به نام خدا

دانشکده مهندسی پزشکی دانشگاه آزاد

واحد تهران جنوب

درس بینایی ماشین

آقای دکتر اسلامی

دانشکده مهندسی پزشکی دانشگاه آزاد واحد تهران جنوب

نام و نام خانوادگی	**ندا سفندارمذ**
شماره دانشجویی	**۴۰۲۱۴۱۴۰۱۰۶۰۰۷**
تاریخ ارسال گزارش	**۱۴۰۲/۰۴/۲۴**

.

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

تشخیص بیماران مبتلا به کووید با استفاده از عکس ریه

جمع آوری داده و پیش پردازش تصاویر

همانطور که در بخش اول مقاله [1] بیان شده، افزایش داده[۱] رویکردی است که می‌تواند به طور قابل ملاحظه‌ای تعداد نمونه‌های داده در مجموعه داده را برای آموزش یک مدل افزایش دهد. در مورد مجموعه داده‌های تصویر، این رویکرد از عملیات پردازشی مانند انعکاس[۲]، چرخش[۳]، برش[۴] یا پر کردن[۵] برای افزایش داده ها استفاده می‌کند. در این مطالعه، دو عملیات پردازش تصویر، یعنی انعکاس و چرخش، برای افزایش داده استفاده شده‌اند. برای این منظور رویکرد زیر در نظر گرفته شد:

۱- در مرحله اول از افزایش داده، ۹۰ تصویر ایکس‌ری برای به دست آوردن ۹۰ تصویر جدید دیگر، منعکس(flipped) شده‌اند.

۲- در مرحله دوم، ۹۰ تصویر اصلی با زاویه ۹۰ درجه چرخانده شده‌اند تا ۹۰ تصویر دیگر به دست آید.

۳- در گام بعد تصاویر اصلی را با زاویه ۱۸۰ درجه چرخانده تا ۹۰ تصویر جدید دیگر به حاصل شود.

۴- در نهایت، ۹۰ تصویر اصلی با زاویه ۲۷۰ درجه چرخانده شده‌اند تا به ۹۰ تصویر دیگر برسیم.

این عملیات‌ها منجر به تولید یک مجموعه داده که شامل ۴۵۰ تصویر ایکس‌ری COVID-19 است، شدند.

Table 2: COVID-19 image count after data augmentation.

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 ۱ تصاویر مرتبط با عملیات های انجام شده برای افزایش داده در مقاله

۱ Data augmentation
۲ flipping
۳ rotating
۴ cropping
۵ padding

حال با توجه به عملیات انجام شده در مقاله [1] به منظور افزایش داده، اقدام به افزایش داده های داده شده برای این تمرین می‌کنیم. اما ابتدا باید داده ها را در محیط colab بارگذاری کنیم. برای این کار از قطعه کد های زیر بهره بردیم.

```
# check the file Dataset.zip exsits in colab
!ls -l .
```

```
total 92104
-rw-r--r-- 1 root root 94304102 Nov 23 06:42 Dataset.zip
drwxr-xr-x 1 root root     4096 Nov 21 14:24 sample_data
```

```
[ ] # unzip the Dataset.zip into dataset folder
    !unzip Dataset.zip -d dataset
```

```
Archive:  Dataset.zip
   creating: dataset/Dataset/
  inflating: dataset/Dataset/Test.json
  inflating: dataset/Dataset/Train.json
  inflating: dataset/Dataset/Validation.json
  inflating: dataset/Dataset/readme_pquad_en.txt
  inflating: dataset/Dataset/readme_pquad_fa.txt
   creating: dataset/Dataset/xray_dataset_covid19/
   creating: dataset/Dataset/xray_dataset_covid19/test/
   creating: dataset/Dataset/xray_dataset_covid19/test/COVID/
  inflating: dataset/Dataset/xray_dataset_covid19/test/COVID/ryct.2020200034.fig2.jpeg
```

شکل ۲ قطعه کد بارگذاری و **unzip** کردن فایل مربوط به داده های تمرین

```
[ ] # extract the data relevent to our works:
    !mv ./dataset/Dataset/xray_dataset_covid19/ ./dataset/
```

```
[ ] # chack the dataset folder:
    !ls -l ./dataset/
```

```
total 8
drwxr-xr-x 2 root root 4096 Nov 23 07:10 Dataset
drwxr-xr-x 4 root root 4096 Nov 11 11:10 xray_dataset_covid19
```

```
[ ] # remove the unnecessary files:)
    !rm -r ./dataset/Dataset/
```

```
[ ] # chack the dataset folder:
    !ls -l ./dataset/
```

```
total 4
drwxr-xr-x 4 root root 4096 Nov 11 11:10 xray_dataset_covid19
```

شکل ۳ قطعه کد مربوط به پاک سازی فایل های نامرتبط موجود در **Dataset.zip**

حال که داده ها را در محیط colab بارگذاری کردیم، اقدام به data augmentation یا افزایش داده (داده های آموزش) با توجه به عملیات های مورد استفاده در مقاله می‌کنیم. برای این منظور اقدام به تعریف تابع data_augment می‌کنیم. می‌توانید قطعه کد مربوط به آن را در شکل ۴ مشاهده کنید.

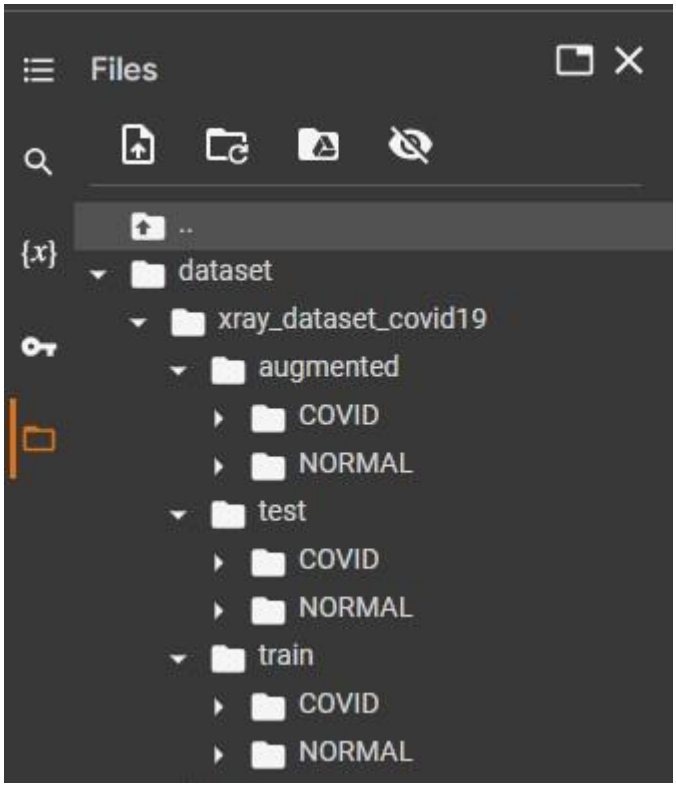
```python
import cv2
import os
import random

def augment_data(input_folder, output_folder):
    # Create the output folder if it doesn't exist
    if not os.path.exists(output_folder):
        os.makedirs(output_folder)

    # Loop through each image in the input folder
    for filename in os.listdir(input_folder):
          if filename.lower().endswith((".jpg", ".png", ".jpeg")):

            # Read the original image
            original_image = cv2.imread(os.path.join(input_folder, filename))
            # Save the original image
            save_path = os.path.join(output_folder, f"original_{filename}")
            cv2.imwrite(save_path, original_image)

            # Randomly choose to flip horizontally or vertically
            flip_direction = random.choice(["horizontal", "vertical"])
            # Flip the original image
            if flip_direction == "horizontal":
                flipped_image = cv2.flip(original_image, 1)  # 1 means horizontal flip
                save_path = os.path.join(output_folder, f"flipped_horizontal_{filename}")
            else:
                flipped_image = cv2.flip(original_image, 0)  # 0 means vertical flip
                save_path = os.path.join(output_folder, f"flipped_vertical_{filename}")
            # Save the Flipped image
            cv2.imwrite(save_path, flipped_image)

            # Rotate the original image
            for angle in [90, 180, 270]:
              if angle == 90:
                rotated_image = cv2.rotate(original_image, cv2.ROTATE_90_CLOCKWISE)
              elif angle == 180:
                rotated_image = cv2.rotate(original_image, cv2.ROTATE_90_CLOCKWISE)
                rotated_image = cv2.rotate(rotated_image, cv2.ROTATE_90_CLOCKWISE)
              elif angle == 270:
                rotated_image = cv2.rotate(original_image, cv2.ROTATE_90_CLOCKWISE)
                rotated_image = cv2.rotate(rotated_image, cv2.ROTATE_90_CLOCKWISE)
                rotated_image = cv2.rotate(rotated_image, cv2.ROTATE_90_CLOCKWISE)
              # Save the rotated image
              cv2.imwrite(save_path, flipped_image)

              # Rotate the original image
              for angle in [90, 180, 270]:
                if angle == 90:
                  rotated_image = cv2.rotate(original_image, cv2.ROTATE_90_CLOCKWISE)
                elif angle == 180:
                  rotated_image = cv2.rotate(original_image, cv2.ROTATE_90_CLOCKWISE)
                  rotated_image = cv2.rotate(rotated_image, cv2.ROTATE_90_CLOCKWISE)
                elif angle == 270:
                  rotated_image = cv2.rotate(original_image, cv2.ROTATE_90_CLOCKWISE)
                  rotated_image = cv2.rotate(rotated_image, cv2.ROTATE_90_CLOCKWISE)
                  rotated_image = cv2.rotate(rotated_image, cv2.ROTATE_90_CLOCKWISE)
                # Save the rotated image
                save_path = os.path.join(output_folder, f"rotated_{angle}_{filename}")
                cv2.imwrite(save_path, rotated_image)
```

