل کم بودن داده برچسب خورده نیاز مند آن خواهیم بود که از منابع گوناگون بهره بگیریم. به همین منظور برای Validation و Train از داده‌های به دست آمده دیتا-ست موجود Kaggle و Github که در مقاله اشاره شده استفاده می‌کنیم. برای داده های تست نیز در داده‌های مستقل IEEE که باز در مقاله آورده شده استفاده می‌کنیم.
در ادامه برای حل تمرین دیتاست جامع کل در اختیار دانشجو قرار داده شده در ادامه حل تمرین هر دو روش پیش پردازش برای این دو مدل جمع آوری دادگان ارائه خواهد شد.
به این منظور لازم است داده‌ها را پس از تفکیک به Train, Test, and Validation به طبق جدول زیر گروه بندی تعداد کنیم. برای این کار داده‌های 90 تایی Train را Augment خواهیم کرد که جلوتر اشاره می‌شود.

Dataset	COVID-19 images	Normal images	Total images
Total data	450	450	900
Training data	400	400	800
Testing data	50	50	100
Independent validation data	100	100	200

شکل 13. جدول 1 مقاله – نحوه توزیع دادگان مقاله

۱-۲-۳-۱. اضافه کردن کتابخانه‌ها

برای حل تمرین و ساخت شبکه CNN میتوان هر یک از کتابخانه‌های Pythorch یا Kerberos را استفاده کرد در اینجا ما از Tensorflow و بخصوص Kerberos استفاده کردیم.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import cv2
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from tensorflow import keras
```

```
from tensorflow.keras import layers, models
from tensorflow.keras.models import Sequential
from keras.preprocessing.image import ImageDataGenerator
import keras.utils as image
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense,
BatchNormalization, Dropout
import os
import seaborn as sns
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.utils import to_categorical
from keras.callbacks import ModelCheckpoint, EarlyStopping
from sklearn.model_selection import StratifiedShuffleSplit
from sklearn.utils import compute_class_weight
from sklearn.metrics import confusion_matrix, classification_report, roc_curve, auc
import shutil
from google.colab import drive
from google.colab import files
```

۲-۲-۳-۱. جمع آوری داده ها

- همانطور که اشاره شد یک روش جمع آوری داده‌ها طبقه بندی مطابق مقاله است

برای این منظور داده‌ها را از سایت Kaggle با TOKEN JSON در COLAB وارد می‌کنیم.

```
uploaded = files.upload()

!mkdir -p ~/.kaggle
!mv kaggle.json ~/.kaggle/
!chmod 600 ~/.kaggle/kaggle.json
!kaggle datasets download -d paultimothymooney/chest-xray-pneumonia
!unzip -q chest-xray-pneumonia.zip
```

همینطور داده گیت‌هاب را Clone می‌کنیم.

```
!git clone https://github.com/ieee8023/covid-chestxray-dataset.git
```

در ادامه کارهای تفکیک دادگانی که مورد انتظار است را انجام می‌دهیم.

```
FILE_PATH = "/content/covid-chestxray-dataset/metadata.csv"
IMAGE_PATH = "/content/covid-chestxray-dataset/images/"

df = pd.read_csv(FILE_PATH)
print(df.shape)
```

```
(950, 30)
```

```
df.head()
```

	patientid	offset	sex	age	finding	RT_PCR_positive	survival	intubated	intubation_present	went_icu	...	date	location	folder	filename	doi	
0	2	0.0	M	65.0	Pneumonia/Viral/COVID-19		Y	Y	N	N	N	...	January 22, 2020	Cho Ray Hospital, Ho Chi Minh City, Vietnam	images	auntminnie-a-2020_01_28_23_51_6665_2020_01_28_...	10.1056/nejmc2001272	https:/
1	2	3.0	M	65.0	Pneumonia/Viral/COVID-19		Y	Y	N	N	N	...	January 25, 2020	Cho Ray Hospital, Ho Chi Minh City, Vietnam	images	auntminnie-b-2020_01_28_23_51_6665_2020_01_28_...	10.1056/nejmc2001272	https:/
2	2	5.0	M	65.0	Pneumonia/Viral/COVID-19		Y	Y	N	N	N	...	January 27, 2020	Cho Ray Hospital, Ho Chi Minh City, Vietnam	images	auntminnie-c-2020_01_28_23_51_6665_2020_01_28_...	10.1056/nejmc2001272	https:/
3	2	6.0	M	65.0	Pneumonia/Viral/COVID-19		Y	Y	N	N	N	...	January 28, 2020	Cho Ray Hospital, Ho Chi Minh City, Vietnam	images	auntminnie-d-2020_01_28_23_51_6665_2020_01_28_...	10.1056/nejmc2001272	https:/
4	4	0.0	F	52.0	Pneumonia/Viral/COVID-19		Y	NaN	N	N	N	...	January 25, 2020	Changhua Christian Hospital, Changhua City, Ta...	images	nejmc2001573_f1a.jpeg	10.1056/NEJMc2001573	https:/

5 rows × 30 columns

شکل 14. مشاهده **Head** دادگان **Kaggle**

```
DIR_NORMAL = os.path.join(PARENT_DIR, "Normal")
TARGET_DIR_COVID = os.path.join(PARENT_DIR, "Covid")
PARENT_DIR = "/content/Dataset"


if not os.path.exists(PARENT_DIR):
    os.mkdir(PARENT_DIR)
    print("Parent directory created")

if not os.path.exists(TARGET_DIR_COVID):
    os.mkdir(TARGET_DIR_COVID)
    print("Covid folder created")
if not os.path.exists(DIR_NORMAL):
    os.mkdir(DIR_NORMAL)
    print("Normal directory created")
```

```
Parent directory created
Covid folder created
Normal directory created
```

از 178 تصویر، 136 تصویر اشعه ایکس متعلق به بیماران تایید شده کووید-19 و 42 تصویر دیگر متعلق به افراد عادی یا مبتلا به بیماری های دیگر است. تجزیه و تحلیل تصاویر اشعه ایکس توسط متخصصان پزشکی. یک تجزیه و تحلیل عمیق بر روی تصاویر اشعه ایکس توسط متخصصان پزشکی انجام شد. از 135 تصویر اشعه ایکس از بیماران تایید شده COVID-19، تنها مجموعه ای از 90 تصویر اشعه ایکس به عنوان کاندیدای عالی برای آموزش مدل ها انتخاب شد.

