به نام خدا

دانشگاه تهران

دانشکده مهندسی برق و کامپیوتر

درس شبکه‌های عصبی و یادگیری عمیق

تمرین دوم

پرسش ۱	نام و نام خانوادگی	علی خرم فر
	رایانامه	khoramfar@ut.ac.ir
پرسش ۲	نام و نام خانوادگی	محمدمهدی برقی
	رایانامه	mmbarghi@ut.ac.ir
	تاریخ ارسال پاسخ	۱۴۰۳/۰۲/۱۱

فهرست

فهرست تصاویر ..... ت

پرسش ۱- تشخیص آلزایمر با استفاده از تصویربرداری مغزی ۱

۱-۲ پیش‌پردازش تصویر ۱

۱-۳ داده‌افزایی ۳

۱-۳ پیاده‌سازی ۶

۱-۳ تحلیل نتایج ۸

نتایج پیاده‌سازی با نسبت ۰.۱ ..... ۱۰

۱-۳ مقایسه نتایج ۱۲

اثر تقسیم‌بندی داده‌ها به نسبت ۰.۳ ..... ۱۲

اثر تقسیم‌بندی داده‌ها به نسبت ۰.۵ ..... ۱۳

اثر Dropout ۱۵

اثر **Glorot Initialization** ۱۶

اثر تغییر معماری ۱۷

نتایج استفاده از معماری مدل Testing Model 1 ..... ۱۷

نتایج استفاده از معماری مدل Testing Model 2 ..... ۱۸

مقایسه مدل‌های Test با مدل پیشنهادی ۱۸

پرسش ۲. بررسی تاثیر افزایش داده بر عملکرد شبکه‌های کانولوشنی Fine-Tune شده ۲۰

۲-۱- آماده‌سازی اولیه ۲۰

۲-۲- پیش پردازش تصاویر ۲۰

Augmentation داده‌ها ۲۲

۲-۳- پیاده‌سازی ۲۴

۱-۲-۳- VGG16: ۲۵

۱-۲-۳- ResNet50: ۲۹

۴-۲- نتایج و تحلیل: ۳۳

فهرست تصاویر

شکل ۱ تصاویر ورودی پس از انجام پیش‌پردازش ۲
شکل ۲ توزیع داده‌های ورودی قبل از انجام داده‌افزایی ۲
شکل ۳تصاویر ورودی پس از انجام داده‌افزایی ۴
شکل ۴ توزیع داده‌های ورودی پس از انجام داده‌افزایی ۴
شکل ۵ نمودارهای Accuracy و Loss نسبت به Epoch برای داده‌های آموزشی و ارزیابی ۱۰
شکل ۶ نمودار ROC برای طبقه‌بندی تصاویر ۱۱
شکل ۷ گزارش طبقه‌بندی ۱۱
شکل ۸ نمودارهای Accuracy و Loss نسبت به Epoch برای نسبت ۰.۳ ..... ۱۲
شکل ۹ گزارش طبقه‌بندی برای نسبت ۰.۳ ..... ۱۳
شکل ۱۰ نمودارهای Accuracy و Loss نسبت به Epoch برای نسبت ۰.۵ ..... ۱۳
شکل ۱۱ گزارش طبقه‌بندی برای نسبت ۰.۵ ..... ۱۴
شکل ۱۲ نمودار ROC پس از تغییر نسبت تقسیم داده‌ها ۱۴
شکل ۱۳ نمودارهای Accuracy و Loss نسبت به Epoch پس از Dropout ۱۵
شکل ۱۴گزارش طبقه‌بندی پس از Dropout ۱۵
شکل ۱۵ نمودارهای Accuracy و Loss نسبت به Epoch پس از وزن‌دهی اولیه یک ۱۶
شکل ۱۶نمودارهای Accuracy و Loss نسبت به Epoch مدل Testing Model 1 ..... ۱۷
شکل ۱۷ گزارش طبقه‌بندی مدل Testing Model 1 ..... ۱۷
شکل ۱۸نمودارهای Accuracy و Loss نسبت به Epoch مدل Testing Model 2 ..... ۱۸
شکل ۱۹ گزارش طبقه‌بندی مدل Testing Model 2 ..... ۱۸

پرسش ۱– تشخیص آلزایمر با استفاده از تصویربرداری مغزی

برای پاسخ به این پرسش یک مقاله ارائه شده که با کمک روش‌های مبتنی بر شبکه عصبی به حل مسئله تشخیص بیماری آلزایمز با کمک پردازش تصاویر اسکن شده از مغز انسان می‌پردازیم. در این پرسش ۲ کلاس داریم که AD و MCI هستند و تصاویر به صورت لیبل‌شده هستند و در پوشه‌های مجزا قرار داده شده‌اند.

به صورت کلی ۹۶۵ تصویر از کلاس AD و ۶۸۹ تصویر از کلاس MCI موجود است و توزیع متوازنی از تصاویر ارائه نشده است. پس اگه به صورت تصادفی از بین کل ۱۶۵۴ تصویر انتخاب کنیم، احتمال بیشتر وجود دارد که تصویر انتخاب شده از کلاس AD باشد.

حال به ترتیب قسمت‌های مختلف این پرسش را بررسی می‌کنیم.

۲-۱ پیش‌پردازش تصویر

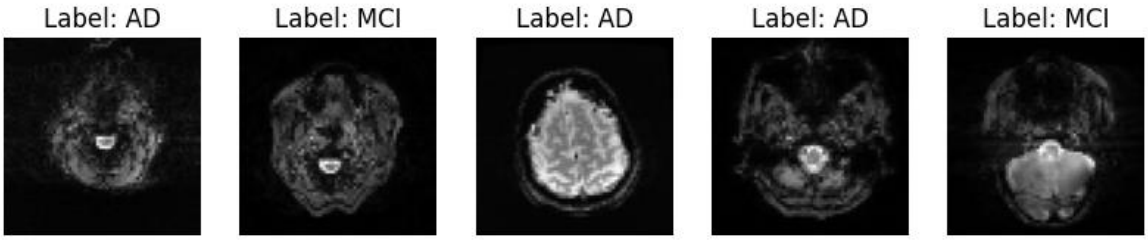
برای این مسئله با توجه به اینکه در ابتدا قصد داشتیم از ۲ روش آن را حل کنیم، ۲ روش مختلف پیش‌پردازش انجام شد. روش اول به صورت دستی انجام شد و روش دوم به هنگام تعریف دیتاست برای استفاده از پکیج PyTorch پیش‌پردازش داده‌ها نیز انجام شد. ولی باتوجه به اینکه در بخش ۳-۱ تمرین نیاز بود که داده‌ها به صورت واقعی زیاد شوند و آن‌ها را در درایو ذخیره کنیم این مرحله به صورت دستی هم انجام شد.

```
[ ] X = []
    y = []


    for class_index, class_name in enumerate(classes):
        class_dir = os.path.join(data_dir, class_name)

        for filename in tqdm(os.listdir(class_dir), desc=f'Preprocessing {class_name}'):
            file_path = os.path.join(class_dir, filename)
            img = cv2.imread(file_path)
            if img is not None:
                img = cv2.resize(img, (64, 64))  # Resize image
                img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)  # Convert to grayscale
                X.append(img)
                y.append(class_index)

    X = np.array(X, dtype=np.float32) / 255.0  # Normalize
    y = np.array(y, dtype=np.int64)

    # Print the shape of the arrays
    print('X shape:', X.shape)
    print('y shape:', y.shape)

    Preprocessing MCI: 100%|██████████| 689/689 [00:05<00:00, 120.56it/s]
    Preprocessing AD: 100%|██████████| 965/965 [00:09<00:00, 98.00it/s] X shape: (1654, 64, 64)
    y shape: (1654,)
```

برای انجام پیش‌پردازش به صورت دستی ۳ کار انجام شد. اول اینکه باتوجه به یک بخش از مقاله که نتیجه خوبی دریافت شده بود قصد شد که تصاویر به اندازه ۶۴ در ۶۴ تبدیل شده و سپس شبکه با ورودی این اندازه از تصاویر آموزش ببینند. دوم اینکه تصاویر ارائه شده در تمرین ۳ کاناله RGB بودند و با تبدیل

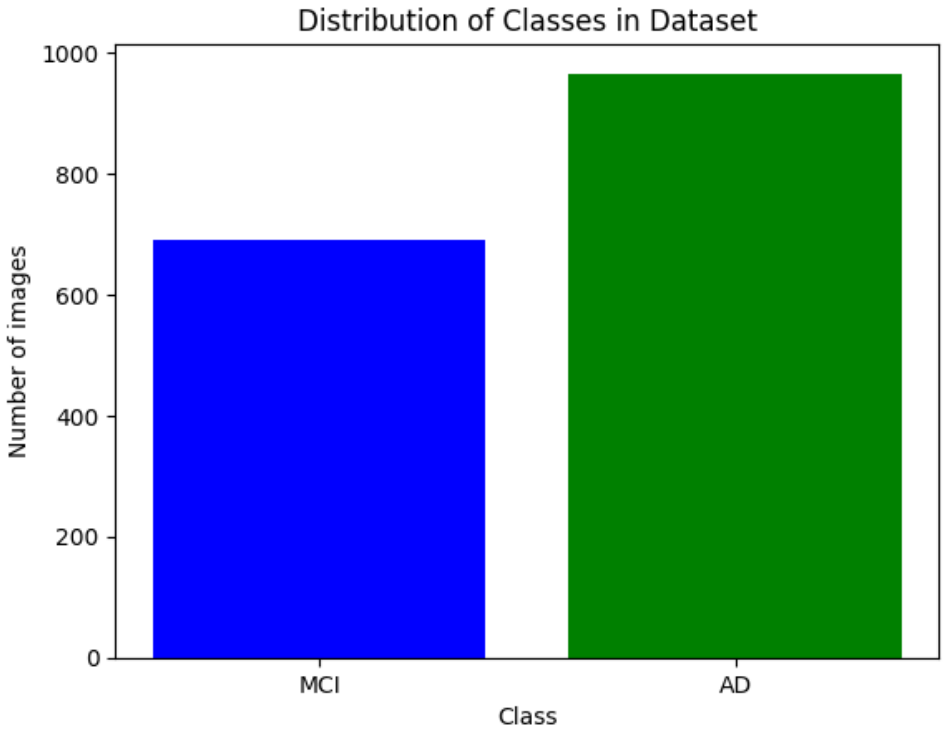
آن‌ها به ۱ کانال سیاه و سفید پیش‌پردازش دوم نیز انجام شد. و سوم اینکه باتوجه به اینکه تصاویر در بازه ۰ تا ۲۵۵ هستند نیاز هست آن‌ها را نرمال بین ۰ و ۱ کنیم که این مورد هم با تقسیم مقادیر آرایه بر ۲۵۵ انجام شد.

پس از انجام این پیش‌پردازش‌ها نتیجه را بر روی درایو ذخیره کردیم. تصویر زیر خروجی ۵ تصویر تصادفی هستند که پیش‌پردازش روی آن‌ها انجام شده است.