شکل ۴ قطعه کد مربوط به تابع **data_augment**

همانطور که در این تابع مشاهده می‌کنید، ابتدا دایرکتوری مربوط به داده های اصلی و دایرکتوری خروجی برای ذخیره داده های augment شده را دریافت می‌کند. ابتدا بررسی می‌کند که آیا دایرکتوری خروجی وجود دارد یا خیر؟ اگر وجود نداشت، اقدام به ایجاد آن می‌کند. سپس تمامی داده های عکس که دارای پسوند ”.jpg“، ”.jpeg“ و ”.png“ هستند را ابتدا ذخیره می‌کند و سپس اقدام به انجام عملیات های مربوط به افزایش داده به ترتیب کرده و بعد از هر عملیات فایل جدید را با نام مناسب ذخیره می‌کند. برای عملیات انعکاس(flip)، از آنجایی که در مقاله اشاره ایی صریح به انعکاس عمودی یا افقی نشده، قبل از اقدام به انعکاس تصویر داده، نوع انعکاس با استفاده از یک تولیدکننده عدد تصادفی انتخاب می‌شود. چرخش ها نیز به ترتیب ۹۰، ۱۸۰ و ۲۷۰ درجه انجام می‌شوند و هر کدام ذخیره می‌شوند. نحوه فراخوانی این تابع را می‌توانید در شکل ۵ مشاهده کنید.

```
# Do the augmentation on train/COVID images and store them in augmented/COVID
augment_data(input_folder="dataset/xray_dataset_covid19/train/COVID/", output_folder="dataset/xray_dataset_covid19/augmented/COVID")
# Do the augmentation on train/NORMAL images and store them in augmented/NORMAL
augment_data(input_folder="dataset/xray_dataset_covid19/train/NORMAL/", output_folder="dataset/xray_dataset_covid19/augmented/NORMAL")

[ ] # check the result of augmentation
    ! ls -l ./dataset/xray_dataset_covid19/augmented/

    total 64
    drwxr-xr-x 2 root root 36864 Nov 23 07:11 COVID
    drwxr-xr-x 2 root root 24576 Nov 23 07:11 NORMAL
```

شکل ۵ قطعه کد مربوط به افزایش داده (**data augmentation**)



شکل ۶ وضعیت دایرکتوری های **colab** پس از انجام عملیات افزایش داده

حال داده های افزایش داده شده و داده های آزمون را با استفاده از قطعه کد شکل ۷ در برنامه بارگذاری می‌کنیم. همانطور که در خروجی شکل ۷ مشاهده می‌کنید، ۷۴۰ داده آموزش و ۴۰ داده آزمون داریم.

```
[ ]  import tensorflow as tf
     import os
     import numpy as np
     from matplotlib import pyplot as plt


     # Load the augmented data
     augmented_data = tf.keras.utils.image_dataset_from_directory(
         directory='dataset/xray_dataset_covid19/augmented/',
         image_size=(150, 150),
         )


     # Load the test data
     test_data = tf.keras.utils.image_dataset_from_directory(
         directory='dataset/xray_dataset_covid19/test/',
         image_size=(150, 150),
         )
```

```
Found 740 files belonging to 2 classes.
Found 40 files belonging to 2 classes.
```

شکل ۷ قطعه کد مربوط به بارگذاری داده های آموزش و آزمون در برنامه

همانطور که در شکل ۸ مشاهده می‌کنید برچسب COVID برابر با ۰ و برای NORMAL برابر ۱ است.

```
# Show the class names of augmented data
class_names = augmented_data.class_names
class_names
```

```
['COVID', 'NORMAL']
```

```
[ ]  # Show the class names of test data
     class_names = test_data.class_names
     class_names
```

```
['COVID', 'NORMAL']
```

```
class 0 => covid

class 1 => normal
```

شکل ۸ قطعه کد نمایش کلاس های مربوط به داده های بارگذاری شده و برچسب های آن ها

حال اقدام به rescale کردن داده های آموزش و آزمون می‌کنیم. برای داده های validation نیز از داده های آموزش زیر مجموعه‌ای با اندازه ۰,۲۵ داده های آموزش، ایجاد می‌کنیم و آن ها را نیر rescale می‌کنیم. قطعه کد مربوط به این عملیات را می‌توانید در مشاهده کنید.

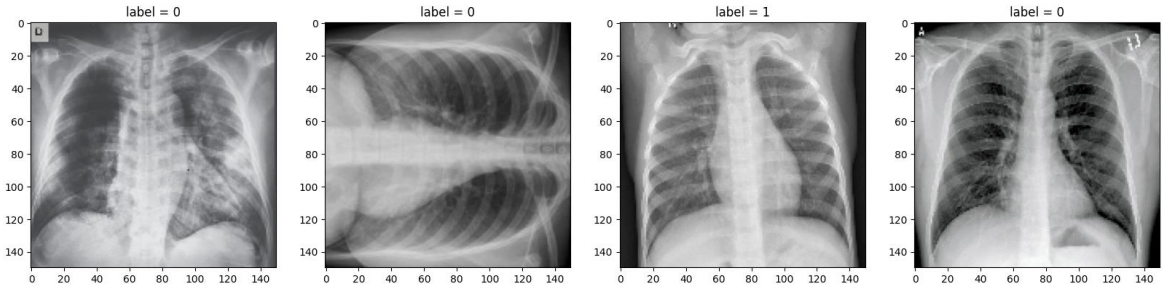
```
# Rescale the augmented data
augmented_data = augmented_data.map(lambda x, y: (x/255, y))

# Rescale the test data
test_data = test_data.map(lambda x, y: (x/255, y))
```

```
[ ] # Create validation data
validation_data = tf.keras.utils.image_dataset_from_directory(
    directory='dataset/xray_dataset_covid19/augmented/',
    image_size=(150, 150),
    validation_split=0.25,
    subset="validation",
    seed=1337
)

# Normalize the pixel values to be between 0 and 1
validation_data = validation_data.map(lambda x, y: (x / 255, y))
```

```
Found 740 files belonging to 2 classes.
Using 185 files for validation.
```