```
cnt_covid = 0

max_images_covid = 90

for i, row in df.iterrows():
    filename = row["filename"]
    image_path = os.path.join(IMAGE_PATH, filename)

    if row["finding"] == "Pneumonia/Viral/COVID-19" and row["view"] == "PA" and
cnt_covid < max_images_covid:
        image_copy_path = os.path.join(TARGET_DIR_COVID, f"covid_{cnt_covid}.png")
        shutil.copy2(image_path, image_copy_path)
        cnt_covid += 1
```

```
        if cnt_covid >= max_images_covid:
            break


print(f"{cnt_covid} COVID-19 images (renamed) copied to {TARGET_DIR_COVID}")
cnt_normal = 0
max_images_normal = 42
for i, row in df.iterrows():
    filename = row["filename"]
    image_path = os.path.join(IMAGE_PATH, filename)

    if row["finding"] != "Pneumonia/Viral/COVID-19" and cnt_normal < max_images_normal:
        image_copy_path = os.path.join(DIR_NORMAL, f"normal_{cnt_normal}.png")
        shutil.copy2(image_path, image_copy_path)
        cnt_normal += 1

    if  cnt_normal >= max_images_normal:
        break

print(f"{cnt_normal} normal images copied to {DIR_NORMAL}")
```

بنابر این به روش بالا میتوانستیم داده‌هاي آموزش را تفکیك کنیم و در پوشه عادی یا COVID بریزیم.

```
ADDITIONAL_NORMAL_DIR = "/content/chest_xray/train/NORMAL"

cnt_additional_normal = 0

max_additional_normal_images = 136

for filename in os.listdir(ADDITIONAL_NORMAL_DIR):
    if cnt_additional_normal >= max_additional_normal_images:
        break

    image_path = os.path.join(ADDITIONAL_NORMAL_DIR, filename)
    image_copy_path = os.path.join(DIR_NORMAL, f"normal_{cnt_additional_normal +
max_images_normal}.png")

    shutil.copy2(image_path, image_copy_path)
    cnt_additional_normal += 1

print(f"{cnt_additional_normal} additional normal images copied to {DIR_NORMAL}")
```

- روش دوم اما ساده تر است چرا که دستیاران آموزشی دیتاست تمیز را در اختیار دانشجو قرار داده اند. برای همین میتوان ادامه مسیر را از این روش نیز استفاده کرد.

برای افزودن داده ها به این روش ابتدا داده‌ها را در درایو ریخته از انجا import می‌کنیم در Colab:

```
drive.mount('/content/drive')
```

```
import zipfile

dataset_path = '/content/drive/MyDrive/NN_Q3_Dataset/Dataset.zip'

extraction_path = '/content/dataset/'
```

```
os.makedirs(extraction_path, exist_ok=True)

with zipfile.ZipFile(dataset_path, 'r') as zip_ref:
    zip_ref.extractall(extraction_path)
dataset_base_directory = '/content/dataset/Dataset/xray_dataset_covid19/'
sub_directories = ['train/', 'test/']

classes = os.listdir(os.path.join(dataset_base_directory, sub_directories[0]))

num_classes = len(classes)

print(f"Number of classes: {num_classes}")
print("Classes:")
for class_name in classes:
    print(class_name)
```

```
Number of classes: 2
Classes:
NORMAL
COVID
```

سپس فرایند نرمال سازی و تعیین سایز 150 شروع می‌شود.

```
IMG_SIZE = 150
X = []
y = []

for sub_dir in sub_directories:
    for c in range(len(classes)):
        dir = os.path.join(dataset_base_directory, sub_dir, classes[c])

        for f in os.listdir(dir):
            img = cv2.imread(os.path.join(dir, f))
            img = cv2.resize(img, (IMG_SIZE, IMG_SIZE))
            img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
            img = img / 255.0
            X.append(img)
            y.append(c)

X = np.asarray(X)
y = np.asarray(y)


print(f"Shape of X: {X.shape}")
print(f"Shape of y: {y.shape}")
print("Label distribution:", np.bincount(y) / len(y))
```

```
Shape of X: (188, 150, 150, 3)
Shape of y: (188,)
Label distribution: [0.5 0.5]
```

۳-۲-۳-۱. تکثیر داده ها Data Augment

بر اساس مقاله بین تعداد داده‌های آموزش و تست و Fine Tune ناهمگونی تعداد وجود دارد پس نیاز است که تعداد را زیاد کنیم. به همین منظور داده‌ها را Flip و Rotate می‌دهیم:

Image type	Count
Original	90
Original flipped	90
Original with a 90-degree rotation	90
Original with 180-degree rotation	90
Original with 270-degree rotation	90
Total	450

شکل 15. داده افزایی مطابق مقاله

در ادامه چون مدل نیاز بود 50 Epoch آموزش ببیند تعداد داده‌ها زیاد شده و زمان زیادی صرف یادگیری می‌شد. تعداد داده ها را به صورت هدف دار کمتر کردیم که مدل سبک تر باشد. در ادامه یک لایه از شبکه را نیز به همین علت کمتر میکنیم صرفا برای اینکه شبکه سریع تر Converge کند. و توازن داده ها را تا حد خوبی نزدیک مقاله نگه داشتیم.

```
X_reversed = np.flip(X, axis=2)
y_reversed = y

X = np.concatenate((X, X_reversed), axis=0)
y = np.concatenate((y, y_reversed), axis=0)

X_rotated_90= np.rot90(X, axes=(1, 2))
# X_rotated_180 = np.rot90(X_rotated_90, axes=(1, 2))
# X_rotated_270 = np.rot90(X_rotated_180, axes=(1, 2))

y_rotated = y

X = np.concatenate((X, X_rotated_90), axis=0)
# X = np.concatenate((X, X_rotated_180), axis=0)
# X = np.concatenate((X, X_rotated_270), axis=0)
y = np.concatenate((y, y_rotated), axis=0)
# y = np.concatenate((y, y_rotated), axis=0)
# y = np.concatenate((y, y_rotated), axis=0)
```

در مثال کد بالا نشان داده میشود که از داده افزایی 180 و 270 جلوگیری کردیم.