شکل ۱ تصاویر ورودی پس از انجام پیش‌پردازش

توزیع تصاویر قبل از مرحله بعد نیز در اینجا خروجی گرفته شد که نتیجه را به صورت زیر مشاهده می‌کنیم:



شکل ۲ توزیع داده‌های ورودی قبل از انجام داده‌افزایی

۳-۱ داده‌افزایی

در این قسمت از مسئله قصد داریم که داده‌ها را برای آموزش افزایش دهیم. برای این کار از تکنیک‌های مختلفی استفاده می‌شود. قصد ما در این تمرین پیاده‌سازی روش‌هایی است که در مقاله مرجع استفاده شده‌است. در این مقاله برای داده‌افزایی به موارد زیر اشاره شده است:

> The experiments in this section are carried out to examine the data augmentation effect on the classification process. Four images are generated from a single image. The total number of images in the dataset, after augmentation, reaches 211,655. Data augmentation factors are horizontal flipping, shearing with a range of 0.2, shifting with a range of 0.1, rotating with 15 degrees, and zooming with a range of 0.2. Table 8 shows the effect of different dataset sizes and batch sizes with / without applying Dropout. The image size, in this case, is (64, 64). Dataset split sizes range

در این قسمت از تمرین برای هر تصویر ۴ تصویر جدید خلق شده و علاوه بر این تصویر اصلی نیز در درایو ذخیره می‌شود. یعنی تعداد داده‌های جدید ۵ برابر می‌شوند. پس از انجام آزمایشات فراوان از مقادیر زیر برای داده‌افزایی استفاده شد:

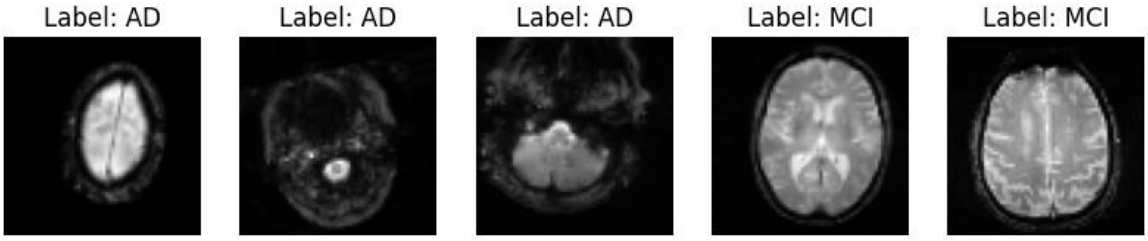
1-3 Data augmentation

```
data_gen = ImageDataGenerator(
    rotation_range=15,
    width_shift_range=0.1,
    height_shift_range=0.1,
    shear_range=0.2
)
```

در Shear Range مختصات فضایی تصویر دچار تغییر می‌شود. ابعاد جدید می‌توانند از ترکیب خطی ابعاد سابق یا صرفا تغییر زاویه ابعاد سابق حاصل شوند. در Width و Height Shift Range، نسبت به تصویر اصلی بخشی از ابعاد عرضی و طولی تصویر حذف می‌شود. در Rotation Range زاویه تصویر دچار تغییر می‌گردد. Horizontal Flip نیز تصویر را افقی معکوس می‌کند. Zoom Range نیز روی تصویر زوم اعمال می‌کند. البته به جز مورد اول، هر یک از موارد فوق به احتمال مشخصی شانس اعمال شدن دارد. مثلا در Rotation Range، زاویه تصویر به صورت شبه تصادفی از ۱۵- تا ۱۵+ تغییر پیدا می‌کند یا در Horizontal Flip با احتمال مشخصی ممکن است تصویر به صورت افقی معکوس شود یا نشود.

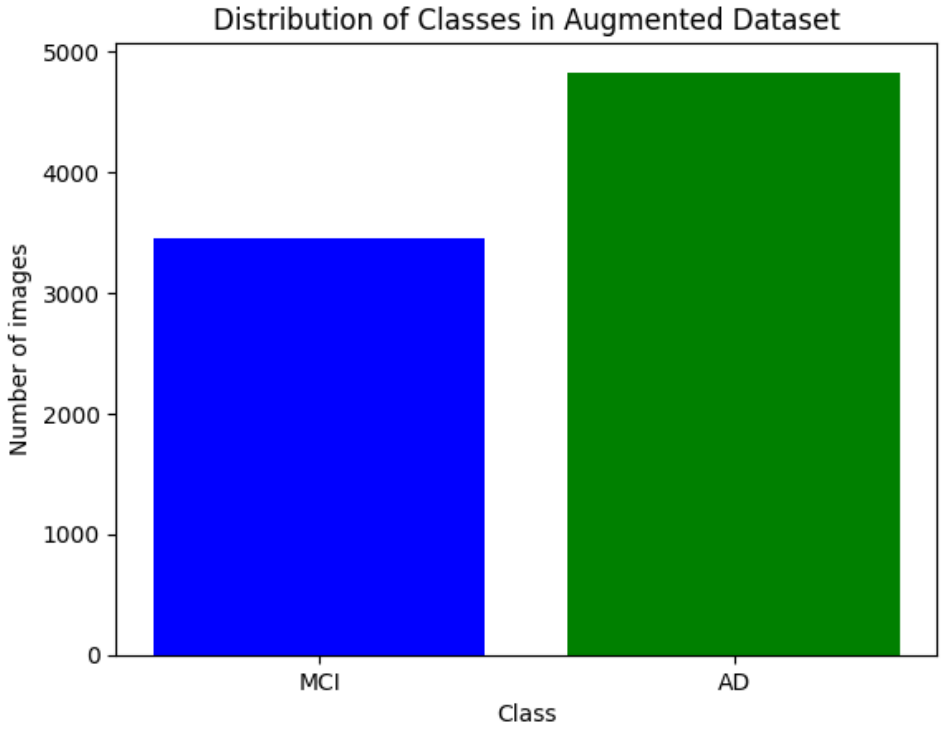
پس از تولید هر عکس آن‌ها را به صورت مجزا بر اساس کلاس در پوشه‌ی Augmented_Data در کنار پوشه اصلی ذخیره کردیم تا بتوانیم از آن‌ها برای مرحله آموزش استفاده کنیم.

همانطور که در قسمت قبلی بررسی شد در داده‌های این مسئله به اندازه برابر از هر کلاس داده وجود ندارد و تعداد داده‌های کلاس AD بیشتر هستند. همین مسئله باعث می‌شود در تحلیل برخی نتایج دچار اشتباه محاسباتی شویم که باید به آن دقت کنیم. پس از انجام عمل داده‌افزایی به صورت تصادفی ۵ تصویر خروجی گرفته شد که در شکل زیر آن‌ها را مشاهده می‌کنیم و اثر هرکدام از مواردی که توضیح داده‌شد مشاهده می‌شود:



شکل ۳تصاویر ورودی پس از انجام داده‌افزایی

برای اینکه مطمئن شویم از هر تصویر ۵ تصویر ذخیره شده بار دیگر نمودار توزیع برای هر کلاس را رسم می‌کنیم:



شکل ۴ توزیع داده‌های ورودی پس از انجام داده‌افزایی

باتوجه به شکل بالا قابل انتظار بود و توزیع نامتناسب هردو کلاس در عمل داده‌افزایی نیز حفظ شد. پیش از انجام این عمل ۹۶۵ تصویر از کلاس AD و ۶۸۹ تصویر از کلاس MCI وجود داشت که با ۵ برابر شدن آن‌ها تعداد تصاویر کلاس AD به تعداد ۴۸۲۵ و تصاویر کلاس MCI به تعداد ۳۴۴۵ افزایش یافت که در شکل بالا این موارد قابل مشاهده است.

```
Dataset ImageFolder
    Number of datapoints: 8270
    Root location: /content/gdrive/My Drive/Colab Notebooks/NNDL HW2/Augmented_Data
    StandardTransform
```

باید به این مورد توجه داشته باشیم باتوجه به اینکه پیاده‌سازی در PyTorch انجام شد میتوانستیم عمل داده‌افزایی را به صورت on the fly انجام دهیم. در این روش بر خلاف روش نهایی انجام شده در تمرین داده‌ها به صورت یکجا تبدیل نمی‌شوند و بلکه هر دسته داده که برای آموزش انتخاب می‌شود، تبدیل‌های مذکور در داده‌افزایی بر روی آن‌ها اعمال می‌شود. ولی در پیاده‌سازی این تمرین باتوجه به خواسته سوال از این روش استفاده نشد هرچند که این روش آسان تر بوده و منابع کمتری نیاز دارد. باتوجه به اینکه داده‌های ورودی مجددا دریافت شدند برخی اعمال پیش‌پردازشی مثل نرمال سازی و تبدیل به ۱ کانال مجددا در ساخت دیتاست PyTorch استفاده شد.

برای تقسیم‌بندی داده‌ها از متد random_split از پکیج PyTorch استفاده شد. باتوجه به تقسیم‌بندی‌های مختلف ارائه شده در این مقاله، نسبت ۰.۱ دقت بالاتری نسبت به بقیه موارد داشت و از رویکرد زیر برای تقسیم‌بندی استفاده شد. داده‌ها به صورت تصادفی تقسیم می‌شوند.

> The third layer in the framework is the cross-validation strategy used to train the CNN. Data are divided, randomly, into three sets; training set, validation set, and testing set. The whole dataset is divided into (95%) training set and (5%) testing set. The training set is further divided into (90%) training and (10%) validation sets. The main objective of cross-validation is obtaining the best values for training parameters to avoid overfitting.

در این رویکرد از ۹۵ درصد داده‌ها برای آموزش و ۵ درصد برای آزمون استفاده می‌شود. از آن قسمت ۹۵ درصدی داده آموزشی نیز، ۹۰ درصد آن برای خود آموزش و ۱۰ درصد ارزیابی استفاده شد.

```
test_size = int(0.3 * total_size)  # 30% for testing
train_size = total_size - test_size  # 70% for training

train_dataset, test_dataset = random_split(dataset, [train_size, test_size])

train_train_size = int(0.9 * train_size)  # 90% of training set
val_size = train_size - train_train_size  # Remaining for validation
```

۳-۱ پیاده‌سازی

نحوه عملکرد Glorot Initialization

این رویکرد یکی از روش‌های وزن‌دهی اولیه مدل است و با استفاده از این روش، مقدارهای اولیه وزن‌ها بر اساس میانگین صفر و یک واریانس خاص انجام می‌شود. به طور کلی این روش از ۲ توزیع نرمال و یکنواخت برای مقداردهی استفاده می‌شود که نوع یکنواخت آن در این مقاله اشاره شده است. برای پیاده‌سازی آن از متد xavier_uniform_ استفاده شد.

در مرحله آموزش به صورت زیر پیاده‌سازی شد:

```
# Glorot initialization
def init_weights(m):
    if type(m) == nn.Conv2d or type(m) == nn.Linear:
        xavier_uniform_(m.weight)
        if m.bias is not None:
            m.bias.data.fill_(0.01)

model.apply(init_weights)
```

بر اساس توضیحات مقاله با کمک این روش برای مقداردهی اولیه وزن‌ها، سرعت یادگیری افزایش یافته و باعث افزایش سرعت همگرایی و طبعا افزایش دقت مدل خواهندشد.

تابع هزینه

در این مسئله برای تابع هزینه از CrossEntropy استفاده شد. این تابع یک معیار استاندارد برای بدست‌آوردن مقدار اختلاف بین مقدار پیش‌بینی شده توسط مدل و مقدار واقعی است. این تابع برای مسائل طبقه‌بندی که در این مقاله نیز همین مسئله مطرح شده است بسیار کمک‌کننده است و به همگرایی سریع‌تر شبکه کمک می‌کند. در این مقاله به این مورد هم اشاره شده که استفاده از این تابع هزینه به جلوگیری از مشکل محوشدگی گرادیان یا Vanishing کمک می‌کند.