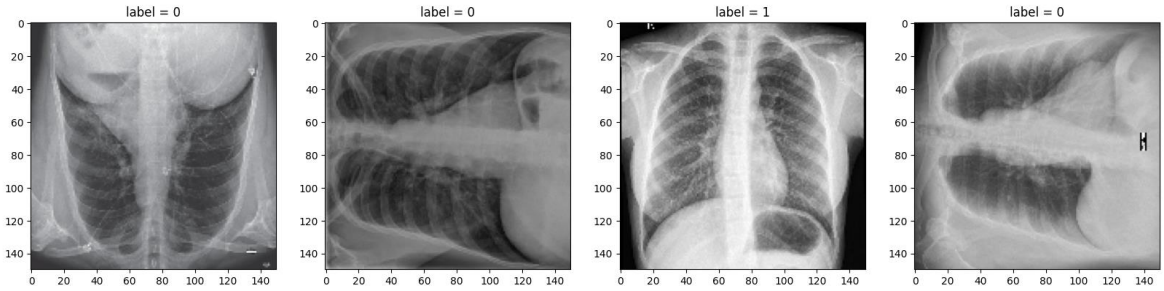
شکل ۹ قطعه کد مربوط به تولید داده های **validation** و **rescale** کردن داده ها

حال یک batch از داده های آموزش که افزایش داده شده اند را گرفته و اقدام به چاپ ۴ داده اول آن batch کردیم. همانطور که می‌دانید هر batch دارای ساختار (۳۲, ۱۵۰, ۱۵۰, ۳) می‌باشد. یعنی در هر batch به تعداد ۳۲ داده موجود است. می‌توانید ۴ تصویر اول مربوط به batch گرفته شده از داده های آموزش، به همراه برچسب هر کدام را در شکل ۱۰ مشاهده کنید.



شکل ۱۰ چاپ ۴ تصویر از **batch** مربوط به داده های آموزش افزایش داده شده

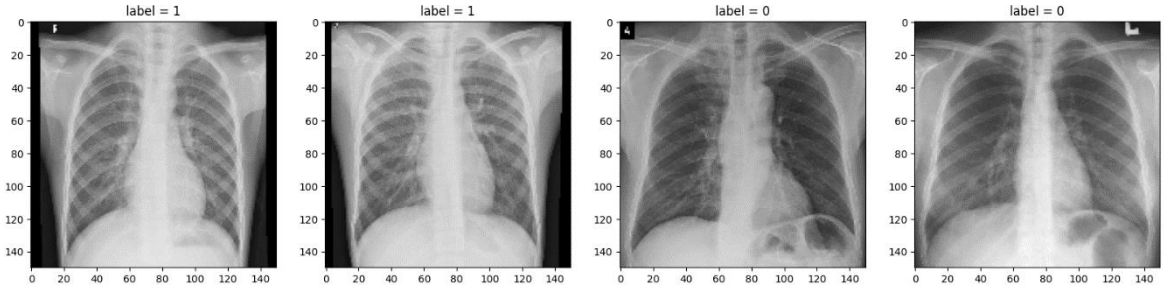
همچنین batch دیگری را از این داده ها در نظر گرفتیم و ۴ تصویر دیگر آن را نیز چاپ کردیم. می‌توانید تصاویر مربوط به آن را در شکل ۱۱ مشاهده کنید. همچنین برای مشاهده قطعه کد ها می‌توانید به فایل colab مراجعه کنید.



شکل ۱۱ تصاویر مربوط به ۴ داده از یک **batch** دیگر مربوط به داده های آموزش افزایش داده شده

همانطور که در شکل ۱۰ و شکل ۱۱ مشاهده می‌کنید داده های افزایش داده شده دارای انعکاس و چرخش می باشند.

حال اقدام به چاپ ۴ تصویر از یک batch مربوط به داده های آزمون کردیم. می‌توانید این تصاویر را درشکل ۱۲ مشاهده کنید.



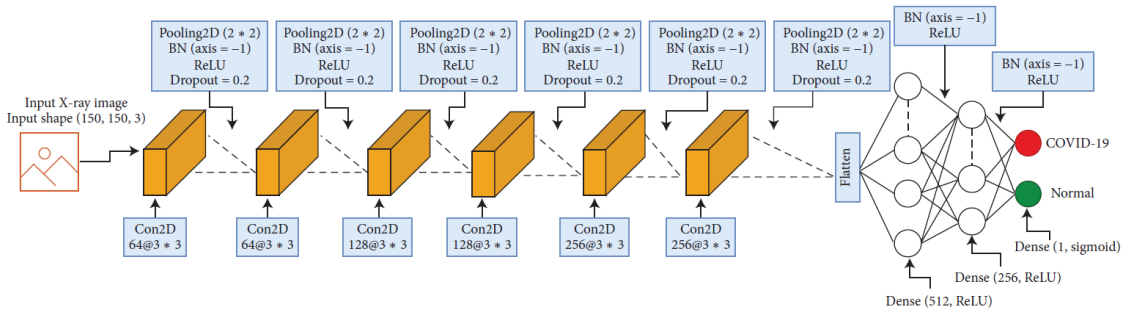
شکل ۱۲ چاپ ۴ تصاویر از داده های آزمون برای یک **batch**

همانطور که در مشاهده می‌کنید، داده های آزمون augment نشده‌اند. درحالی که برای آموزش و validation از داده های augment شده مربوط به داده های آموزش بهره بردیم.

آموزش شبکه

در این مقاله [1]، یک معماری شبکه عصبی پیچشی (CNN) برای وظایف دسته‌بندی دودویی معرفی شده است. مدل CNN شامل ۳۸ لایه است که شامل لایه‌های کانولوشن (Conv2D)، max pooling، dropout، تابع فعال‌سازی، نرمال‌سازی دسته‌ای (batch normalization)، flatten، و لایه‌های کاملاً متصل(fully connected layers) می‌شود. ابعاد تصویر ورودی به (۱۵۰، ۱۵۰، ۳) برای تصاویر RGB تعیین

شده است. لایه‌های کانولوشن از هسته ۳ × ۳ استفاده می‌کنند. پس از هر لایه Conv2D، تکنیک‌های مختلفی اعمال می‌شود، از جمله max pooling (با اندازه ۲ × ۲)، نرمال‌سازی دسته‌ای (با محور ۱-)، تابع فعال‌سازی ReLU، و لایه dropout (با نرخ ۲۰٪). خروجی نهایی، که از ۲۵۶ نورون در آخرین لایه Conv2D به دست می‌آید، از طریق لایه‌های max pooling، نرمال‌سازی دسته‌ای، فعال‌سازی، و لایه dropout عبور می‌کند. برای دسته‌بندی دودویی، مدل از تابع از دست دادن متقاطع دودویی (BCE) و تابع فعال‌سازی sigmoid استفاده می‌کند، زیرا تنها یک نود خروجی برای دسته‌بندی داده به یکی از دو کلاس موجود لازم است. بهینه‌ساز "Adam" برای تنظیم پویا ویژگی‌های وزن و نرخ یادگیری مدل، به منظور کاهش از دست دادن مدل استفاده می‌شود. معماری به صورت تصویری در شکل ۱۳ نمایش داده شده است.



شکل ۱۳ معماری شبکه

حال با توجه به معماری بیان شده و بهره از کتابخانه tensorflow اقدام به پیاده‌سازی مدل پیشنهادی که در شکل ۱۳ مشاهده کردید، می‌کنیم. قطعه کد مربوط به مدل را می‌توانید در شکل ۱۴ مشاهده کنید.

```
[ ]  import tensorflow as tf
     from tensorflow.keras.models import Sequential
     from tensorflow.keras.layers import Conv2D, MaxPooling2D, BatchNormalization, Dropout, Flatten, Dense

     # Create the CNN model
     model = Sequential()

     # Convolutional Layers
     # First layer
     model.add(Conv2D(64, (3, 3), input_shape=(150, 150, 3), activation='relu'))
     model.add(MaxPooling2D(pool_size=(2, 2)))
     model.add(BatchNormalization())
     model.add(Dropout(0.2))

     # Second layer:
     model.add(Conv2D(64, (3, 3), activation='relu', padding='same'))
     model.add(MaxPooling2D(pool_size=(2, 2)))
     model.add(BatchNormalization())
     model.add(Dropout(0.2))

     # Third layer:
     model.add(Conv2D(128, (3, 3), activation='relu', padding='same'))
     model.add(MaxPooling2D(pool_size=(2, 2)))
     model.add(BatchNormalization())
     model.add(Dropout(0.2))
```

```
[ ]
    # Forth layer:
    model.add(Conv2D(128, (3, 3), activation='relu', padding='same'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(BatchNormalization())
    model.add(Dropout(0.2))

    # Fifth layer:
    model.add(Conv2D(256, (3, 3), activation='relu', padding='same'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(BatchNormalization())
    model.add(Dropout(0.2))

    # Sixth layer:
    model.add(Conv2D(256, (3, 3), activation='relu', padding='same'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(BatchNormalization())
    model.add(Dropout(0.2))

    # Flatten Layer
    model.add(Flatten())

    # Fully Connected Layers
    model.add(Dense(512, activation='relu'))
    model.add(BatchNormalization())

    model.add(Dense(256, activation='relu'))
    model.add(BatchNormalization())

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

    # Create an instance of the Adam optimizer with the desired learning rate
    optimizer = tf.keras.optimizers.Adam(learning_rate=0.006)

    # Compile the model
    model.compile(loss='binary_crossentropy', optimizer=optimizer, metrics=['accuracy'])

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

شکل ۱۴ قطعه کد مربوط به پیاده‌سازی مدل پرسش ۳

حال که مدل مورد نظر پیاده سازی شد، با استفاده از قطعه کد شکل ۱۵ اقدام به آموزش مدل می‌کنیم.