```
print("Shape of X_combined:", X.shape)

print("Shape of X:", X.shape)
print("Shape of y:", y.shape)
print("Number of samples:", len(X))
print("Number of labels:", len(y))
print("Unique labels:", np.unique(y))
print("Label distribution:", np.bincount(y) / len(y))
```

```
Shape of X_combined: (752, 150, 150, 3)
Shape of X: (752, 150, 150, 3)
Shape of y: (752,)
Number of samples: 752
Number of labels: 752
Unique labels: [0 1]
Label distribution: [0.5 0.5]
```

```
from sklearn.model_selection import StratifiedShuffleSplit

splitter = StratifiedShuffleSplit(n_splits=1, test_size=0.2, random_state=0)
```

```
train_index, test_index = next(splitter.split(X, y))

X_train, X_test = X[train_index], X[test_index]
y_train, y_test = y[train_index], y[test_index]

splitter = StratifiedShuffleSplit(n_splits=1, test_size=0.5, random_state=0)
test_index, val_index = next(splitter.split(X_test, y_test))

X_test, X_val = X_test[test_index], X_test[val_index]
y_test, y_val = y_test[test_index], y_test[val_index]
```

```
print("Shape of X_train:", X_train.shape)
print("Shape of y_train:", y_train.shape)
print("Shape of X_val:", X_val.shape)
print("Shape of y_val:", y_val.shape)
print("Shape of X_test:", X_test.shape)
print("Shape of y_test:", y_test.shape)
```

```
Shape of X_train: (601, 150, 150, 3)
Shape of y_train: (601,)
Shape of X_val: (76, 150, 150, 3)
Shape of y_val: (76,)
Shape of X_test: (75, 150, 150, 3)
Shape of y_test: (75,)
```

```
print(f'\t\t{classes[0]}\t\t{classes[1]}')
print(f'training:\t{np.count_nonzero(y_train==0)}\t\t{np.count_nonzero(y_train==1)}')
print(f'validation:\t{np.count_nonzero(y_val==0)}\t\t{np.count_nonzero(y_val==1)}')
print(f'testing:\t{np.count_nonzero(y_test==0)}\t\t{np.count_nonzero(y_test==1)}')
```

```
        NORMAL          COVID
training:       300             301
validation:     38              38
testing:        37              38
```

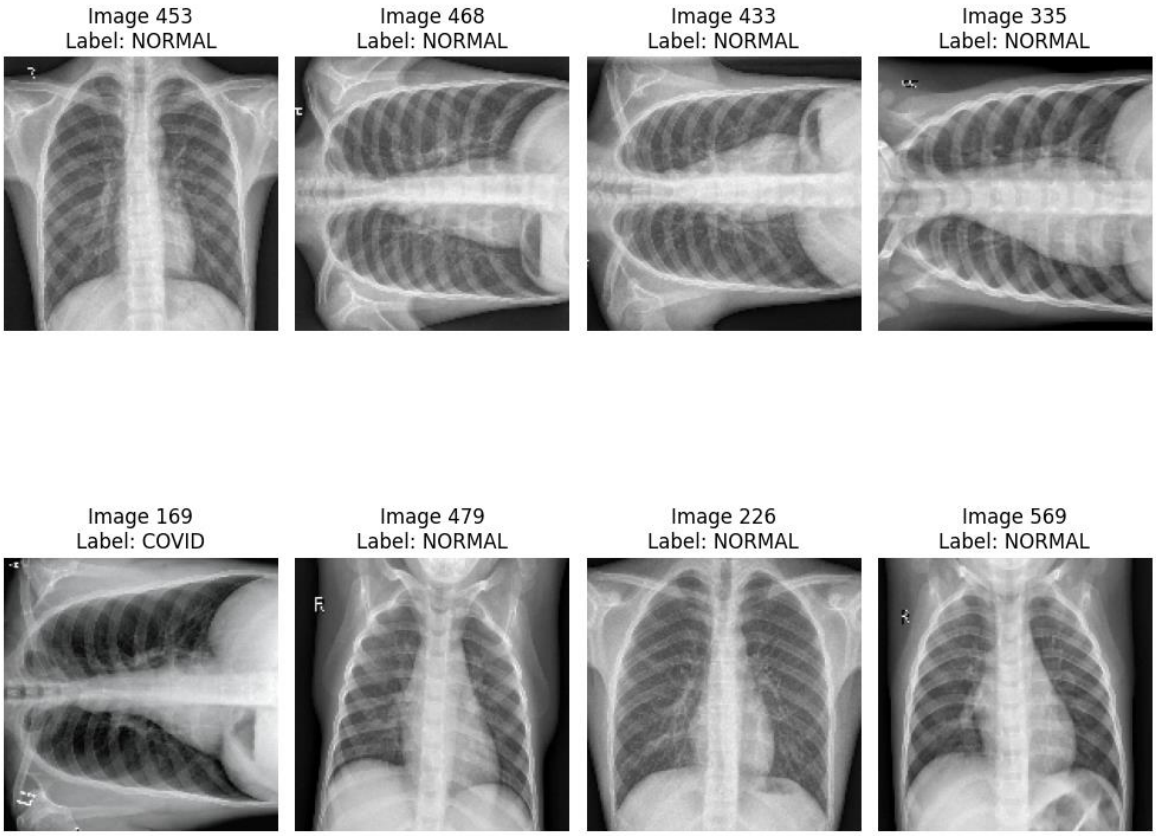
مطابق اینچه بدست می‌آیند نتایج در حقیقت باید مطابق جدول یک مقاله باشد. که همان جدول طبقه بندی دادگان اشاره شده در ابتدای حل این سوال بود. همان طور که گفته شد در اینجا به هدف افزایش Performance تعداد را کمتر کردیم ولی می‌توانستیم داده های Augment شده را uncomment کنیم و به توزیع 80 13 7 تقریبا به توزیع جدول اول برسیم. ولی در این صورت یادگیری خیلی زمان بر می‌شد که صرف نظر کردیم.

۴-۲-۳-۱. نمایش داده‌ها

در این بخش چند عکس رندم از Xray افراد نمایش داده می‌شود.

```
import random

num_images = 8

random_indices = random.sample(range(len(X_train)), num_images)

fig, axes = plt.subplots(2, 4, figsize=(10, 10))
axes = axes.flatten()

for i, index in enumerate(random_indices):
    image = X_train[index]
    label = y_train[index]

    image_number = index + 1
```

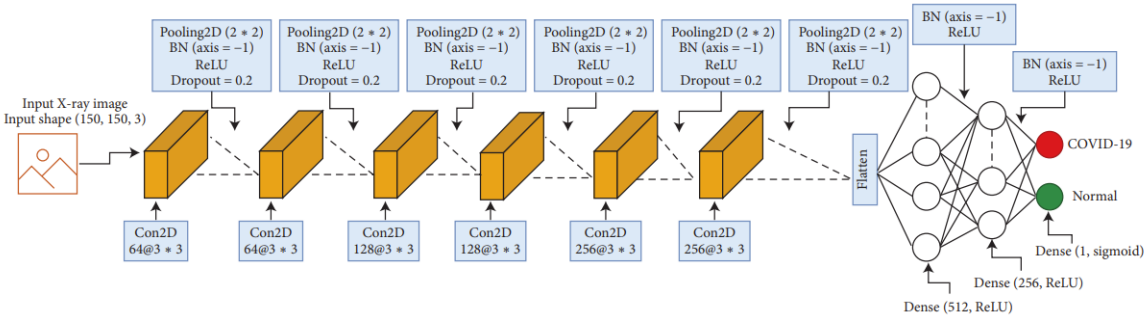
```
    axes[i].set_title(f"Image {image_number}\nLabel: {classes[label]}")