Optimizer و نرخ یادگیری

از تابع Adam برای Optimizer استفاده شد. همانطور که در این مقاله اشاره شده است ، Adaptive Momentum Estimation یا تابع Adam برای حل مسائلی از جنس این مقاله که تصاویر پزشکی مثلا مغزو اعصاب هستند، به همگرایی سریع‌تر شبکه در شرایط دارای نویز کمک می‌کند. برای این تابع از نرخ یادگیری‌های متفاوتی طبق مقاله استفاده شد ولی بهترین دقت با نرخ ۰.۰۰۱ حاصل شد.

پیاده‌سازی مدل‌ها

در این قسمت از تمرین باتوجه به توضیحات ارائه شده در مقاله و جدول ۹ ، ۳ مدل اصلی پیاده‌سازی شد. مدل اول مدل پیشنهادشده در مقاله است و ۲ مدل بعدی مدل‌های تست هستند. در ادامه به توضیح برخی از اجزای این مدل‌ها می‌پردازیم.

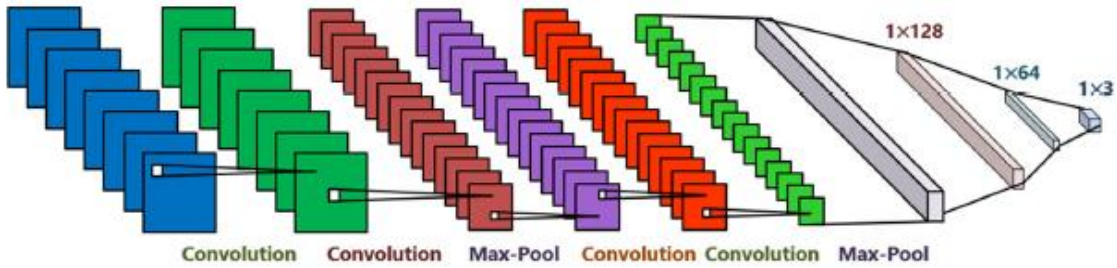
لایه‌ی کانولوشن: در معماری ارائه شده در مقاله،۴ لایه کانولشون وجود دارد. این لایه‌ها برای استخراج ویژگی‌های داده‌ها مورد استفاده قرار می‌گرند. این لایه‌ها ساختار داده‌ی ورودی را تغییر نمیدهند و ارتباط بین پیکسل‌های همسایه را مورد توجه قرار میدهند.

لایه‌ی Maxpooling: لایه‌ی maxpooling یک نمونه‌برداری بر روی ویژگی‌های استخراج شده در بخش قبل انجام میدهد و از این طریق redundancy داده‌ها را کاهش میدهد. ترتیب این لایه بدون تغییر ویژگی‌های مفید استخراج شده در بخش قبل، ابعاد داده و پیچیدگی محاسبات را کاهش میدهد. این لایه سرعت آموزش مدل و قدرت generalization آن را افزایش میدهند تا از بیش‌برازش جلوگیری شود.

لایه‌ی Fully connected: در این لایه هر نود، به تمام نودهای گره‌ی بالاتر متصل میشود. در شبکه‌ی ارائه شده توسط مقاله خروجی لایه‌ی قبلی باید flat شود و به عنوان ورودی لایه‌ی Fully connected قرار داده شود.

Relu: مقاله به عنوان activation function از تابع relu استفاده میکند. این تابع مقدار ورودی را میگیرد، اگر کمتر از صفر بود، صفر برمیگرداند و در غیر اینصورت همان مقدار ورودی را به عنوان خروجی برمیگرداند.

شکل کلی معماری به صورت زیر است، هرچند که تفاوت‌هایی دارد. در مقاله ۳ کلاس داریم ولی در اینجا ۲ کلاس:



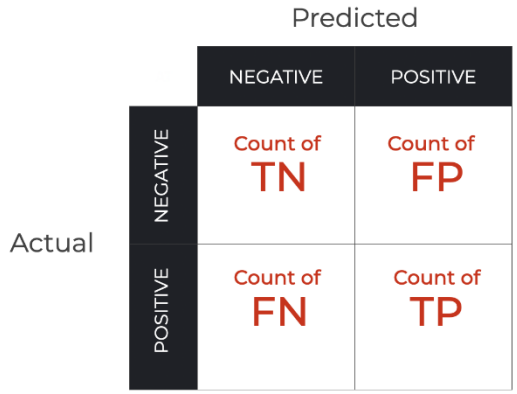
۳-۱ تحلیل نتایج

در ادامه پارامترهای مختلف شبکه را که پس از آموزش آن بدست آمده تحلیل می‌کنیم.

به طور کلی داده‌ها را در قسمت قبلی به سه قسمت آموزش، ارزیابی و آزمون تقسیم‌بندی کردیم. برای نمودارهای Loss و Accuracy این موارد بر حسب داده‌های آموزش و ارزیابی بیان شده و پارامترهای بعدی که شامل Accuracy، Precision، Recall، DCS و AUC هستند بر اساس داده‌های آزمون ارزیابی شده‌اند.

در ابتدای هر نتیجه، نمودارهای Loss و Accuracy بر اساس Epoch رسم شده‌اند که دید خوبی به میزان همگرایی شبکه می‌دهند. نمودار های accuracy و loss به ترتیب نشان دهنده‌ی روند تغییر این دو معیار در جریان train مدل هستند. خطوط آبی نشان دهنده‌ی تغییرات این معیار ها برای داده های train و خطوط نارنجی برای validation هستند.

در قسمت بعدی هر تحلیل، جدول آشفتگی یا Confusion Matrix آورده می‌شود این جدول به فرم زیر است:



جدول ۱ **Confusion Matrix** نمونه

	0	1
0	TRUE NEGATIVE	FALSE POSITIVE
1	FALSE NEGATIVE	TRUE POSITIVE

ابتدا به Accuracy پرداخته می‌شود. این معیار نشان دهنده‌ی نسبت تعداد داده های درست تخمین زده شده به تعداد کل داده ها است.

Confusion Matrix تعداد تمام حالاتی که در پیش بینی یک داده ممکن است پیش آید را، بر حسب هر حالت، نشان می‌دهد. منظور از جمله‌ی قبل آن است که اگر دو دسته‌ی مثبت و منفی داشته باشیم، تمامی ۴ حالت ممکن یعنی تعداد داده هایی که به درستی مثبت تشخیص داده شده‌اند (TP = True positive)، تعداد داده هایی که به درستی منفی تشخیص داده شده‌اند (TN = True negative)، تعدادتمامی داده هایی که به اشتباه مثبت تشخیص داده شده‌اند (FP = False positive) و تعداد تمامی داده هایی که به اشتباه منفی تشخیص داده شده‌اند (FN = False negative) در Confusion matrix قابل مشاهده هستند.

معیار Precision نشان دهنده‌ی نسبت تعداد داده هایی است که به درستی سالم تشخیص داده شده‌اند به مجموع تعداد داده هایی که به درستی یا اشتباه سالم تشخیص داده شده‌اند، است. هر چه این نسبت به یک نزدیک‌تر باشد یعنی داده های کم‌تری به اشتباه سالم تشخیص داده شده‌اند. این معیار آن جا کاربرد دارد که تشخیص تمامی افراد بیمار مهم باشد حتی اگر به اشتباه افراد سالم بیمار تشخیص داده شوند.

حال به معیار F1 Score یا DSC پرداخته می‌شود. یکی از راه های ارزیابی درست داده های نامتوازن (imbalance)، مثل دیتاست این سوال، استفاده از F1 score است. در این معیار، همانگونه که خواهیم دید، علاوه بر خطای مطلق (مثل Accuracy)، نوع خطا را نیز تاثیر می‌دهد. البته راه های دیگری نیز مانند متوازن کردن دیتاست.

محاسبه‌ی Recall مانند Precision است با این تفاوت که در مخرج نسبت جای تعداد افرادی که به اشتباه سالم تشخیص داده شده‌اند را با تعداد افرادی که با اشتباه بیمار تشخیص داده شده‌اند عوض کنیم. با در نظر گرفتن توضیحات فوق به بیان فرمول F1 score پرداخته می‌شود:

$$F1\ score = 2 * \frac{Precision\ * Recall}{Precision + Recall}$$

فرمول فوق بیان می‌کند زمانی F1 score بیشتر است که داده های هر دو دسته به درستی تشخیص داده شده باشند. این فرمول در مقاله به صورت زیر آورده شده است:

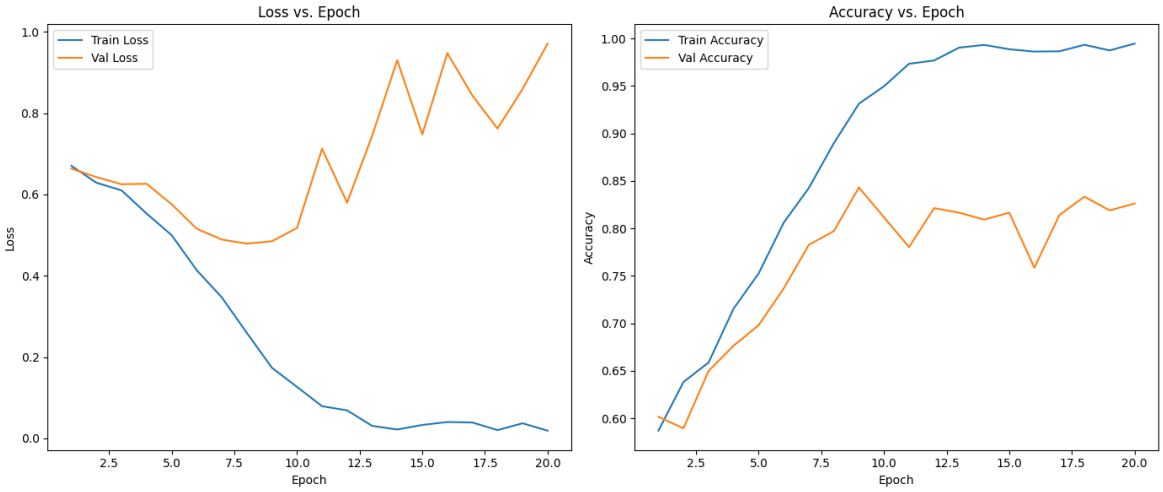
$$DSC = \frac{2 \times Precision \times Recall}{Precision + Recall} = \frac{2TP}{2TP + FP + FN}$$

نمودار یا منحنی ROC توسط ترسیم نسبت Recall برحسب Precision، ایجاد می‌شود. در این نمودار محور عمودی بیانگر تغییرات Recall برحسب داده ها و نمودار افقی بیانگر تغییرات Precision است. شکل

منحنی در این سوال نشانگر عملگر مناسب مدل است و به منحنی ایده آل نزدیک است. به مساحت زیر این نمودار AUC می‌گویند که هرچه بیشتر باشد مدل عملکرد بهتری دارد.