```
    # Train the model using the augmented_data for training and validation_data for validation
    history = model.fit(
        augmented_data,
        epochs=50,
        validation_data=validation_data
    )

    # Evaluate the model on the test data
    test_loss, test_acc = model.evaluate(test_data)
    print(f'Test accuracy: {test_acc}')

    # Save the trained model
    model.save("./models/model.h5")
```

شکل ۱۵ قطعه کد مربوط به آموزش مدل پرسش ۳

همانطور که در مشاهده می‌کنید از داده های validation که در گام های قبل تولید کردیم، استفاده کردیم. همچنین تعداد epoch ها را مشابه مقاله برابر با ۵۰ قرار داده ایم. از آنجایی که مجموعه داده در نظر گرفته شده برای این تمرین با مقاله متفاوت است، به منظور نزدیک شدن به خروجی مقاله، همانطور که در شکل ۱۴ مشاهده می‌کنید، مقدار learning rate را برابر با ۰,۰۰۶ قرار دادیم. این مقدار را به صورت تجربی و با انجام آزمایش به دست آوردیم.

پس از آموزش مدل اقدام به رسم نمودار loss و accuracy برای داده های آموزش و validation به وسیله قطعه کد زیر کردیم.

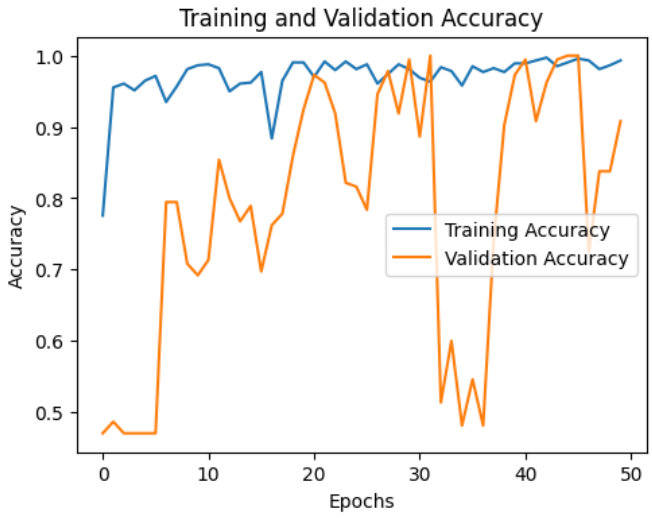
```python
# Plot training and validation loss
plt.figure(figsize=(12, 4))

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

# Plot training and validation 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('Epochs')
plt.ylabel('Accuracy')
plt.legend()

plt.show()
```

شکل ۱۶ قطعه کد مربوط به رسم نمودار **loss** و **accuracy** برای داده های آموزش و **validation**

شکل ۱۷ نمودار **loss** داده های آموزش و **validation**

همانطور که در مشاهده می‌کنید، نمودار loss مربوط به داده های validation و آموزش نزولی است و بعد از مدتی تقریبا به ۰ میل می‌کند و این بیانگر کاهش در مقدار خطا (Loss) مدل در طول فرآیند آموزش است. این نزول در Loss نشان می‌دهد که مدل با پیشرفت زمانی به تسلط بیشتری در تصمیم‌گیری می‌رسد و میزان خطای آن کاهش پیدا می‌کند. به طور کلی، اگر نمودار Loss به سمت صفر میل کند، این بیانگر موفقیت مدل در یادگیری و تعمیم قوانین و الگوهای موجود در داده‌ها است.

حال به نمودار accuracy برای داده های آموزش و validation می‌نگریم. همانطور که در شکل ۱۸ مشاهده می‌کنید، برای داده های آموزشی این مقدار تقریبا روند صعودی دارد و به یک میل می‌کند، اما برای داده های validation مقادیر تا ایپاک ۳۰ ام تقریبا صعوی هستند و به ۱ می‌رسند، اما پس از آن یک افت شدید می‌کند و به ۰,۵ می‌رسد. سپس تا ایپاک ۴۶ دوباره به ۱ صعود می‌کند و بعد از آن یک افت تا مقدار ۰,۷۶ دارد و در ادامه تا ایپاک ۵۰ روند صعودی دارد و دوباره به ۱ می‌رسد.



شکل ۱۸ نمودار **accuracy** برای داده های آموزش و **validation'**

نمودارهای accuracy برای داده‌های آموزش و اعتبارسنجی نشان‌دهنده عملکرد مدل در طول فرآیند آموزش است. روند صعودی در دقت (accuracy) نشان دهنده این است که مدل به طور کلی با داده‌های آموزش خوب هماهنگ شده و توانسته است الگوها و ویژگی‌های موجود در این داده‌ها را یاد بگیرد. در مورد نمودار مربوط به داده های validation، افت شدید دقت در ایپاک ۳۰ ممکن است بیانگر یک مشکل در یادگیری مدل باشد. این مشکل ممکن است به دلیل برازش بیش از حد (overfitting) به داده‌های آموزش باشد، که مدل به‌طور غیرمناسب با داده‌های جدید (اعتبارسنجی) هماهنگ شده است. صعود دوباره دقت در ایپاک ۴۶ ممکن است نشان از تصحیح مدل باشد، اما افت تا ایپاک ۵۰ ممکن است مشکلات دیگری را نشان دهد. روند صعودی دقت در ایپاک‌های بعدی نشان از بهبود یا تنظیم مدل است.

با توجه به این تحلیل، اهمیت افت و صعودهای دقت در دوره‌های آموزشی و اعتبارسنجی مشخص می‌شود. افت شدید ممکن است به مشکلاتی اشاره کند که نیاز به اقدام دارد، مانند استفاده از تکنیک‌های رفع برازش بیش از حد، تنظیم هایپرپارامترها، یا اصلاح معماری مدل.

ارزیابی شبکه استفاده شده در مقاله با ۶ لایه

حال با استفاده از قطعه کد شکل ۱۹ اقدام به ارزیابی شبکه پیاده سازی شده، می‌کنیم.