    axes[i].imshow(image)
    axes[i].axis("off")

plt.tight_layout()
plt.show()
```



شکل 16. نمایش نمونه‌های تصاویر رندم بدست آمده بعد از اعمال داده افزایی

۳-۳-۱. معماری و یادگیری مدل

در این حالت باز ما دوروش برای پیاده سازی شبکه CNN مطابق شکل زیر که مدل آورده شده در مقاله است داریم:



شکل 17. معماری شبکه کانولوشن استفاده شده برای یادگیری در مقاله

- روش اول ایجاد تمام Backbone و Architecture شبکه است (ما از این روش استفاده کردیم):
- روش دوم ساده تر استفاده از مدل های معروف مثل Efficient Net برای Backbone و ایجاد شبکه Classifier لایه اخر است. این روش به کمک کتابخانه‌های اضافه شده به راحتی قابل انجام خواهد بود.

بر اساس روش اول ما مدل شبکه خود را به این صورت تعریف می‌کنیم:

```
model = Sequential()

model.add(Conv2D(64,kernel_size=(3,3),activation="relu",input_shape=(150,150,3)))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(BatchNormalization(axis=-1))
model.add(Dropout(0.2))

model.add(Conv2D(64,(3,3),activation="relu"))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(BatchNormalization(axis=-1))
model.add(Dropout(0.2))

model.add(Conv2D(128,(3,3),activation="relu"))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(BatchNormalization(axis=-1))
model.add(Dropout(0.2))

model.add(Conv2D(128,(3,3),activation="relu"))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(BatchNormalization(axis=-1))
model.add(Dropout(0.2))

model.add(Conv2D(256,(3,3),activation="relu"))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(BatchNormalization(axis=-1))
model.add(Dropout(0.2))


model.add(Flatten())
model.add(Dense(512,activation="relu"))
model.add(BatchNormalization(axis=-1))
model.add(Dense(256,activation="relu"))
model.add(BatchNormalization(axis=-1))
model.add(Dense(1,activation="sigmoid"))

model.compile(loss=keras.losses.binary_crossentropy,optimizer =
"adam",metrics=["accuracy"])
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 148, 148, 64)      1792

 max_pooling2d (MaxPooling2D  (None, 74, 74, 64)       0
 )

 batch_normalization (BatchN  (None, 74, 74, 64)       256
```

```
 ormalization)

 dropout_2 (Dropout)         (None, 74, 74, 64)        0

 conv2d_1 (Conv2D)           (None, 72, 72, 64)        36928

 max_pooling2d_1 (MaxPooling  (None, 36, 36, 64)       0
 2D)

 batch_normalization_1 (Batc  (None, 36, 36, 64)       256
 hNormalization)

 dropout_3 (Dropout)         (None, 36, 36, 64)        0

 conv2d_2 (Conv2D)           (None, 34, 34, 128)       73856
...
Total params: 1,217,345
Trainable params: 1,214,529
Non-trainable params: 2,816
_________________________________________________________________
```

میتوانستیم با روش مدل دوم عمل کنیم و از snippet زیر استفاده کنیم ولی این کار را نکردیم:

```
# input_layer = keras.layers.Input(shape=(IMG_SIZE, IMG_SIZE, 3))

# efficientNet = keras.applications.EfficientNetB3(
#     include_top=False, weights="imagenet", input_shape=(IMG_SIZE, IMG_SIZE, 3))

# x = efficientNet(input_layer, training=True)
# x = keras.layers.GlobalAveragePooling2D()(x)
# x = keras.layers.Dense(IMG_SIZE, activation='relu')(x)
# x = keras.layers.Dropout(0.3)(x)
# x = keras.layers.Dense(64, activation='relu')(x)
# x = keras.layers.Dropout(0.2)(x)
# output_layer = keras.layers.Dense(1, activation='sigmoid')(x)

# model = keras.Model(input_layer, output_layer)
```

۱-۳-۳-۱. یادگیری مدل

در فرآیند یادگیری مدل ابتدا یک Image generator ایجاد می‌کنیم:

```
from keras.preprocessing import image

train_datagen = image.ImageDataGenerator(
    horizontal_flip=True,
    zoom_range=0.2,
    fill_mode='constant',
    cval=0.0
)
test_dataset = image.ImageDataGenerator(rescale = 1./255)
```

```
data_iterator = train_datagen.flow(X_train, y_train, batch_size=32)
steps_per_epoch = X_train.shape[0] // 32
data_iterator.x.size
```

```
40567500
```

میتوانستیم Validation Generator هم بسازیم:

```
validation_generator = test_dataset.flow_from_directory(
    '/content/chest_xray/val/',
```

```
    target_size = (150,150),
    batch_size = 32,
    class_mode = 'binary'
)
```

سپس یا کلاس‌ها را مطابق زیر میسازیم یا با استفاده از دستور کتابخانه Binary ست می‌کنیم:

```
class_weights = compute_class_weight('balanced', classes=[0, 1], y=y_train)

class_weight = {
    0: class_weights[0],
    1: class_weights[1]
}
```

دو دستور زیر را برای Early Stop اگر Epoch 20 مقدار Validation دقت افزایش نداشت اضافه می‌کنیم و نیز Check point قرار می‌دهیم که بهترین Validation را ذخیره کند در مقصد temp_final:

```
early_stopping = EarlyStopping(
    monitor='val_accuracy',   # Monitor validation accuracy
    patience=30,
    verbose=1,
    restore_best_weights=True

)
checkpoints = ModelCheckpoint(
    'temp_final/checkpoint',
    monitor='val_accuracy',   # Monitor validation accuracy
    save_best_only=True,
    verbose=1
)
```

مدل را باز compile کرده و fit می‌کنیم:

```
model.compile(loss=keras.losses.binary_crossentropy,optimizer =
"adam",metrics=["accuracy"])
hist = model.fit(
    data_iterator,
    epochs = 50,
    class_weight = class_weight,
    steps_per_epoch = steps_per_epoch,
    validation_data = (X_val, y_val),
    callbacks = [early_stopping, checkpoints]