نتایج پیاده‌سازی با نسبت ۰.۱

به طور کلی فرایند آموزش با پارامترهای مختلفی از این مقاله انجام شد. نتیجه زیر یکی از مواردی است که به نسبت دقت بهتری دریافت شد. هرچند که در برخی اجراها دقت تا حدود ۹۰ درصد هم بعضا دیده‌شد. برای این پیاده‌سازی از BatchSize برابر ۶۴ استفاده شد. همچنین ابعاد تصاویر ورودی ۶۴ در ۶۴ هستند. دلیل اینکه دقت از مقاله کمتر است می‌توان به این دلیل باشد که تعداد داده‌ها در این تمرین از مقاله کمتر هستند و ممکن است در پیاده‌سازی‌های Augmentation تفاوت‌های جزئی وجود داشته باشد. هرچند که پارامترها مشابه مقاله بود. دقت در داده‌های آموزش حدود ۹۹ درصد و ارزیابی حدود ۸۵ درصد رسید.

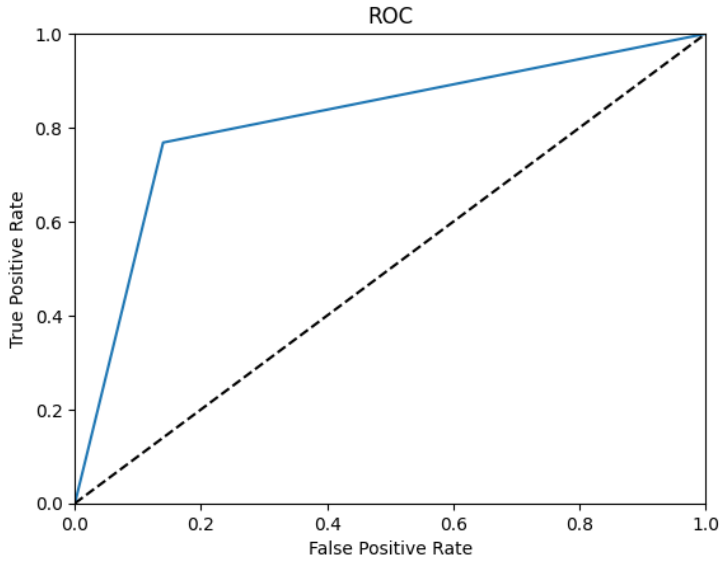


شکل ۵ نمودارهای **Accuracy** و **Loss** نسبت به **Epoch** برای داده‌های آموزشی و ارزیابی

همانطور که در شکل بالا مشاهده می‌شود، مدل در ایپاک ۹ به دقت حدود ۸۵ درصد می‌رسد که دقت به نسبت قابل قبولی است ولی پس از آن هم در LOSS در داده‌های ارزیابی حالت نوسانی داشت. در برخی ایپاک‌ها دقت در آموزش بالا می‌رفت ولی در ارزیابی کاهش داشتیم که به دلیل بیش‌برازش‌شدن مدل بود.

جدول ۲ ماتریس آشفتگی برای نسبت ۰.۱

	0	1
0	203	33
1	41	136



شکل ۶ نمودار **ROC** برای طبقه‌بندی تصاویر

Classification Report:	precision	recall	f1-score	support
0	0.83	0.86	0.85	236
1	0.80	0.77	0.79	177
accuracy			0.82	413
macro avg	0.82	0.81	0.82	413
weighted avg	0.82	0.82	0.82	413

شکل ۷ گزارش طبقه‌بندی

یکی از مشکلاتی که وجود دارد نامتوازن بودن توزیع داده‌های کلاس‌هاست که در قسمت قبل دلیل آن را بررسی کردیم و متوجه شدیم که معیارهایی اضافه برای سنجش بهتر مدل نیاز است.

برای این مدل مقادیر زیر که توسط سوال مطرح شده بود خروجی گرفته شد:

Precision: 0.8184,

Recall: 0.8143,

F1 or DSC Score: 0.8160,

AUC: 0.8143

مدل ارائه شده در مقاله دقت Precision و Recall بالاتری دارد.DSC که نوعی میانگین وزن‌دار است نیز طبعا در نتایج مقاله بهتر هستند. AUC نیز به به مساحت زیر نمودار AUC می‌گویند که هرچه بیشتر باشد مدل عملکرد بهتری دارد و این مورد نیز در مقاله بیشتر است. . دلیل اینکه دقت از مقاله کمتر است

می‌توان به این دلیل باشد که تعداد داده‌ها در این تمرین از مقاله کمتر هستند و ممکن است در پیاده‌سازی‌های Augmentation تفاوت‌های جزئی وجود داشته باشد. هرچند که پارامترها مشابه مقاله بود.

بجز این مورد پیاده‌سازی‌هایی با پارامتر شخصی و به دور مقاله هم انجام شد. که به دقت بالای ۹۰ درصدی رسیدیم ولی از آنجا که در این تمرین خواسته شده موارد طبق مقاله باشند آن مورد نهایی نشد. یکی از تغییرات که انجام شد تغییر شیوه داده‌افزایی بود به نحوی که نوع پیاده‌سازی آن بر روی داده‌ها و روش انجام شده تاثیر منفی داشت.

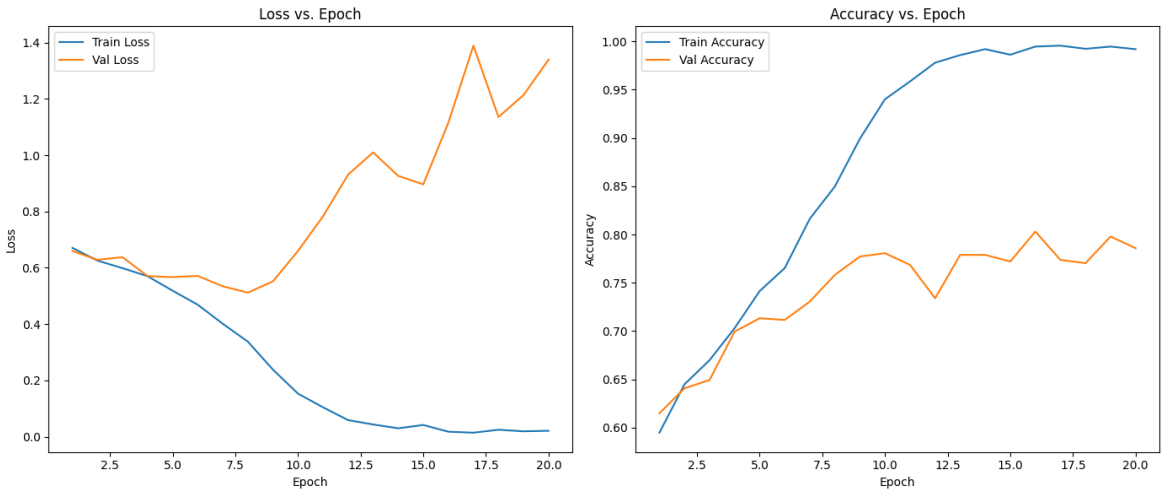
۳-۱ مقایسه نتایج

اثر تقسیم‌بندی داده‌ها به نسبت ۰.۳

به این منظور ۳۰ درصد داده‌ها برای آزمون و ۷۰ درصد آن برای آموزش و ارزیابی کنار گذاشته شد.

```
test_size = int(0.3 * total_size)  # 30% for testing
train_size = total_size - test_size  # 70% for training

train_dataset, test_dataset = random_split(dataset, [train_size, test_size])

train_train_size = int(0.9 * train_size)  # 90% of training set
val_size = train_size - train_train_size  # Remaining for validation
```

نتایج به صورت زیر است:



شکل ۸ نمودارهای **Accuracy** و **Loss** نسبت به **Epoch** برای نسبت ۰.۳

```
Confusion Matrix:
[[1204  202]
 [ 338  737]]
Classification Report:
              precision    recall  f1-score   support

           0       0.78      0.86      0.82      1406
           1       0.78      0.69      0.73      1075

    accuracy                           0.78      2481
   macro avg       0.78      0.77      0.77      2481
weighted avg       0.78      0.78      0.78      2481

Precision: 0.7828, Recall: 0.7710, F1 or DSC Score: 0.7744, AUC: 0.7710
```

شکل ۹ گزارش طبقه‌بندی برای نسبت ۰.۳

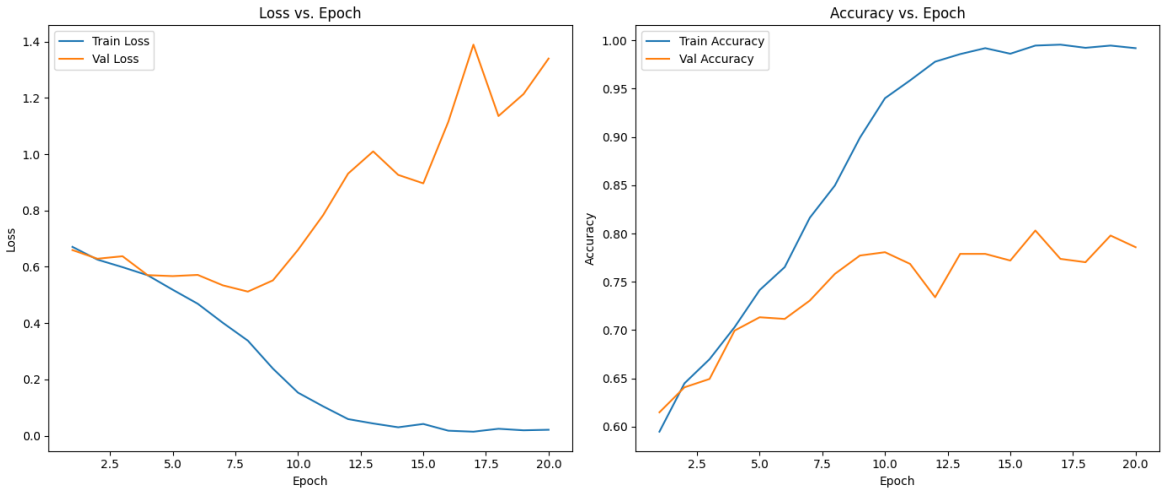
همانطور که مشاهده می‌شود باتوجه به کاهش داده‌های آموزشی دقت مدل کاهش یافت. در اینجا ما داده‌های آموزشی را کم کردیم و از طرفی داده‌های آزمون که بر اساس آن‌ها سنجش بالا انجام می‌شود را افزایش دادیم. این مورد را باید در نظر گرفت که داده‌های آموزشی و آزمون اشتراکی ندارند. تنها نکته مثبت این مورد افزایش سرعت آموزش بود.