```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, recall_score, f1_score

# Make predictions on the test data
predictions = model.predict(test_data)

# Convert predictions to binary values (0 or 1)
binary_predictions = np.round(predictions)

# Extract true labels from the test data
true_labels = np.concatenate([y for x, y in test_data], axis=0)

# Flatten the predictions for calculating metrics
binary_predictions = binary_predictions.flatten()
true_labels = true_labels.flatten()

# Calculate metrics
accuracy = accuracy_score(true_labels, binary_predictions)
precision = precision_score(true_labels, binary_predictions)
recall = recall_score(true_labels, binary_predictions)
f1 = f1_score(true_labels, binary_predictions)

# Confusion matrix
conf_matrix = confusion_matrix(true_labels, binary_predictions)

# Specificity calculation
tn, fp, fn, tp = conf_matrix.ravel()
specificity = tn / (tn + fp)

# Display metrics
print(f"Accuracy: {accuracy:.4f}")
print(f"Precision: {precision:.4f}")
print(f"Recall (Sensitivity): {recall:.4f}")
print(f"F1 Score: {f1:.4f}")
print(f"Specificity: {specificity:.4f}")

# Plot confusion matrix
plt.imshow(conf_matrix, interpolation='nearest', cmap=plt.cm.Blues)
plt.title('Confusion Matrix')
plt.colorbar()
classes = ['COVID', 'NORMAL']
tick_marks = np.arange(len(classes))
plt.xticks(tick_marks, classes)
plt.yticks(tick_marks, classes)
plt.xlabel('Predicted labels')
plt.ylabel('Actual labels')
```

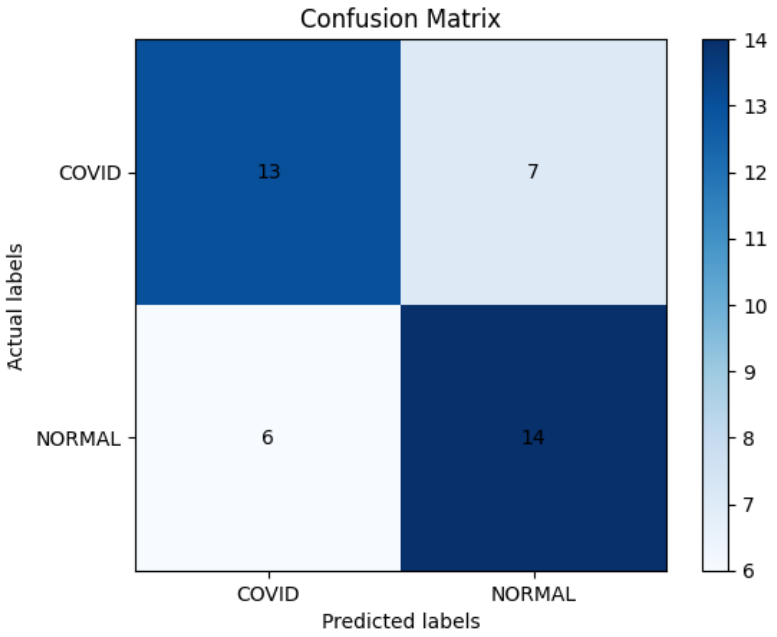
```
# Add text annotations
for i in range(len(classes)):
    for j in range(len(classes)):
        plt.text(j, i, str(conf_matrix[i, j]), ha='center', va='center')