)
```

```
Epoch 1/50
18/18 [==============================] - ETA: 0s - loss: 0.2721 - accuracy: 0.8963
Epoch 1: val_accuracy improved from -inf to 0.50000, saving model to
temp_final/checkpoint
WARNING:absl:Found untraced functions such as _jit_compiled_convolution_op,
_jit_compiled_convolution_op, _jit_compiled_convolution_op,
_jit_compiled_convolution_op, _jit_compiled_convolution_op while saving (showing 5 of
6). These functions will not be directly callable after loading.
18/18 [==============================] - 82s 4s/step - loss: 0.2721 - accuracy: 0.8963 -
val_loss: 2.3750 - val_accuracy: 0.5000
Epoch 2/50
18/18 [==============================] - ETA: 0s - loss: 0.1826 - accuracy: 0.9262
Epoch 2: val_accuracy did not improve from 0.50000
```

```
18/18 [==============================] - 72s 4s/step - loss: 0.1826 - accuracy: 0.9262 -
val_loss: 6.1123 - val_accuracy: 0.5000
Epoch 3/50
18/18 [==============================] - ETA: 0s - loss: 0.1118 - accuracy: 0.9631
Epoch 3: val_accuracy did not improve from 0.50000
18/18 [==============================] - 75s 4s/step - loss: 0.1118 - accuracy: 0.9631 -
val_loss: 6.2592 - val_accuracy: 0.5000
Epoch 4/50
18/18 [==============================] - ETA: 0s - loss: 0.1320 - accuracy: 0.9508
Epoch 4: val_accuracy did not improve from 0.50000
18/18 [==============================] - 70s 4s/step - loss: 0.1320 - accuracy: 0.9508 -
val_loss: 5.7431 - val_accuracy: 0.5000
Epoch 5/50
18/18 [==============================] - ETA: 0s - loss: 0.1690 - accuracy: 0.9420
Epoch 5: val_accuracy did not improve from 0.50000
18/18 [==============================] - 70s 4s/step - loss: 0.1690 - accuracy: 0.9420 -
val_loss: 12.6059 - val_accuracy: 0.5000
Epoch 6/50
18/18 [==============================] - ETA: 0s - loss: 0.1710 - accuracy: 0.9367
Epoch 6: val_accuracy did not improve from 0.50000
18/18 [==============================] - 71s 4s/step - loss: 0.1710 - accuracy: 0.9367 -
val_loss: 10.8824 - val_accuracy: 0.5000
Epoch 7/50
18/18 [==============================] - ETA: 0s - loss: 0.0955 - accuracy: 0.9666
Epoch 7: val_accuracy did not improve from 0.50000
18/18 [==============================] - 71s 4s/step - loss: 0.0955 - accuracy: 0.9666 -
val_loss: 10.1022 - val_accuracy: 0.5000
...
18/18 [==============================] - 72s 4s/step - loss: 0.0690 - accuracy: 0.9754 -
val_loss: 6.0231 - val_accuracy: 0.5000
Epoch 18/50
18/18 [==============================] - ETA: 0s - loss: 0.1120 - accuracy: 0.9666
Epoch 18: val_accuracy improved from 0.50000 to 0.51316, saving model to
temp_final/checkpoint
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
WARNING:absl:Found untraced functions such as _jit_compiled_convolution_op,
_jit_compiled_convolution_op, _jit_compiled_convolution_op,
_jit_compiled_convolution_op, _jit_compiled_convolution_op while saving (showing 5 of
6). These functions will not be directly callable after loading.
18/18 [==============================] - 78s 4s/step - loss: 0.1120 - accuracy: 0.9666 -
val_loss: 2.6591 - val_accuracy: 0.5132
Epoch 19/50
18/18 [==============================] - ETA: 0s - loss: 0.0755 - accuracy: 0.9736
Epoch 19: val_accuracy did not improve from 0.51316
18/18 [==============================] - 69s 4s/step - loss: 0.0755 - accuracy: 0.9736 -
val_loss: 2.0022 - val_accuracy: 0.5132
Epoch 20/50
18/18 [==============================] - ETA: 0s - loss: 0.0646 - accuracy: 0.9754
Epoch 20: val_accuracy did not improve from 0.51316
18/18 [==============================] - 71s 4s/step - loss: 0.0646 - accuracy: 0.9754 -
val_loss: 4.4073 - val_accuracy: 0.5000
Epoch 21/50
18/18 [==============================] - ETA: 0s - loss: 0.0801 - accuracy: 0.9740
Epoch 21: val_accuracy did not improve from 0.51316
18/18 [==============================] - 72s 4s/step - loss: 0.0801 - accuracy: 0.9740 -
val_loss: 4.1501 - val_accuracy: 0.5132
Epoch 22/50
18/18 [==============================] - ETA: 0s - loss: 0.0580 - accuracy: 0.9754
Epoch 22: val_accuracy did not improve from 0.51316
18/18 [==============================] - 70s 4s/step - loss: 0.0580 - accuracy: 0.9754 -
val_loss: 4.1875 - val_accuracy: 0.5000
Epoch 23/50
18/18 [==============================] - ETA: 0s - loss: 0.0849 - accuracy: 0.9701
Epoch 23: val_accuracy did not improve from 0.51316
18/18 [==============================] - 72s 4s/step - loss: 0.0849 - accuracy: 0.9701 -
val_loss: 5.1779 - val_accuracy: 0.5000
```