اثر تقسیم‌بندی داده‌ها به نسبت ۰.۵

به این منظور ۵۰ درصد داده‌ها برای آزمون و ۵۰ درصد آن برای آموزش و ارزیابی کنار گذاشته شد.

```python
test_size = int(0.3 * total_size)  # 30% for testing
train_size = total_size - test_size  # 70% for training

train_dataset, test_dataset = random_split(dataset, [train_size, test_size])

train_train_size = int(0.9 * train_size)  # 90% of training set
val_size = train_size - train_train_size  # Remaining for validation
```

نتایج به صورت زیر است:



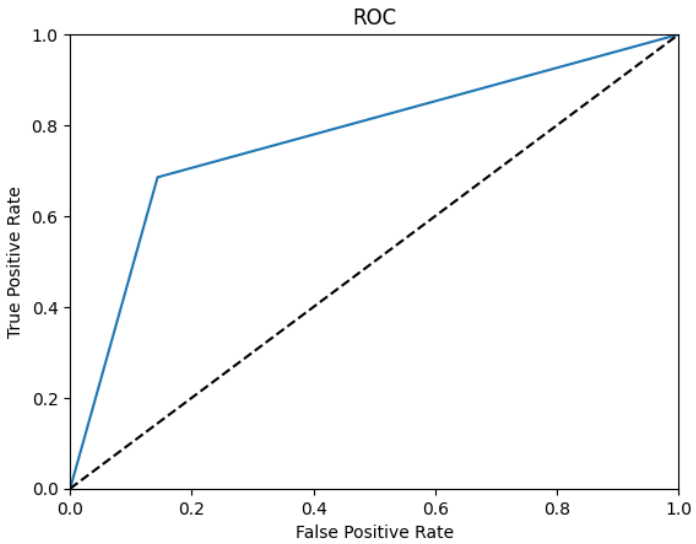
شکل ۱۰ نمودارهای **Accuracy** و **Loss** نسبت به **Epoch** برای نسبت ۰.۵

```
Confusion Matrix:
[[1613  320]
 [ 417  958]]
Classification Report:
              precision    recall  f1-score   support

           0       0.79      0.83      0.81      1933
           1       0.75      0.70      0.72      1375

    accuracy                           0.78      3308
   macro avg       0.77      0.77      0.77      3308
weighted avg       0.78      0.78      0.78      3308

Precision: 0.7721, Recall: 0.7656, F1 or DSC Score: 0.7681, AUC: 0.7656
```

شکل ۱۱ گزارش طبقه‌بندی برای نسبت ۰.۵

همانطور که مشاهده می‌شود باتوجه به کاهش داده‌های آموزشی دقت مدل کاهش یافت. در اینجا ما داده‌های آموزشی را کم کردیم و از طرفی داده‌های آزمون که بر اساس آن‌ها سنجش بالا انجام می‌شود را افزایش دادیم. این مورد را باید در نظر گرفت که داده‌های آموزشی و آزمون اشتراکی ندارند. تنها نکته مثبت این مورد افزایش سرعت آموزش بود.

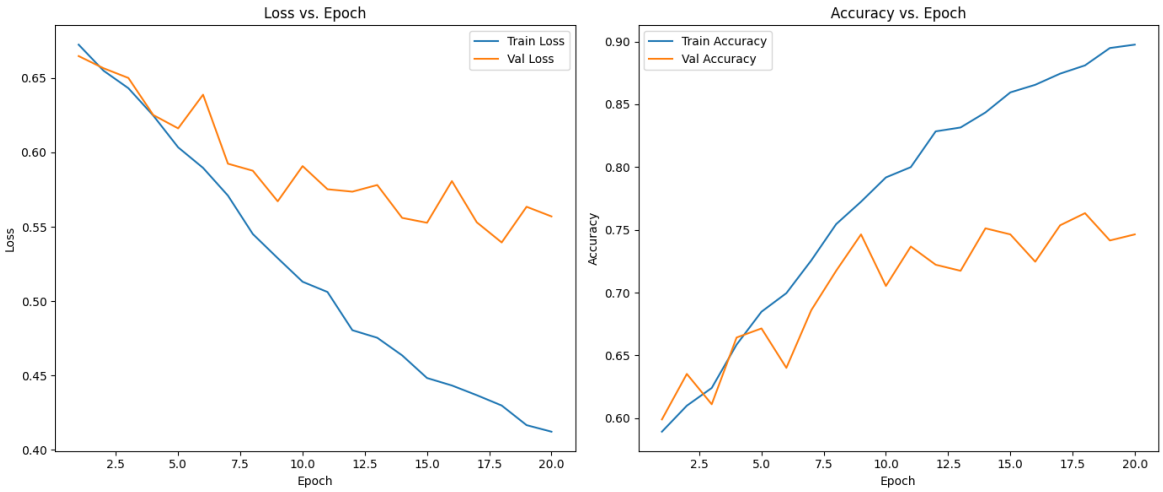
نمودار ROC برای آزمایش بالا به صورت زیر است. واضح است مقادیر از مقدار ایده‌آل و منحنی ایده‌آل فاصله گرفته‌اند:



شکل ۱۲ نمودار **ROC** پس از تغییر نسبت تقسیم داده‌ها

اثر Dropout

مقاله از این تکنیک نیز برای سرعت بخشیدن به آموزش مدل و جلوگیری از over fitting با افزایش قدرت generalization آن استفاده میکند. در واقع در مدل‌های دیپ لرنینگ اگر تعداد پارامترهای training زیاد باشد و داده ورودی کم باشد، ممکن است مدل به دقت بالا در داده‌های آموزشی و دقت پایین در داده‌های تست دست پیدا کند. به این منظور در مقاله از Dropout استفاده میشود و این لایه به صورت رندوم و با احتمال مشخصی که در مقاله در نظر گرفته شده است، از برخی connected چشم پوشی می‌کند تا از over fitting جلوگیری شود سرعت افزایش پیدا کند.



شکل ۱۳ نمودارهای **Accuracy** و **Loss** نسبت به **Epoch** پس از **Dropout**

اعمال این تغییر در شبکه به افزایش همگرایی داده‌های آموزشی کمک کرد ولی به صورت کلی دقت مدل افزایش نیافت.اگرچه این اتفاق در داده‌های آموزشی افتاد ولی در داده‌های تست به نسبت تغییرات قبلی تاثیر آن در افزایش دقت مدل در داده‌های مشاهده‌نشده به نسبت بهتر شد. این مورد مطابق با مقاله بود و در داده‌های آموزشی کاهش دقت و در داده‌های تست افزایش دقت به نسبت مشاهده می‌شود.

```
Confusion Matrix:
[[209  33]
 [ 50 121]]
Classification Report:
              precision    recall  f1-score   support

           0       0.81      0.86      0.83       242
           1       0.79      0.71      0.74       171

    accuracy                           0.80       413
   macro avg       0.80      0.79      0.79       413
weighted avg       0.80      0.80      0.80       413

Precision: 0.7963, Recall: 0.7856, F1 or DSC Score: 0.7895, AUC: 0.7856
```

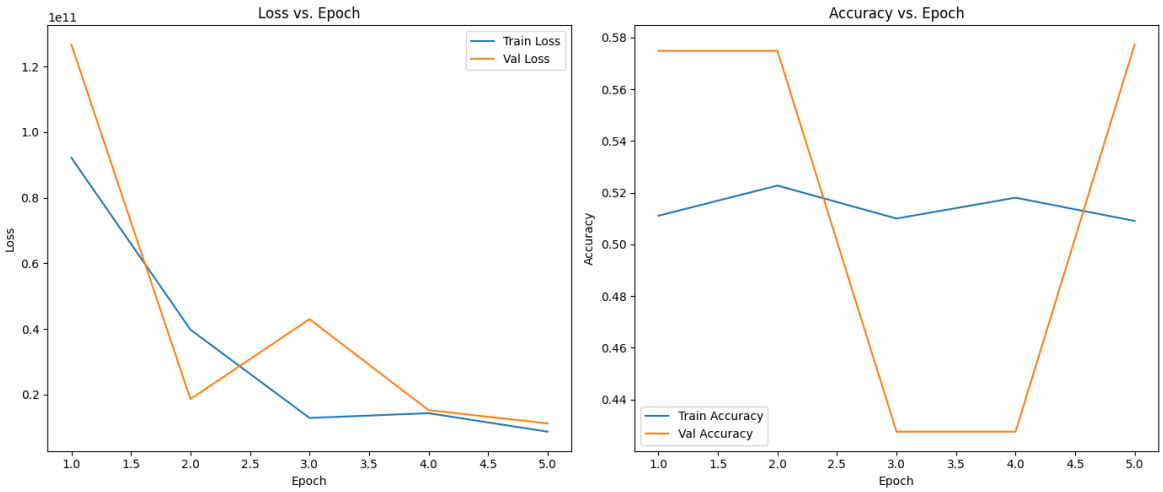
شکل ۱۴گزارش طبقه‌بندی پس از **Dropout**

این مورد به Generalization مدل و جلوگیری از بیش‌برازش کمک می‌کند.

اثر Glorot Initialization

این رویکرد یکی از روش‌های وزن‌دهی اولیه مدل است و با استفاده از این روش، مقدارهای اولیه وزن‌ها بر اساس میانگین صفر و یک واریانس خاص انجام می‌شود. به طور کلی این روش از ۲ توزیع نرمال و یکنواخت برای مقداردهی استفاده می‌شود که نوع یکنواخت آن در این مقاله اشاره شده است. برای پیاده‌سازی آن از متد xavier_uniform_ استفاده شد. برای مقایسه تاثیر آن دو روش دیگر که در مقاله اشاره شده‌اند برای مقداردهی اولیه وزن‌ها استفاده شد. یکی از آن‌ها وزن‌دهی صفر و دیگری وزن‌دهی ۱ بود.

پس از اعمال این تغییرات، شبکه نتوانست به نقطه بهتری همگرا شود و دقت مدل بسیار پایین شد. مقدار ۵۰ درصد برای یک طبقه‌بند دودویی کمترین حالتی است که با پرتاب سکه قابل دریافت است. این مورد نشان می‌دهد که وزن‌دهی اولیه برای همگرایی شبکه بسیار مهم است. نتایج دقت این آزمایش به صورت زیر است:



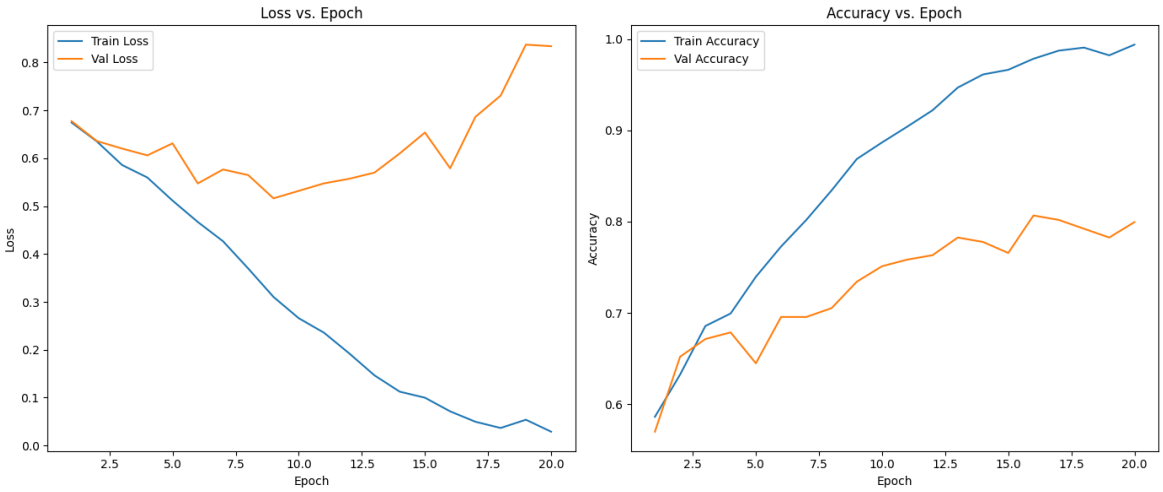
شکل ۱۵ نمودارهای **Accuracy** و **Loss** نسبت به **Epoch** پس از وزن‌دهی اولیه یک

تمام موارد بالا برای وزن‌دهی صفر نیز انجام شد و مجددا نتیجه مشابهی دریافت شد.