plt.show()
```

```
2/2 [==============================] - 1s 13ms/step
Accuracy: 0.6750
Precision: 0.6667
Recall (Sensitivity): 0.7000
F1 Score: 0.6829
Specificity: 0.6500
```

شکل ۱۹ قطعه کد و خروجی مربوط ارزیابی مدل و رسم نمودار درهم ریختگی

همانطورکه در شکل ۱۹ مشاهده می‌کنید، خروجی دقت (Accuracy) برابر با ۰,۶۷۵۰ شده است. دقت میزان صحت کلی مدل را اندازه‌گیری می‌کند و هر دو صحت‌های واقعی مثبت و واقعی منفی را در مقایسه با همه پیش‌بینی‌ها در نظر می‌گیرد. در اینجا، مدل برای حدود ۶۷,۵٪ از موارد درست پیش‌بینی کرده است. خروجی Precision برابر با ۰,۶۶۶۷ است و دقت یا ارزش پیش‌بینی مثبت، دقت پیش‌بینی‌های مثبت انجام‌شده توسط مدل را نشان می‌دهد. یعنی حدود ۶۶,۶۷٪ از مواردی که مدل به عنوان مثبت پیش‌بینی کرده است، واقعاً موارد مثبت هستند. حساسیت یا recall، توانایی مدل در شناسایی صحیح موارد مثبت در میان همه موارد واقعی مثبت را اندازه‌گیری می‌کند. مقدار recall برای این مدل تحت داده های آزمون برابر با ۰,۷۰ است، یعنی مدل حدود ۷۰٪ از موارد مثبت واقعی را شناسایی کرده است. امتیاز F1 میانگین هندسی از دقت و حساسیت است و اندازه‌گیری متوازنی از دقت و حساسیت فراهم می‌کند. مقدار حدود ۰,۶۸۲۹ نشان‌دهنده توازن مناسبی بین دقت و حساسیت است. خصوصیت(Specificity) توانایی مدل را در شناسایی صحیح موارد منفی در میان همه موارد واقعی منفی را اندازه‌گیری می‌کند. خصوصیت حدود ۶۵٪ نشان‌دهنده این است که مدل حدود ۶۵٪ از موارد منفی واقعی را شناسایی کرده است.

به طور خلاصه، مدل عملکرد متوسطی دارد. همانطور که در ماتریس در هم ریختگی در شکل ۲۰ مشاهده می‌کنید، مدل ۶ تشخیص نادرست و ۱۴ تشخیص درست را برای حالت Normal دارد. از طرفی در مقابل مدل ۷ تشخیص نادرست و ۱۳ تشخیص درست برای حالت Covid دارد. همانطور که مشاهده می‌کنید، ظاهرا مدل بر روی داده های آموزش overfit شده است و برای داده های آزمون به مانند نتایج موجود در مقاله عمل نمی‌کند، به عبارت دیگر ممکن است تغییر مجموعه داده ها و یا انتخاب learning rate برابر با ۰,۰۰۶ موجب این نتایج شده باشد.



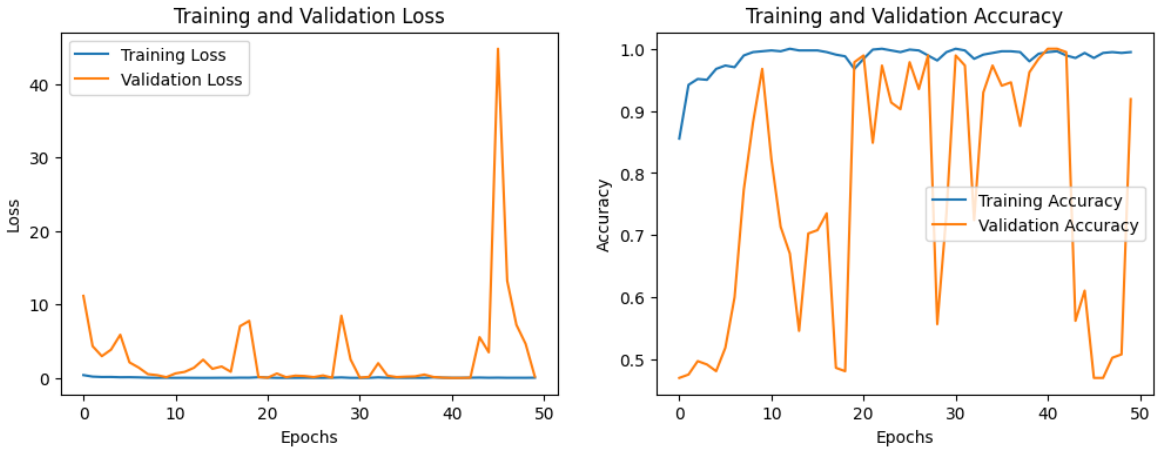
شکل ۲۰ ماتریس درهم ریختگی مدل پرسش ۳

ارزیابی کامل شبکه

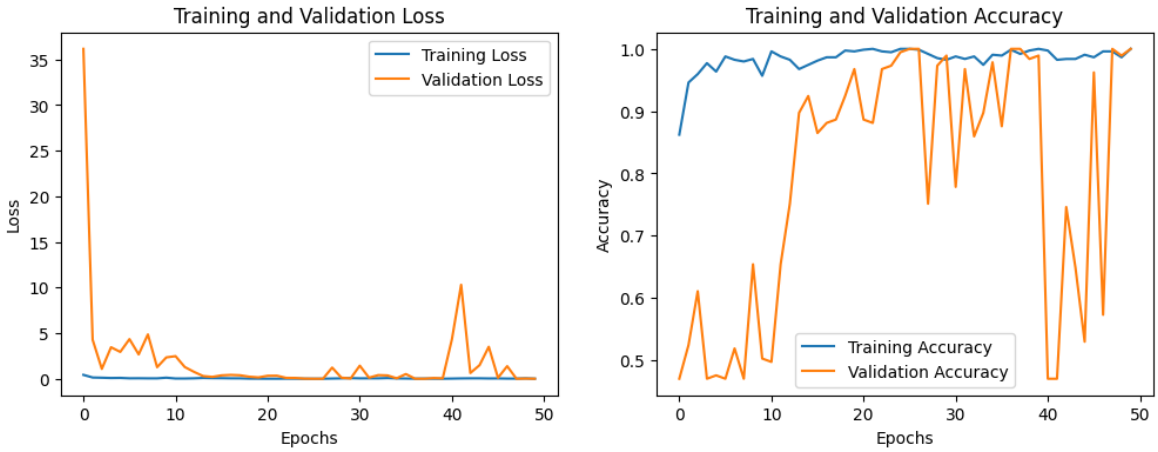
حال برای ارزیابی بیشتر این ساختار شبکه به مانند جدول ۶ در مقاله اقدام به تعریف مدل هایی با تعداد لایه کانولوشن ۱ ، ۲، ۳، ۴، ۵ کردیم و با داده های افزایش داده شده مشابه مدل مقاله که دارای ۶ لایه کانولوشن است، آن ها را آموزش دادیم. سپس با استفاده از داده های آزمون و validation اقدام به ارزیابی هرکدام از مدل ها کردیم[1]. نتایجی که مقاله برای این رویکرد به دست آورد را در شکل ۲۱ مشاهده می‌کنید.

Convolutional layer	Test data	Independent validation data
One Conv2D	0.715	0.455
Two Conv2D	0.940	0.895
Three Conv2D	0.995	0.957
Four Conv2D	0.995	0.980
Five Conv2D	0.995	0.995
Six Conv2D	1.000	0.995

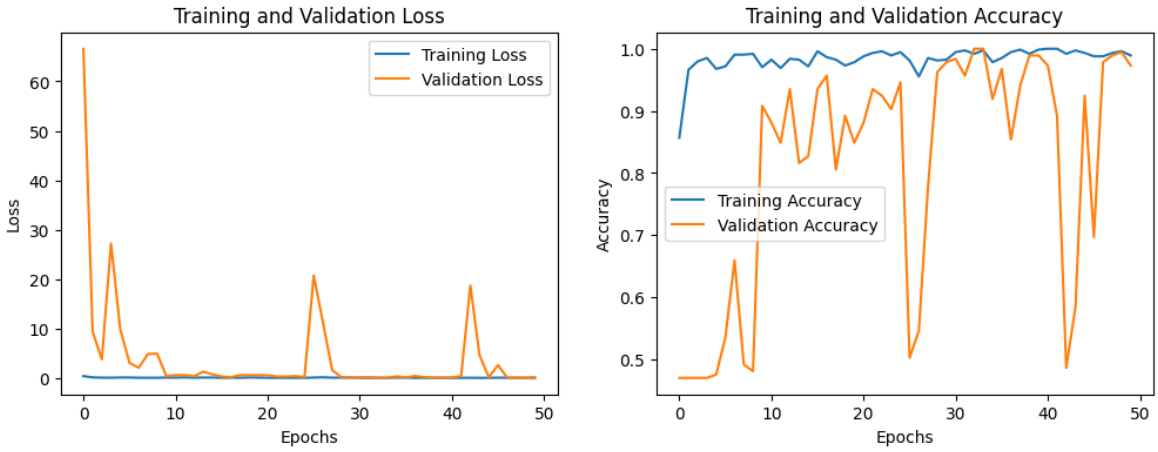
شکل ۲۱ نتایج ارزیابی مقاله برای مدل ها با تعداد متفاوت لایه کانولوشن



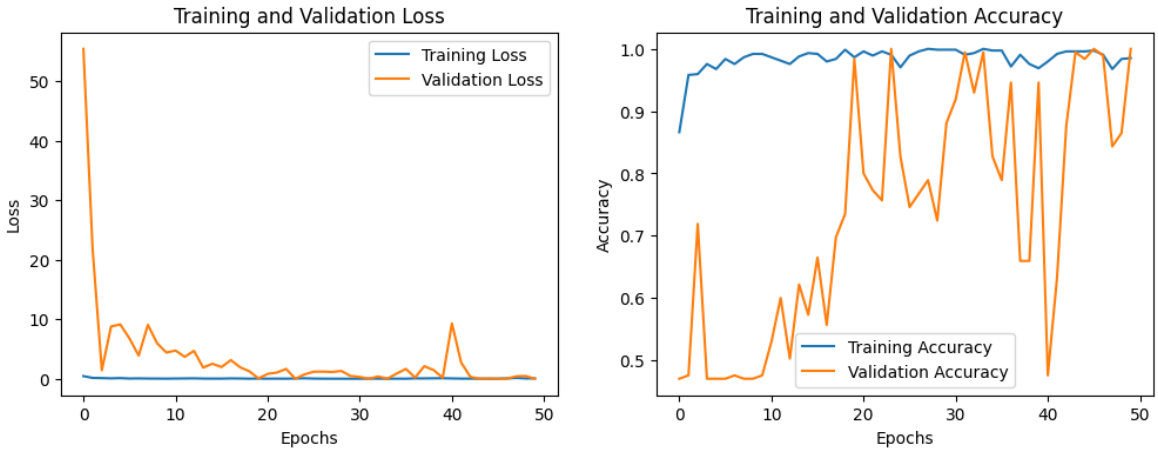
شکل ۲۲ نمودار **loss** و **accuracy** برای داده های آموزش و **validation** شبکه با یک لایه کانولوشن



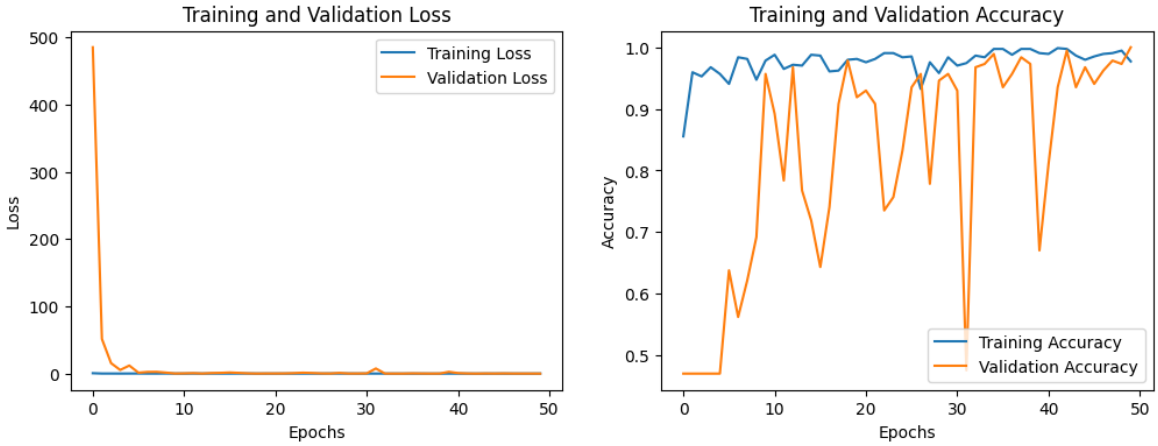
شکل ۲۳ نمودار **loss** و **accuracy** برای داده های آموزش و **validation** شبکه با دو لایه کانولوشن