```
Epoch 24/50
18/18 [==============================] - ETA: 0s - loss: 0.0223 - accuracy: 0.9930
Epoch 24: val_accuracy did not improve from 0.51316
18/18 [==============================] - 72s 4s/step - loss: 0.0223 - accuracy: 0.9930 -
val_loss: 5.5570 - val_accuracy: 0.5000
...
18/18 [==============================] - 72s 4s/step - loss: 0.0131 - accuracy: 0.9982 -
val_loss: 2.9027 - val_accuracy: 0.5000
Epoch 28/50
18/18 [==============================] - ETA: 0s - loss: 0.0197 - accuracy: 0.9965
Epoch 28: val_accuracy improved from 0.51316 to 0.60526, saving model to
temp_final/checkpoint
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

```
WARNING:absl:Found untraced functions such as _jit_compiled_convolution_op,
_jit_compiled_convolution_op, _jit_compiled_convolution_op,
_jit_compiled_convolution_op, _jit_compiled_convolution_op while saving (showing 5 of
6). These functions will not be directly callable after loading.
18/18 [==============================] - 73s 4s/step - loss: 0.0197 - accuracy: 0.9965 -
val_loss: 1.6085 - val_accuracy: 0.6053
Epoch 29/50
18/18 [==============================] - ETA: 0s - loss: 0.0371 - accuracy: 0.9912
Epoch 29: val_accuracy improved from 0.60526 to 0.84211, saving model to
temp_final/checkpoint
WARNING:absl:Found untraced functions such as _jit_compiled_convolution_op,
_jit_compiled_convolution_op, _jit_compiled_convolution_op,
_jit_compiled_convolution_op, _jit_compiled_convolution_op while saving (showing 5 of
6). These functions will not be directly callable after loading.
18/18 [==============================] - 75s 4s/step - loss: 0.0371 - accuracy: 0.9912 -
val_loss: 0.7960 - val_accuracy: 0.8421
Epoch 30/50
18/18 [==============================] - ETA: 0s - loss: 0.0340 - accuracy: 0.9859
Epoch 30: val_accuracy improved from 0.84211 to 0.85526, saving model to
temp_final/checkpoint
WARNING:absl:Found untraced functions such as _jit_compiled_convolution_op,
_jit_compiled_convolution_op, _jit_compiled_convolution_op,
_jit_compiled_convolution_op, _jit_compiled_convolution_op while saving (showing 5 of
6). These functions will not be directly callable after loading.
18/18 [==============================] - 75s 4s/step - loss: 0.0340 - accuracy: 0.9859 -
val_loss: 0.6903 - val_accuracy: 0.8553
Epoch 31/50
18/18 [==============================] - ETA: 0s - loss: 0.0529 - accuracy: 0.9754
Epoch 31: val_accuracy did not improve from 0.85526
18/18 [==============================] - 70s 4s/step - loss: 0.0529 - accuracy: 0.9754 -
val_loss: 1.0528 - val_accuracy: 0.7763
Epoch 32/50
18/18 [==============================] - ETA: 0s - loss: 0.0707 - accuracy: 0.9807
Epoch 32: val_accuracy did not improve from 0.85526
18/18 [==============================] - 70s 4s/step - loss: 0.0707 - accuracy: 0.9807 -
val_loss: 0.7828 - val_accuracy: 0.8289
Epoch 33/50
18/18 [==============================] - ETA: 0s - loss: 0.0746 - accuracy: 0.9719
Epoch 33: val_accuracy did not improve from 0.85526
18/18 [==============================] - 69s 4s/step - loss: 0.0746 - accuracy: 0.9719 -
val_loss: 0.6295 - val_accuracy: 0.7632
Epoch 34/50
18/18 [==============================] - ETA: 0s - loss: 0.0629 - accuracy: 0.9842
Epoch 34: val_accuracy did not improve from 0.85526
18/18 [==============================] - 68s 4s/step - loss: 0.0629 - accuracy: 0.9842 -
val_loss: 0.9331 - val_accuracy: 0.7500
Epoch 35/50
18/18 [==============================] - ETA: 0s - loss: 0.0508 - accuracy: 0.9772
Epoch 35: val_accuracy improved from 0.85526 to 0.94737, saving model to
temp_final/checkpoint
WARNING:absl:Found untraced functions such as _jit_compiled_convolution_op,
_jit_compiled_convolution_op, _jit_compiled_convolution_op,
```

```
_jit_compiled_convolution_op, _jit_compiled_convolution_op while saving (showing 5 of
6). These functions will not be directly callable after loading.
18/18 [==============================] - 77s 4s/step - loss: 0.0508 - accuracy: 0.9772 -
val_loss: 0.1766 - val_accuracy: 0.9474
Epoch 36/50
18/18 [==============================] - ETA: 0s - loss: 0.0230 - accuracy: 0.9912
Epoch 36: val_accuracy did not improve from 0.94737
18/18 [==============================] - 70s 4s/step - loss: 0.0230 - accuracy: 0.9912 -
val_loss: 3.1055 - val_accuracy: 0.5263
Epoch 37/50
18/18 [==============================] - ETA: 0s - loss: 0.0488 - accuracy: 0.9789
Epoch 37: val_accuracy did not improve from 0.94737
18/18 [==============================] - 72s 4s/step - loss: 0.0488 - accuracy: 0.9789 -
val_loss: 0.8372 - val_accuracy: 0.7763
Epoch 38/50
18/18 [==============================] - ETA: 0s - loss: 0.0371 - accuracy: 0.9912
Epoch 38: val_accuracy did not improve from 0.94737
18/18 [==============================] - 70s 4s/step - loss: 0.0371 - accuracy: 0.9912 -
val_loss: 0.2165 - val_accuracy: 0.9474
Epoch 39/50
18/18 [==============================] - ETA: 0s - loss: 0.0138 - accuracy: 0.9947
Epoch 39: val_accuracy did not improve from 0.94737
18/18 [==============================] - 69s 4s/step - loss: 0.0138 - accuracy: 0.9947 -
val_loss: 0.4990 - val_accuracy: 0.8553
Epoch 40/50
18/18 [==============================] - ETA: 0s - loss: 0.0417 - accuracy: 0.9824
Epoch 40: val_accuracy did not improve from 0.94737
18/18 [==============================] - 70s 4s/step - loss: 0.0417 - accuracy: 0.9824 -
val_loss: 2.3125 - val_accuracy: 0.6579
Epoch 41/50
18/18 [==============================] - ETA: 0s - loss: 0.0259 - accuracy: 0.9930
Epoch 41: val_accuracy did not improve from 0.94737
18/18 [==============================] - 71s 4s/step - loss: 0.0259 - accuracy: 0.9930 -
val_loss: 0.6070 - val_accuracy: 0.8158
...
18/18 [==============================] - 70s 4s/step - loss: 0.0328 - accuracy: 0.9912 -
val_loss: 0.4436 - val_accuracy: 0.8816
Epoch 44/50
18/18 [==============================] - ETA: 0s - loss: 0.0255 - accuracy: 0.9912
Epoch 44: val_accuracy improved from 0.94737 to 0.97368, saving model to
temp_final/checkpoint
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