اثر تغییر معماری

برای مشاهده اثر مدل پیشنهادشده در این مقاله دو مدل دیگر که برای آزمون ارائه شده‌اند پیاده‌سازی شد و نتجه آموزش شبکه بر اساس این معماری‌ها در ادامه آورده شده‌است. به طور کلی مدل پیشنهادی مقاله بالاترین دقت را داشت. پس از آن مدل آزمون ۱ دقتی با حدود ۵ درصد کاهش ارائه می‌دهد. در آخر نیز مدل آزمون ۲ عملکرد ضعیف تری نسبت به هر دوی آن‌ها در بررسی‌های انجام‌شده با کمک داده‌های آزمون داشت.

نتایج استفاده از معماری مدل Testing Model 1

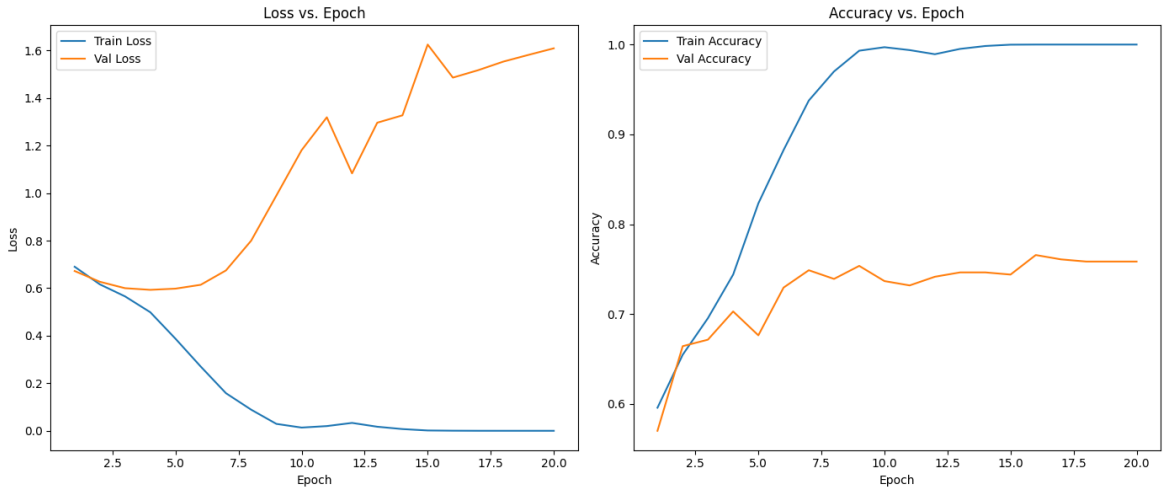


شکل ۱۶نمودارهای **Accuracy** و **Loss** نسبت به **Epoch** مدل **Testing Model 1**

```
Confusion Matrix:
[[179  63]
 [ 29 142]]
Classification Report:
              precision    recall  f1-score   support

           0       0.86      0.74      0.80       242
           1       0.69      0.83      0.76       171

    accuracy                           0.78       413
   macro avg       0.78      0.79      0.78       413
weighted avg       0.79      0.78      0.78       413

Precision: 0.7766, Recall: 0.7850, F1 or DSC Score: 0.7754, AUC: 0.7850
```

شکل ۱۷ گزارش طبقه‌بندی مدل **Testing Model 1**

نتایج استفاده از معماری مدل Testing Model 2



شکل ۱۸نمودارهای **Accuracy** و **Loss** نسبت به **Epoch** مدل **Testing Model 2**

```
Confusion Matrix:
[[201  41]
 [ 50 121]]
Classification Report:
              precision    recall  f1-score   support

           0       0.80      0.83      0.82       242
           1       0.75      0.71      0.73       171

    accuracy                           0.78       413
   macro avg       0.77      0.77      0.77       413
weighted avg       0.78      0.78      0.78       413

Precision: 0.7739, Recall: 0.7691, F1 or DSC Score: 0.7711, AUC: 0.7691
```

شکل ۱۹ گزارش طبقه‌بندی مدل **Testing Model 2**

مقایسه مدل‌های Test با مدل پیشنهادی

به طور کلی نتایج مشاهده شده پس از انجام آزمایش مدل‌های مختلف مطابق مقاله بود. در مدل پیشنهادی توسط مقاله، از ۴ لایه Convolutional استفاده شده است. پس از هر لایه Convolutional نیز یک لایه MaxPooling داریم. در آخر نیز ۲ لایه ۱۲۸ و ۶۴ نورونی از نوع Fully Connected قرار داده شده است.

در مدل تست ۱ از ۲ لایه Convolutional استفاده شده که ۲ لایه از مدل پیشنهادی کمتر است که هر کدام ۳۲ فیلتر داشته و از Kernel ۳ در ۳ استفاده می‌کنند که مشابه مدل پیشنهادشده است. یک تفاوت دیگر آن با مدل پیشنهادی این است که اینبار پس از هر لایه Convolutional یک لایه

MaxPooling داریم. در آخر نیز از یک لایه Fully Connected ۱۲۸ نورونی و یک خروجی استفاده شده است.

در مدل تست ۲ از ۲ لایه Convolutional استفاده شده که ۲ لایه از مدل پیشنهادی کمتر است که هر کدام ۳۲ فیلتر داشته و از Kernel ۳ در ۳ استفاده می‌کنند که مشابه مدل پیشنهادشده است. یک تفاوت دیگر آن با مدل پیشنهادی این است که اینبار فقط پس از لایه دوم Convolutional یک لایه MaxPooling داریم. در آخر نیز ۲ لایه ۱۲۸ و ۶۴ نورونی از نوع Fully Connected قرار داده شده است که مشابه مدل پیشنهادی است.

به طور کلی مدل پیشنهادی این مقاله از لایه‌های Convolutional بیشتری استفاده کرده که کمک می‌کند الگوهای بیشتری از تصاویر شناخته شوند. لایه‌های MaxPooling هم به کاهش محاسبات و کاهش بیش‌برازش مدل کمک می‌کنند. به طور کلی معماری پیشنهادی عمق بیشتری داشته و عملکرد بهتری در این مسئله طبقه‌بندی دارد.

پرسش ۲. بررسی تاثیر افزایش داده بر عملکرد شبکه‌های کانولوشنی Fine-Tune شده

۲-۱- آماده‌سازی اولیه

در ابتدا کتابخانه‌های مورد نیاز در پروژه را افزوده و چون برروی گوگل کولب فعالیت می کردم، محیط colab را با google drive که داده های در آن قرار دارند mount کرده ام:

```
Import libraries

[1] from keras.preprocessing.image import ImageDataGenerator, array_to_img, img_to_array, save_img
    import random
    import numpy as np
    import matplotlib.pyplot as plt
    import os
    from keras.applications import VGG16
    from keras.applications import ResNet50
    from keras import models
    from keras import layers
    from keras import optimizers
    from keras.optimizers.schedules import ExponentialDecay

connect to google drive

[2] from google.colab import drive
    drive.mount('/content/drive')

    Mounted at /content/drive
```

۲-۲- پیش پردازش تصاویر

ابتدا می بایست دیتاستی که در اختیار قرار گرفته است را لود کنیم:

```
test_dir = '/content/drive/MyDrive/Deep_learning/HW2/Q2 Dataset/HW2_Dataset/Test'
train_dir = '/content/drive/MyDrive/Deep_learning/HW2/Q2 Dataset/HW2_Dataset/Train'
augmented_dir = '/content/drive/MyDrive/Deep_learning/HW2/Q2 Dataset/final_aug'
augmented_dir_cats = '/content/drive/MyDrive/Deep_learning/HW2/Q2 Dataset/final_aug/Cats'
augmented_dir_dogs = '/content/drive/MyDrive/Deep_learning/HW2/Q2 Dataset/final_aug/Dogs'
augmented_train_dir = '/content/drive/MyDrive/Deep_learning/HW2/Q2 Dataset/final_aug'
```

ابتدا مسیر پوشه داده های تست و آموزش به همراه مسیری که داده های augmented را درون آن می‌خواهیم بنویسیم را مشخص کرده ایم.

سپس داده ها را لود کرده و در سه دسته تقسیم کردم:

```
datagen = ImageDataGenerator(rescale=1./255, validation_split=0.3)
train_generator = datagen.flow_from_directory(train_dir, target_size=(224, 224), class_mode='categorical', batch_size=128, subset='training')
validation_generator = datagen.flow_from_directory(train_dir, target_size=(224, 224), class_mode='categorical', batch_size=128, subset='validation')
test_generator = datagen.flow_from_directory(test_dir, target_size=(224, 224), class_mode='categorical', batch_size=128)
train_images, train_labels = next(train_generator)
test_images, test_labels = next(test_generator)

Found 492 images belonging to 2 classes.
Found 210 images belonging to 2 classes.
Found 100 images belonging to 2 classes.
```

ابتدا یک object از ImageDataGenerator ایجاد کردم که در آن میزان تقسیم داده های آموزشی به بخش آموزشی و ولیدیشن را مشخص کرده و همچنین مقداری که برای Scale کردن پیکسل‌های تصویر نیاز هست را مشخص کرده ام تا نرمالیزیشن نیز صورت بگیرد و پیکسل ها در بازه ۰ و ۱ باشند.

سپس از طریق تابع flow_from_directory داده ها را خوانده و ضمن آن با توجه به نام دایرکتوری که در آن قرار داشتند، برچسب‌گذاری نیز کردم. به همین ترتیب داده های validation و test را نیز از محل مورد

نظر خواندم و به سایز ۲۲۴ پیکسل نیز تبدیل کردم تمام تصاویر را (در مقاله این سایز ذکر شده بود و به نظر ورودی ResNet50 یا VGG16 با همین ابعاد است).

و یک batch از هریک از داده های آموزشی و تست را نیز خواندم تا اطمینان حاصل کنم درست داده ها را وارد کرده ام.

مشاهده می‌شود که ۴۹۲ داده آموزشی و ۲۱۰ داده validation (مجموعا ۷۰۲ داده برای آموزش مدل) و ۱۰۰ داده برای تست داریم.