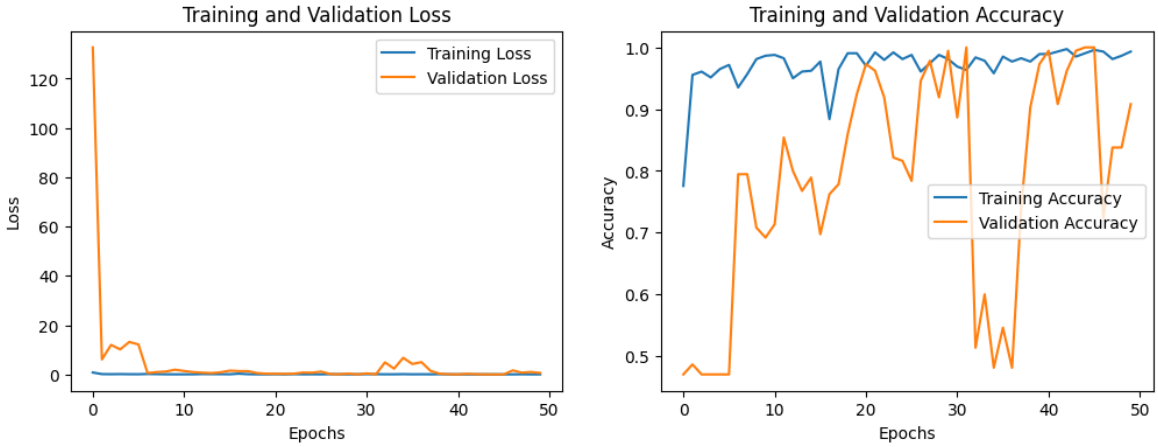
شکل ۲۴ نمودار **loss** و **accuracy** برای داده های آموزش و **validation** شبکه با سه لایه کانولوشن



شکل ۲۵ نمودار **loss** و **accuracy** برای داده های آموزش و **validation** شبکه با چهار لایه کانولوشن



شکل ۲۶ نمودار **loss** و **accuracy** برای داده های آموزش و **validation** شبکه با پنج لایه کانولوشن



شکل ۲۷ نمودار **loss** و **accuracy** برای داده های آموزش و **validation** شبکه با شش لایه کانولوشن(مدل مقاله)

حال با استفاده از داده های آزمون و validation اقدام به ارزیابی این شبکه با تعداد لایه های مختلف می‌کنیم. جدول نتایج را می‌توانید در شکل ۲۸ مشاهده کنید.

Table 6: Accuracy score with different numbers of CNN layers

Convolutional Layers	Test Accuracy	Validation Accuracy
One CONV2D	0.900	0.919
Two CONV2D	1.000	1.000
Three CONV2D	0.950	0.973
Four CONV2D	0.975	1.000
Five CONV2D	0.950	1.000
Six CONV2D	0.975	0.908

شکل ۲۸ نتایج ارزیابی شبکه با تعداد لایه های کانولوشن متفاوت

همانطور که مشاهده می‌کنید، استفاده از یک لایه Conv2D به دقت ۰,۹۰۰ در آزمون و دقت ۰,۹۱۹ در اعتبارسنجی منجر شد. مدل با دو لایه Conv2D عملکرد دقت کامل (۱,۰۰۰) را هم در آزمون و هم در اعتبارسنجی داشته است. مدل با سه لایه Conv2D با دقت ۰,۹۵۰ در آزمون و ۰,۹۷۳ در اعتبارسنجی و افزودن یک لایه Conv2D یا چهارم همچنان سطح دقت بالایی را حفظ کرده و دقت ۰,۹۷۵ در آزمون و دقت ۱,۰۰۰ در اعتبارسنجی را داشته است. مدل‌های با پنج و شش لایه Conv2D دقت پایین‌تری در آزمون دارند، اما مدل پنج همچنان دقت کامل (۱,۰۰۰) را در اعتبارسنجی دارند.

با توجه به نتایج حاصل شده می‌توان گفت که تعداد لایه‌های Conv2D به نظر می‌رسد که بر عملکرد مدل تأثیرگذار است و افزایش تعداد لایه‌ها همیشه به بهبود دقت نمی‌انجامد. برازش بیش از حد (Overfitting) ممکن است یک مسئله باشد، به ویژه در مدل‌هایی با تعداد زیادی لایه‌های Conv2D که اختلاف عملکرد بین مجموعه آزمون و اعتبارسنجی را نشان می‌دهد. انتخاب تعداد بهینه لایه‌های Conv2D وابسته به مجموعه داده و مسئله خاص است و نیاز به یک تعادل مناسب بین برازش بیش از حد و درک الگوهای معنی‌دار دارد. با این حال با توجه به تغییر مجموعه داده و انتخاب learning rate با مقدار ۰,۰۰۶ ظاهرا تحت این شرایط مدل با دو لایه کانولوشن عملکرد بهتری را دارا خواهد بود.

نوآوری

بهبودهایی که در کد انجام شده بر روی کاهش overfitting (برازش بیش از حد) و بهینه‌سازی ساختار مدل برای عملکرد بهتر و تعمیم‌پذیری بالاتر تمرکز دارد.

این کد یک مدل شبکه عصبی کانولوشنال (CNN) را تعریف می‌کند که تغییرات به شرح زیر است:

. (Global Average Pooling):۱

لایه Flatten با GlobalAveragePooling2D جایگزین شده است .این کار تعداد پارامترها را کاهش می‌دهد، در نتیجه باعث کاهش overfitting و بهبود عملکرد خواهد شد.

۲ .نرخ ریزش:(Dropout Rate)

نرخ ریزش در لایه‌های کانولوشنال به ۳.۰ تنظیم شده است تا از overfitting جلوگیری کند و در عین حال اطمینان حاصل شود که نورون‌های کافی برای یادگیری فعال هستند.

۳ .نرخ یادگیری:(Learning Rate)

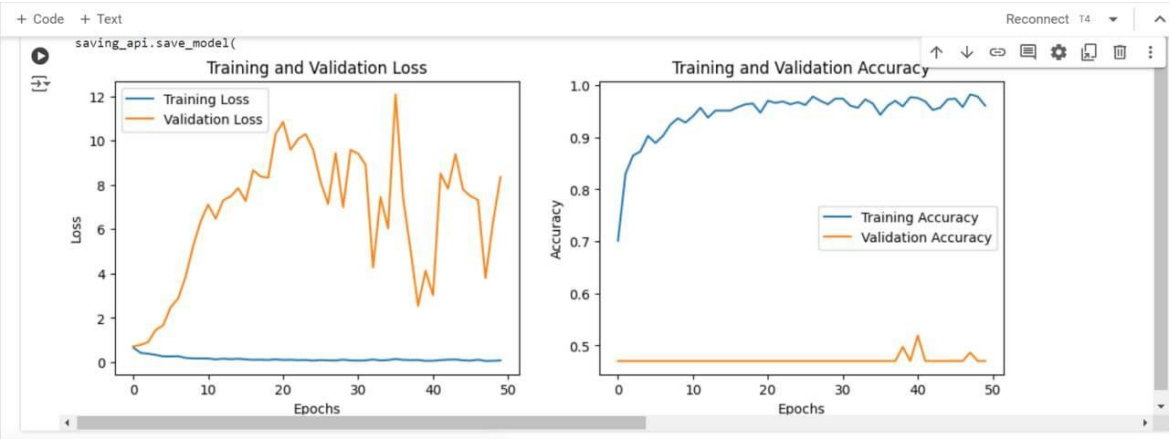
نرخ یادگیری بهینه‌ساز Adam به ۰۰۰۱.۰ کاهش یافته است تا همگرایی (convergence) بهتری در طول آموزش حاصل شود و به مدل اجازه دهد تا در طول زمان با دقت بیشتری یاد بگیرد.