```
WARNING:absl:Found untraced functions such as _jit_compiled_convolution_op,
_jit_compiled_convolution_op, _jit_compiled_convolution_op,
_jit_compiled_convolution_op, _jit_compiled_convolution_op while saving (showing 5 of
6). These functions will not be directly callable after loading.
18/18 [==============================] - 75s 4s/step - loss: 0.0255 - accuracy: 0.9912 -
val_loss: 0.0680 - val_accuracy: 0.9737
Epoch 45/50
18/18 [==============================] - ETA: 0s - loss: 0.0265 - accuracy: 0.9930
Epoch 45: val_accuracy did not improve from 0.97368
18/18 [==============================] - 72s 4s/step - loss: 0.0265 - accuracy: 0.9930 -
val_loss: 2.3441 - val_accuracy: 0.5263
Epoch 46/50
18/18 [==============================] - ETA: 0s - loss: 0.0190 - accuracy: 0.9947
Epoch 46: val_accuracy did not improve from 0.97368
18/18 [==============================] - 69s 4s/step - loss: 0.0190 - accuracy: 0.9947 -
val_loss: 2.1944 - val_accuracy: 0.5000
Epoch 47/50
18/18 [==============================] - ETA: 0s - loss: 0.0208 - accuracy: 0.9912
Epoch 47: val_accuracy did not improve from 0.97368
18/18 [==============================] - 72s 4s/step - loss: 0.0208 - accuracy: 0.9912 -
val_loss: 3.0615 - val_accuracy: 0.4868
Epoch 48/50
18/18 [==============================] - ETA: 0s - loss: 0.0359 - accuracy: 0.9859
Epoch 48: val_accuracy did not improve from 0.97368
```

```
18/18 [==============================] - 70s 4s/step - loss: 0.0359 - accuracy: 0.9859 -
val_loss: 2.8135 - val_accuracy: 0.5263
Epoch 49/50
18/18 [==============================] - ETA: 0s - loss: 0.0247 - accuracy: 0.9912
Epoch 49: val_accuracy did not improve from 0.97368
18/18 [==============================] - 69s 4s/step - loss: 0.0247 - accuracy: 0.9912 -
val_loss: 5.8175 - val_accuracy: 0.5000
Epoch 50/50
18/18 [==============================] - ETA: 0s - loss: 0.0183 - accuracy: 0.9912
Epoch 50: val_accuracy did not improve from 0.97368
18/18 [==============================] - 69s 4s/step - loss: 0.0183 - accuracy: 0.9912 -
val_loss: 5.2955 - val_accuracy: 0.5000
```

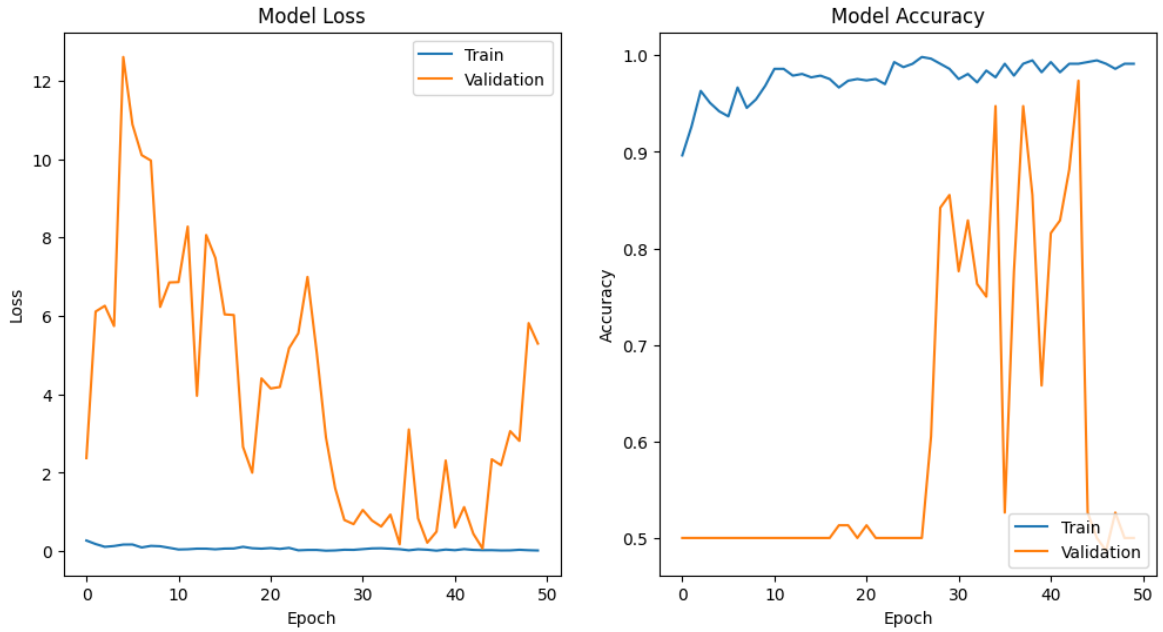
خب همان طور که اشاره شده دادهها را مطابق شکل زیر

Parameter	Value
Input dimension	(150, 150, 3)
Filter to learn	64, 128, 256
Max pooling	2×2
Batch normalization	Axis $= -1$
Activation functions	ReLU, sigmoid
Dropout rate	20%
Kernel size	3×3
Epochs	50
Optimizer	Adam
Loss function	binary_crossentropy

شکل 17. فاکتورهای آموزش مدل

آموزش داديم طي 50 نسل، اگر مقدار final_temp را نداشتيم روي نسل آخر نمايش داده شده كه Value_accuracy برابر 50 درصد است پس مدل در واقع Overfit كرده روي داده آموزش و روي تست و Validation به شكل 50 درصد مي‌رسيديم روي دقت داده تست تقريبا كه انگار مدل هيچ كيفيتي نداشت.
از طرفي الان ميتوانيم بهترين مدل عملكردي خودمان كه دقت بالاي 90 درصد روي هم داده Validation و هم Train گرفت را لود كنيم.