برای بررسی داده داریم:

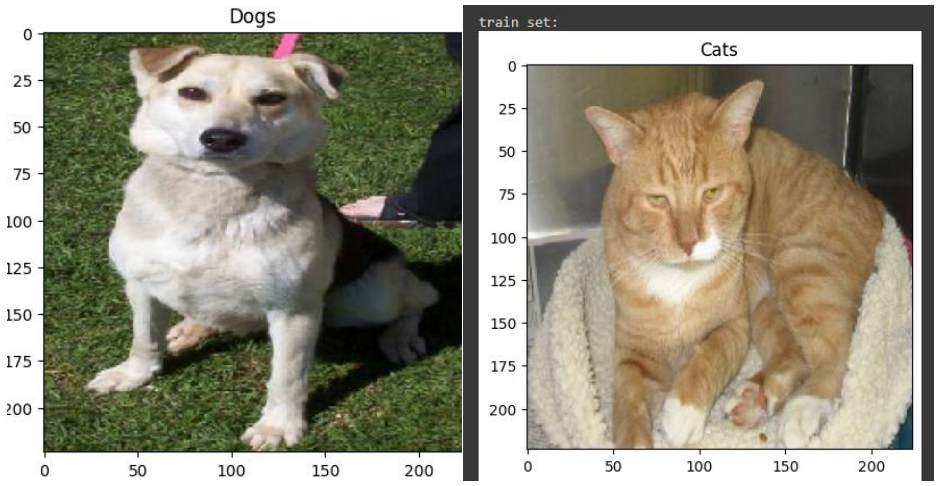
```
train_samples = list(train_generator.class_indices.keys())
print(train_samples)

test_samples = list(test_generator.class_indices.keys())
print(test_samples)
```

```
['Cats', 'Dogs']
['Cats', 'Dogs']
```

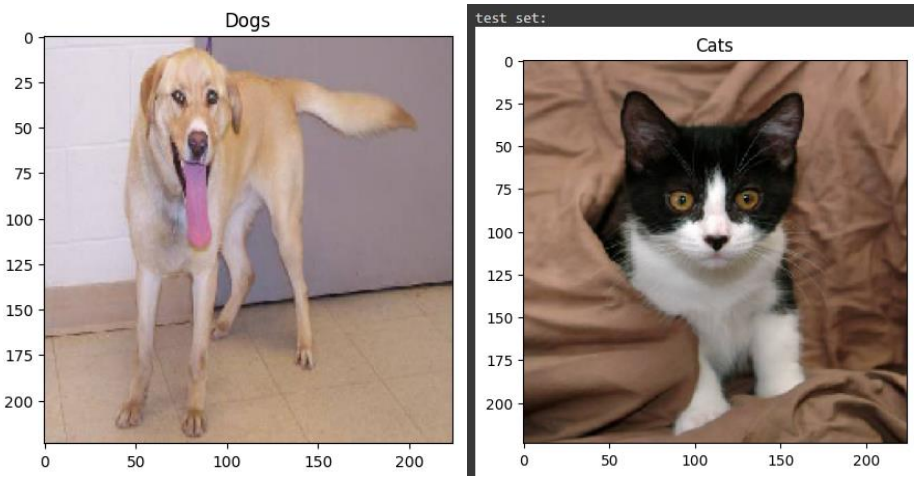
مشاهده می شود در هر دو تست، هر دو لیبل Cats و Dogs را داریم.

و برای نمایش یک نمونه از هر یک داریم:

در داده های آموزشی:



در داده های تست:



(مشخصا این تصاویر با هر بار اجرا به دلیل تغییراتی که در نحوه چیدمان رندوم ایپاک ها داریم تغییر می‌کنند.)

قبل از augmentation تعداد داده ها و نمونه در هر کلاس بدین صورت است:

```
def count_samples(generator, class_labels):
    counts = {label: 0 for label in class_labels}

    for label in generator.labels:
        class_label = class_labels[label]
        counts[class_label] += 1

    return counts
```

```
train_counts = count_samples(train_generator, train_samples)
print(f'Train counts: {train_counts}')

test_counts = count_samples(test_generator, test_samples)
print(f'Test counts: {test_counts}')
```

```
Train counts: {'Cats': 350, 'Dogs': 352}
Test counts: {'Cats': 50, 'Dogs': 50}
```

در ادامه به فرایند افزون سازی داده ها با توجه به روش ها درون مقاله می پردازیم:

Augmentation داده‌ها

برای گسترش داده‌ها در مقاله سه راه حل ارائه شده که تغییراتی هستند که برروی داده می‌توان ایجاد کرد و داده جدید به وجود آورد:

- Horizontal flipping: This is more common than vertical flipping especially for datasets such as CATs vs DOGs. This augmentation is one of the easiest to implement and has proven useful on datasets such as CIFAR-10 and ImageNet. [1]. Vertical flip has no meaning when the dataset has images of CATs and DOGs.
- Rotation: This augmentation is done by rotating images in the clockwise or counterclockwise direction. In this study, random rotation up to 30º is made use of.

- Scaling or zooming: This augmentation zooms the image to make the image look smaller or bigger depends on the zoom value. Zoom value of less than 1.0 will reduce the size of the image and a value greater than 1.0 will increase the size. In this study, a zoom range between 0.75 and 1.25 is used.

۱. معکوس کردن به صورت افقی (لازم به ذکر است که معکوس کردن به صورت عمودی نیز ممکن است اما با توجه به تسکی که داریم و توضیحات مقاله نیاز نیست و در اینجا استفاده نمی کنیم)
۲. چرخش: به صورت کلی با چرخاندن هر تصویر به صورت رندوم در بازه بین ۰ تا ۳۰ درجه به صورت ساعتگرد یا پاد ساعتگرد.
۳. Zoom out/In: بدین صورت که تصویر را به صورت رندوم بین ۱.۲۵ تا ۰.۷۵ بزرگتر یا کوچک‌تر کرده و سایز تصویر تغییر کرده را مجدد به ۲۲۴*۲۲۴ اسکیل می کنیم.

حال در کد بدین صورت داریم:

```
andis = 0
for batch_index in range(len(train_generator)):
    images, labels = train_generator[batch_index]
    andis += 1
    for i in range(len(images)):
        print(f"image number: {i+andis}")
        image = images[i]
        label = labels[i]

        image = np.expand_dims(image, axis=0)

        for epoch in range(50):
            augmentation = random.choice(['flip', 'rotate', 'zoom'])

            if augmentation == 'flip':
                datagen = ImageDataGenerator(horizontal_flip=True)
            elif augmentation == 'rotate':
                datagen = ImageDataGenerator(rotation_range=30)
            else:
                datagen = ImageDataGenerator(zoom_range=[0.75, 1.25])

            image_augmented = datagen.random_transform(img_to_array(image[0]))

            img = array_to_img(image_augmented)

            img.save(os.path.join(augmented_dir_cats if label == 0 else augmented_dir_dogs, f'aug_{batch_index}_{i}_epoch_{epoch}.jpeg'))
```

```
image number: 1
image number: 2
image number: 3
image number: 4
image number: 5
image number: 6
image number: 7
```

من اینطور عمل کرده ام، بدین صورت ابتدا برروی batch های مختلف train_generator که کلیه ۷۰۲ داده آموزشی در آن قرار دارد یک حلقه زده ام تا به تمامی تصویر آموزشی ام دسترسی داشته باشم، به همین دلیل در ادامه در هر batch برروی تصاویر داخل آن batch نیز یک حلقه زده ام حالا در حلقه دوم به تمام تصاویر آموزشی ام دسترسی دارم. حالا روی هر تصویر طبق متن مقاله در طی ۵۰ epoch و ۵۰ مرتبه یکی از سه تغییر مختلف که خود شامل انواع تغییرات هستند (بدلیل وجود شاخص رندوم بدون مثل درجه چرخش یا میزان Zoom OUT/IN) را اعمال کردم.

در نهایت هم فقط کافی است تا تصویر تغییر کرده را با توجه به کلاس آن در پوشه مخصوص خود و جدیدش ذخیره کنم

به این ترتیب اگر همه چیز درست باشد می بایست هر عکس به ۵۰ عکس با تغییرات ذکر شده گسترش یافته باشد. و در نهایت ۵۰*۷۲۰ = ۳۵۱۰۰ تصویر برای داده آموزشی داشته باشیم. (من داده هایی که برای validation نیاز دارم را نیز گسترش داده ام.)

که مشاهده می شود همین طور هم هست:

```
[20] augmented_datagen = ImageDataGenerator(rescale=1./255, validation_split=0.3)
     augmented_train_generator = augmented_datagen.flow_from_directory(augmented_dir, target_size=(224, 224), class_mode='binary', batch_size=128, subset='training')
     augmented_validation_generator = augmented_datagen.flow_from_directory(augmented_dir, target_size=(224, 224), class_mode='binary', batch_size=128, subset='validation')

     Found 24570 images belonging to 2 classes.
     Found 10530 images belonging to 2 classes.
```

در اینجا من مجدد داده‌های augment را بارگذاری کرده تحت عنوان augmented_train_generator برای داده های آموزشی و augmented_validation_generator داده های validation مشخص کرده‌ام. که ۲۴۵۷۰ داده آموزشی و ۱۰۵۳۰ داده validation داریم.

پیش پردازش‌های مورد نیاز انجام شده نیز در همانطور که در کد مشخص هستند من تصاویرم را در batch های ۱۲۸ تایی ذخیره کرده و سایزشان را چون همگی باید هم اندازه باشند، طبق حالت مد نظر مقاله همه را ۲۲۴*۲۲۴ در نظر گرفتم.

همچنین لازم به ذکر است که ۳۰ درصد تقسیم train و validation که در مقاله گفته شده بود نیز مجدد در اینجا هم داریم و البته داده های تست همان ۱۰۰ تایی هستند که در ابتدا لود کردیم تا تغییر نکند و نتایج در مدل‌ها برای حالت بدون افزونی داده و با افزونی داده قابل قیاس با یکدیگر باشند.

۲-۳- پیاده‌سازی

در بخش پیاده سازی برای fine tune کردن دو شبکه VGG16 و ResNet50 به صورت جداگانه عمل کرده‌ام و در هر کدام سه مورد بررسی شده است، ابتدا شبکه بدون هیچ گونه آموزش اضافه علاوه بر آنچه وزن‌های از پیش تعیین شده دارند را مورد آزمایش قرار دادم، سپس با نمونه ۷۰۲ تایی آموزشی (۴۹۲ آموزش و ۲۱۰ validation) fine tune کرده و سپس مورد ارزیابی قرار دادم و در نهایت با داده افزوده شده (augmented) یعنی نمونه آموزشی ۳۵۱۰۰ تایی (۲۴۵۷۰ داده آموزشی و ۱۰۵۳۰ داده validation) fine-tune کرده و مورد آزمایش قرار دادم.

که در ادامه به بررسی هر یک می پردازیم:

۱-۲-۳- VGG16:

در ابتدا این مدل را ایجاد کردم:

```
base_model = VGG16(weights='imagenet', include_top=False, input_shape=(224, 224, 3))

model = models.Sequential()
model.add(base_model)
model.add(layers.Flatten())
model.add(layers.Dense(4096, activation='relu'))
model.add(layers.Dense(2, activation='softmax'))
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5
58889256/58889256 [==============================] - 0s 0us/step
```

بدین صورت که یک base_model دارم که در حقیقت مدل VGG16 با پارامترهای آموزش دیده روی داده آموزشی imagenet است و سایز تصاویر هم مطابق آنچه داشته ۲۲۴*۲۲۴ در سه کانال RGB هست.

در ادامه یک آبجکت model تعریف کردم که در ابتدا مدل اولیه base مدل را به آن افزودم و در ادامه لایه flatten افزوده ام تا با توجه به خروجی شبکه های CNN از یک خروجی سه بعدی، یک خروجی یک بعدی داشته باشم.

سپس با توجه به خروجی های مسئله‌ام دو لایه دیگر به شبکه اضافه کردیم، که آنها دقیقا لایه هایی هستند که در ادامه آموزش می بینند.لایه اول یک لایه fully-connected با ۴۰۹۶ نورون است که آموزش می بینند و لایه آخر هم با توجه به اینکه یک خروجی دو حالت داریم دو نورون در نهایت برای لایه خروجی در نظر گرفتیم که وزن آنها نیز در حین آموزش مشخص می‌گردند.