```python
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, BatchNormalization, Dropout, Flatten, Dense, GlobalAveragePooling2D

# Create the CNN model
model = Sequential()

# Convolutional Layers with increased filters and better structure
# First layer
model.add(Conv2D(64, (3, 3), input_shape=(150, 150, 3), activation='relu', padding='same'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(BatchNormalization())
model.add(Dropout(0.3))

# Second layer
model.add(Conv2D(128, (3, 3), activation='relu', padding='same'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(BatchNormalization())
model.add(Dropout(0.3))

# Third layer
model.add(Conv2D(256, (3, 3), activation='relu', padding='same'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(BatchNormalization())
model.add(Dropout(0.3))

# Fourth layer
model.add(Conv2D(512, (3, 3), activation='relu', padding='same'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(BatchNormalization())
model.add(Dropout(0.3))

# Global Average Pooling Layer instead of Flatten
model.add(GlobalAveragePooling2D())
```

```
# Fully Connected Layers
model.add(Dense(512, activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.5))

model.add(Dense(256, activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.5))

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

# Create an instance of the Adam optimizer with the desired learning rate
optimizer = tf.keras.optimizers.Adam(learning_rate=0.0001)  # Lower learning rate for better convergence

# Compile the model
model.compile(loss='binary_crossentropy', optimizer=optimizer, metrics=['accuracy'])

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

```
+ Code  + Text                                                                    Reconnect T4
Epoch 41/50
24/24 [==============================] - 13s 438ms/step - loss: 0.0628 - accuracy: 0.9757 - val_loss: 3.0266 - val_ac
Epoch 42/50
24/24 [==============================] - 14s 420ms/step - loss: 0.0916 - accuracy: 0.9689 - val_loss: 8.5146 - val_accuracy: 0.4703
Epoch 43/50
24/24 [==============================] - 14s 407ms/step - loss: 0.1123 - accuracy: 0.9527 - val_loss: 7.8344 - val_accuracy: 0.4703
Epoch 44/50
24/24 [==============================] - 14s 416ms/step - loss: 0.1207 - accuracy: 0.9568 - val_loss: 9.3909 - val_accuracy: 0.4703
Epoch 45/50
24/24 [==============================] - 14s 401ms/step - loss: 0.0825 - accuracy: 0.9730 - val_loss: 7.7943 - val_accuracy: 0.4703
Epoch 46/50
24/24 [==============================] - 14s 451ms/step - loss: 0.0685 - accuracy: 0.9743 - val_loss: 7.5007 - val_accuracy: 0.4703
Epoch 47/50
24/24 [==============================] - 14s 463ms/step - loss: 0.1112 - accuracy: 0.9581 - val_loss: 7.3257 - val_accuracy: 0.4703
Epoch 48/50
24/24 [==============================] - 14s 385ms/step - loss: 0.0520 - accuracy: 0.9824 - val_loss: 3.7965 - val_accuracy: 0.4865
Epoch 49/50
24/24 [==============================] - 14s 467ms/step - loss: 0.0587 - accuracy: 0.9784 - val_loss: 6.3108 - val_accuracy: 0.4703
Epoch 50/50
24/24 [==============================] - 14s 436ms/step - loss: 0.0795 - accuracy: 0.9608 - val_loss: 8.3560 - val_accuracy: 0.4703
2/2 [==============================] - 1s 504ms/step - loss: 10.3922 - accuracy: 0.5000
Test accuracy: 0.5
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3103: UserWarning: You are saving your model as an HDF5 file via `model.save()`.
  saving_api.save_model(
```

شکل ۲۹ قطعه کد و خروجی مربوط ارزیابی مدل و رسم نمودار درهم ریختگی



شکل ۳۰-نمودار **loss** و **accuracy** برای داده های آموزش و **validation** شبکه با چهار لایه کانولوشن

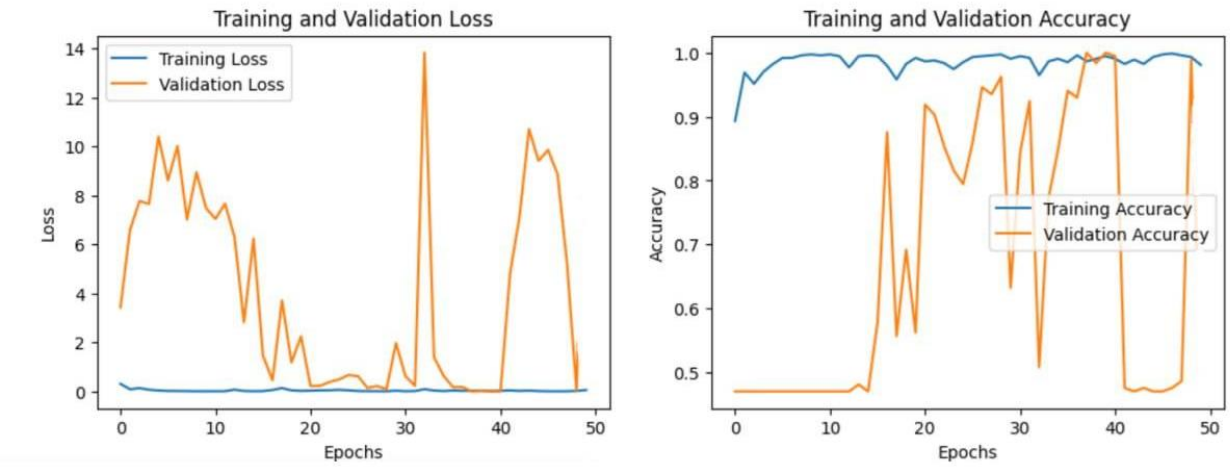
تغییرات ایجاد شده در برنامه اعمال گردید ولی با توجه به خروجی شکل های شماره۲۹ و ۳۰ مشاهده می شود که بهبودی حاصل نشده است. این نتیجه تا حدی قابل پیش بینی بود . زیرا قاعدتاً نویسندگان مقاله بهترین حالت معماری لایه ها را انتخاب نموه اند.

با توجه به محدودیت گوگل کولب در استفاده از GPU تغییرات بیشتر در کد و بررسی نتایج ممکن نبود.

مدل بهبود یافته

با توجه به محدودیت‌های رویکرد قبلی، تصمیم گرفتیم مدل را اصلاح کنیم که منجر به بهبود قابل توجه عملکرد شد، همانطور که در شکل نهایی به وضوح قابل مشاهده است .این پیشرفت‌ها اثربخشی مدل اصلاح‌شده را نشان می‌دهد.

مدل اصلاح‌شده نسبت به تکرار قبلی عملکرد بهتری را نشان می‌دهد .نتایج، کارایی اصلاحات اعمال‌شده را تأیید می‌کند .مدل اصلاح‌شده گواهی بر ماهیت تکرارشونده توسعه و بهینه‌سازی مدل است .نتایج را می توان در شکل ۳۱ مشاهده کرد.



شکل۳۱- نمودار تلفات و دقت برای داده های آموزشی و اعتبارسنجی شبکه با تغییر مدل از Sequential به Functional

بهبود عملکرد مدل تشخیص کووید-۱۹ با استفاده از شبکه‌های عصبی کانولوشن سبک

این پژوهش بر استفاده از شبکه‌های عصبی کانولوشن سبک (Lightweight CNNs) برای تشخیص خودکار بیماری کووید-۱۹ از تصاویر اشعه ایکس قفسه سینه تمرکز دارد .اگرچه مدل اولیه CNN به دقت بالایی (۹۹,۵ درصد) دست یافت، محققان محدودیت‌هایی را در رویکرد اولیه شناسایی کردند .سپس آن‌ها مدل را بهبود بخشیدند که منجر به افزایش قابل توجه عملکرد شد.

مشکل :تشخیص زودهنگام و دقیق کووید-۱۹ بسیار مهم است، اما در استفاده از داده‌های اشعه ایکس قفسه سینه برای مدل‌های یادگیری ماشین محدودیت‌هایی وجود دارد.

راه حل :محققان استفاده از شبکه‌های عصبی کانولوشن سبک را برای تشخیص خودکار کووید-۱۹ پیشنهاد می‌کنند.

پیش پردازش داده‌ها :تکنیک‌هایی مانند متعادل‌سازی داده‌ها، بررسی توسط متخصص و بزرگ‌نمایی داده‌ها، کیفیت داده‌ها را برای مدل بهبود بخشید.

توسعه مدل :یافتن تعداد بهینه لایه‌های کانولوشن برای جلوگیری از بیش‌برازش (overfitting) مهم است .محققان به این موضوع پرداختند و مدل را بیشتر بهبود بخشیدند.

این یافته‌ها اهمیت ارزیابی و اصلاح مداوم در حوزه یادگیری ماشین را برجسته می‌کند .با ارزیابی مکرر عملکرد و رفع نواقص، می‌توانیم قابلیت‌های مدل‌های خود را به تدریج ارتقا دهیم .عملکرد بهبود یافته مدل، تاییدی بر تلاش‌های ماست و این باور را تقویت می‌کند که استراتژی‌های بهینه‌سازی سیستماتیک می‌توانند منجر به پیشرفت‌های قابل توجهی در اثربخشی مدل شوند .ما متعهد به اصلاح بیشتر مدل‌های خود و کشف مسیرهای جدید برای بهبود هستیم.

منابع

[۱] A. A. a. R. F. a. M. A. a. A. A. a. A. Z. a. A. A. a. A. H. a. C. G. S. Reshi, “An Efficient CNN Model for COVID-19 Disease Detection Based on X-Ray Image Classification,” *Complexity*, 2021, p. 1–12.