```
from tensorflow.keras.models import load_model

loaded_model = load_model('temp_final/checkpoint')
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.plot(hist.history['loss'], label='Train')
plt.plot(hist.history['val_loss'], label='Validation')
plt.title('Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend(loc='upper right')

plt.subplot(1, 2, 2)
plt.plot(hist.history['accuracy'], label='Train')
plt.plot(hist.history['val_accuracy'], label='Validation')
plt.title('Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(loc='lower right')
```



شکل 18. نمودار خطا و دقت شبکه طراحی شده

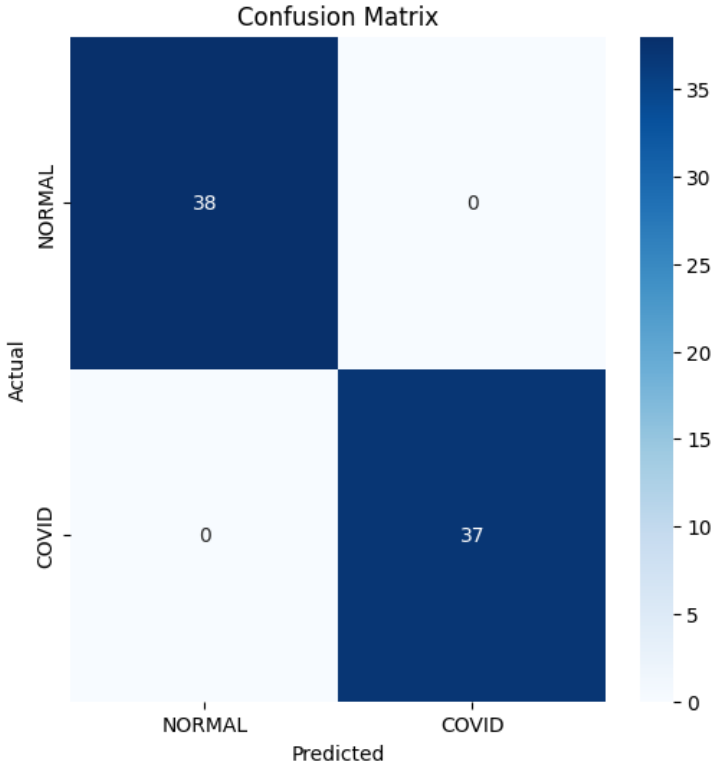
بر طبق نمودارهاي Loss و Accuracy مي‌بينيم كه داده نوسان زيادي روي داده‌هاي Validation دارد و دقت آن روي حدود نسل 44 تا حد Maximumخود رسيده و Loss آن كمينه شده است اين داده به ما كمك ميكند كه نسل مناسب مدل خود را براي داده تست انتخاب كنيم.

۴-۳-۱. ارزیابی

در بحث ارزيابي مدل معيارهايAccuracy ، Precision، Specificity، Sensitivityو score 1F را گزارش مي‌كنيم. همينطور ماتريس آشفتگي را نيز رسم مي‌كنيم:

```
probas = loaded_model.predict(X_test)

y_pred = np.asarray(0.5 < probas, dtype=np.int32)
y_pred_binary = (y_pred > 0.5).astype(int)

3/3 [==============================] - 4s 1s/step
test_loss, test_accuracy = loaded_model.evaluate(X_test, y_test)
print(f"\nTest Loss: {test_loss:.4f}")
print(f"Test Accuracy: {test_accuracy * 100:.2f}%")

y_pred = loaded_model.predict(X_test)
y_pred_binary = (y_pred > 0.5).astype(int)

print("\nClassification Report:")
print(classification_report(y_test, y_pred_binary))

conf_mat = confusion_matrix(y_test, y_pred_binary)

plt.figure(figsize=(6, 6))
sns.heatmap(conf_mat, annot=True, fmt="d", cmap="Blues", xticklabels=["NORMAL",
"COVID"], yticklabels=["NORMAL", "COVID"])
plt.title("Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```

```
3/3 [==============================] - 3s 823ms/step - loss: 0.0059 - accuracy: 1.0000

Test Loss: 0.0059
Test Accuracy: 100.00%
3/3 [==============================] - 2s 447ms/step

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        38
           1       1.00      1.00      1.00        37

    accuracy                           1.00        75
   macro avg       1.00      1.00      1.00        75
weighted avg       1.00      1.00      1.00        75
```



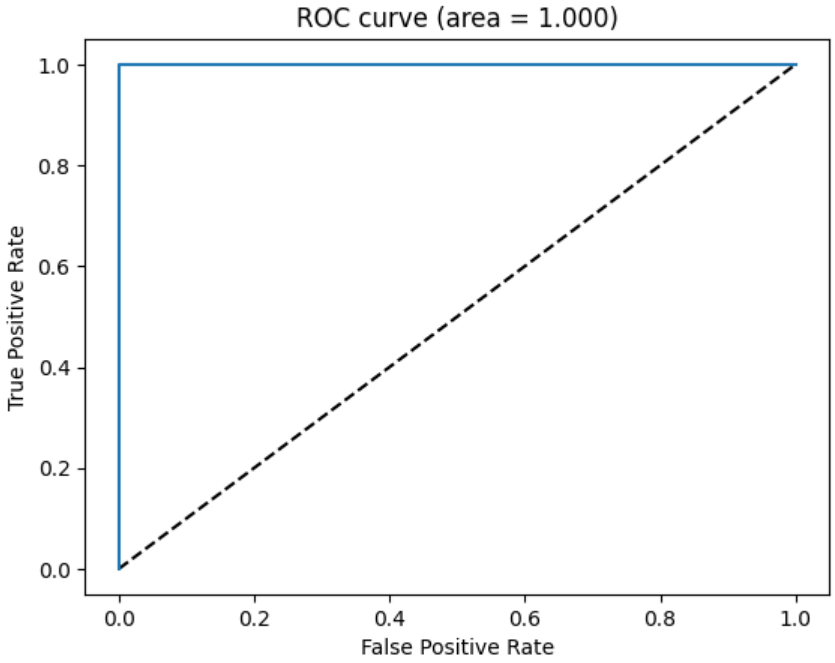
شکل 18. ماتریس آشفتگی شبکه اموزش داده شده روی داده‌های تست

بر اساس نتايج بدست آمده شبكه طراحي شده تمام داده ها را به درستي ارزيابي و طبقه بندي كرده است . اين داده ها داده هاي تست بوده و مدل بر اساس آنها آموزش نديده است پس نتيجه بدست آمده براي اين تعداد داده قابل قبول است.

ميتوانستيم در تفكيك داده ها داده هاي بيشتري را نيز مورد سنجش تست قرار بدهيم ولي زمان يادگيري روي داده Train طولاني مي‌شد با در نظر گرفتن تفكيك دقيقا به تعداد جدول اول پس صرفا تناسب رعايت شد. از طرفي نتايج روي داده هاي تست اموزش و Validation همگي گواه تفكيك درست اين تعداد داده بودند.

```
fpr, tpr, thresholds = roc_curve(y_test, probas.ravel())

area_under_curve = auc(fpr, tpr)

plt.plot([0, 1], [0, 1], 'k--')
plt.plot(fpr, tpr)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC curve (area = {:.3f})'.format(area_under_curve))
```

```
plt.show()
```



شکل 19. نمودار **ROC** شبکه **Fit** شده

همانطور که اشاره شد هدف این تحقیق حداکثر رساندن نسبت نمودار بوده که همینطور شده است.