در ادامه تنظیم هایپرپارامترها میرسیم که با توجه به مقاله داریم:

Table 2. Hyperparameters used for training fine-tuned VGG16 and ResNet50 models

Hyperparameters	Initial Learning Rate (LR)	LR Decay rate	Momentum	Min-batch size	No. of Epochs
Value	0.1	0.002	0.9	10	50

برای قسمت تنظیم پارامترها و کامپایل کردن مدل داریم:

```
[22] lr_schedule = ExponentialDecay(initial_learning_rate=0.1, decay_steps=100000, decay_rate=0.002)

     model.compile(loss='binary_crossentropy',
                   optimizer=optimizers.SGD(learning_rate=lr_schedule, momentum=0.9),
                   metrics=['accuracy'])
```

در این کد طبق خواسته مقاله initial_learning_rate را ۰.۱ در نظر گرفته و LR decay rate نیز ۰.۰۰۲ در نظر گرفته شده است، momentum نیز در قسمت کامپایل کردن مدل مقدار دهی شده و ۰.۹ در نظر گرفته شده است. برای تعداد ایپاک نیز در هر آموزش تعداد ایپاک‌های که می بایست آموزش ببیند را در نظر گرفتم. همچنین لازم به ذکر است که با توجه به تابع loss درون مقاله:

4.3.4. Loss function

In deep learning, the objective function is referred to as a loss function and it indicates the error generated during the forward pass. The main purpose of training a model is to minimize the loss function by adjusting the parameters based on the loss function. Cross-entropy and mean square error (MSE) are the commonly used loss functions. Overall loss value for a given batch of training images is referred to as cost function. Equation (4) shows the categorical cross-entropy used as a loss function in this study. In this equation, P_y & P_f refer to ground-truth distribution and score distribution of 'x' respectively [22].

$$L(y, f(x)) = H(P_y, P_f) \triangleq -\sum_{i=1}^{n} P_y(x_i) \log P_f(x_i) \quad (4)$$

تابع loss برای مدل را نیز binary_crossentropy در نظر گرفتم.

حالا بدون هیچ گونه آموزش و تنها با در نظر گرفتن وزن‌های اولیه مدل VGG16 بر اساس imagenet دقت ۵۶٪ را داریم:

```
[23] loss, accuracy = model.evaluate(test_generator)
     print(f'Test accuracy: {accuracy}, Test loss: {loss}')

     1/1 [==============================] - 18s 18s/step - loss: 0.7134 - accuracy: 0.5700
     Test accuracy: 0.5699999928474426, Test loss: 0.7133709192276001
```

حالا مطابق مراحل گفته شده در مقاله برای fine tune کردن مدل پیش می رویم:

- Load the VGG16 or ResNet50 with weights pre-trained on the ImageNet dataset.
- Replace the original fully connected (FC) layers with a new one as per the given image class.
- Freeze all the CONV layers for transferring what has been learned.
- Train the network
- Unfreeze the last CONV block which extracts task-specific features.
- Train the network again to fine-tune parameters of the last CONV layer block.

ابتدا طبق مراحل یک و دو مدل را لود کرده و لایه های مورد نیاز را افزوده و تعییر دادم (توضیح ایجاد model بر اساس base_model) حالا برای گام سوم و گام های بعدی دارم:

در کد و برای داده‌های افزایش نیافته (۷۰۲ تایی):

```
for layer in base_model.layers:
    layer.trainable = False

lr_schedule = ExponentialDecay(initial_learning_rate=0.1, decay_steps=100000, decay_rate=0.002)

model.compile(loss='categorical_crossentropy',
              optimizer=optimizers.SGD(learning_rate=lr_schedule, momentum=0.9),
              metrics=['accuracy'])
```

ابتدا طبق گام سوم تمامی لایه های از پیش آموزش دیده را freeze کردم.

سپس مجدد طبق خواسته مقاله مدل را کامپایل و با داده‌های کوچک آموزش و validation کردم:

```
history = model.fit(
    train_generator,
    steps_per_epoch=len(train_generator),
    epochs=20,
    validation_data=validation_generator,
    validation_steps=len(validation_generator))
```

```
Epoch 1/20
4/4 [==============================] - 56s 12s/step - loss: 6.3665 - accuracy: 0.5102 - val_loss: 1.0102 - val_accuracy: 0.5000
Epoch 2/20
4/4 [==============================] - 5s 1s/step - loss: 0.7825 - accuracy: 0.5285 - val_loss: 0.6720 - val_accuracy: 0.5000
Epoch 3/20
4/4 [==============================] - 5s 1s/step - loss: 0.6152 - accuracy: 0.6951 - val_loss: 0.5740 - val_accuracy: 0.7571
Epoch 4/20
4/4 [==============================] - 5s 1s/step - loss: 0.5555 - accuracy: 0.7175 - val_loss: 0.6383 - val_accuracy: 0.5476
Epoch 5/20
4/4 [==============================] - 5s 1s/step - loss: 0.5518 - accuracy: 0.6829 - val_loss: 1.7342 - val_accuracy: 0.5000
Epoch 6/20
4/4 [==============================] - 5s 1s/step - loss: 1.0773 - accuracy: 0.5813 - val_loss: 0.4594 - val_accuracy: 0.8048
Epoch 7/20
4/4 [==============================] - 5s 1s/step - loss: 0.5219 - accuracy: 0.6951 - val_loss: 0.5497 - val_accuracy: 0.8381
Epoch 8/20
4/4 [==============================] - 5s 1s/step - loss: 0.4839 - accuracy: 0.8130 - val_loss: 0.5561 - val_accuracy: 0.6810
Epoch 9/20
4/4 [==============================] - 7s 2s/step - loss: 0.3934 - accuracy: 0.8150 - val_loss: 0.5113 - val_accuracy: 0.7476
Epoch 10/20
4/4 [==============================] - 5s 1s/step - loss: 0.5517 - accuracy: 0.7154 - val_loss: 0.6236 - val_accuracy: 0.7143
Epoch 11/20
4/4 [==============================] - 5s 1s/step - loss: 0.3651 - accuracy: 0.8171 - val_loss: 0.3741 - val_accuracy: 0.8143
Epoch 12/20
4/4 [==============================] - 6s 2s/step - loss: 0.3663 - accuracy: 0.8293 - val_loss: 0.2710 - val_accuracy: 0.8857
```

این آموزش بدون حضور لایه‌ها از پیش آموزش دیده را ۲۰ ایپاک پیش بردم.

در ادامه لایه هایی که فریز شده بودند را آزاد کرده:

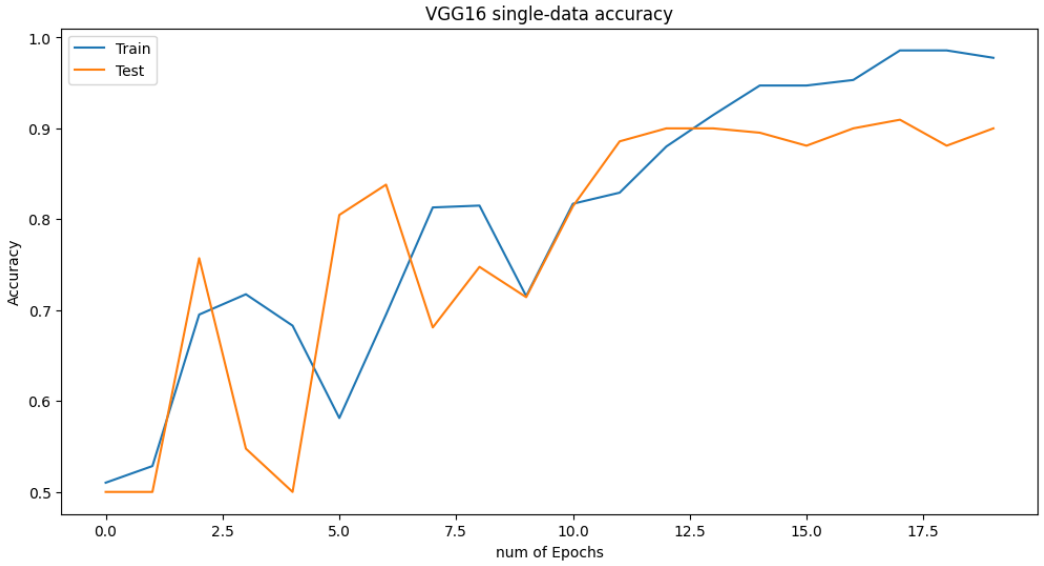
```
for layer in base_model.layers[-4:]:
    layer.trainable = True

model.compile(loss='categorical_crossentropy',
              optimizer=optimizers.SGD(learning_rate=lr_schedule, momentum=0.9),
              metrics=['accuracy'])

history = model.fit(
    train_generator,
    steps_per_epoch=len(train_generator),
    epochs=30,
    validation_data=validation_generator,
    validation_steps=len(validation_generator))
```

و ۳۰ ایپاک نیز برروی با حضور این لایه‌ها آموزش دادم. و در نهایت به این دقت رسیدم:

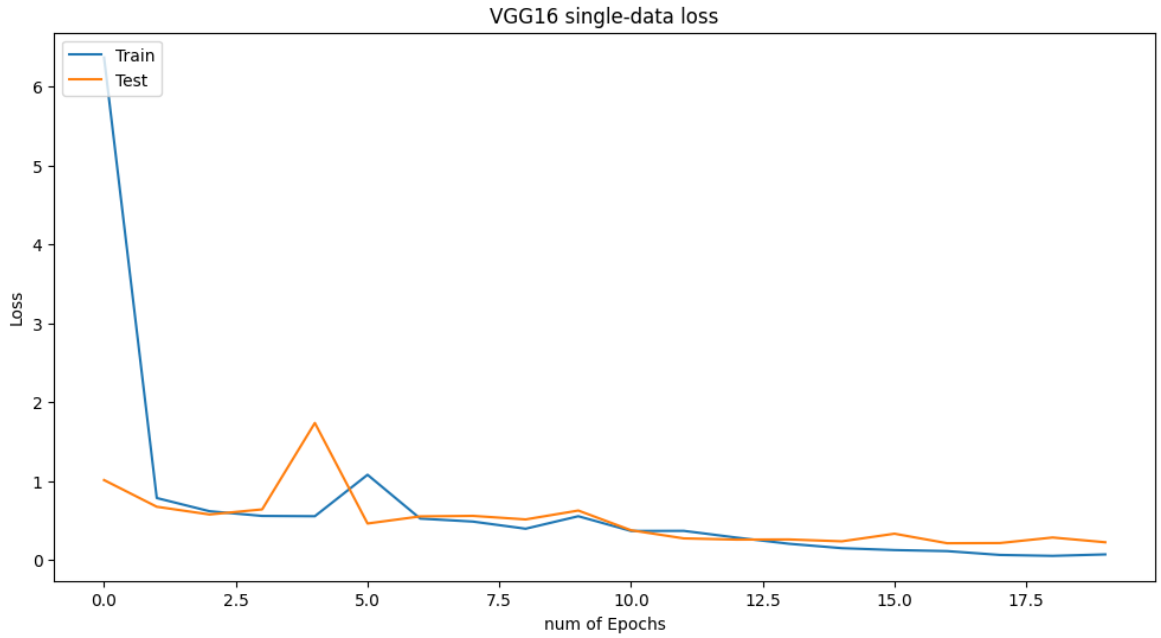
```
loss, accuracy = model.evaluate(test_generator)
print(f'Test accuracy: {accuracy}, Test loss: {loss}')
```

```
1/1 [==============================] - 1s 950ms/step - loss: 0.3700 - accuracy: 0.8600
Test accuracy: 0.8600000143051147, Test loss: 0.3700281083583832
```

و با توجه به ۲۰ ایپاک آموزش اولیه، این نمودار را برای دقت مدل:



و این نمودار را برای میزان Loss داریم:



حالا مدل مان را مجدد ساخته و کاملا مانند گام‌های قبل پیش میرویم با این تفاوت که در گام آخر برای به جای آموزش مدلمان با داده‌های کم با داده‌های افزودن شده و augmented آموزش می دهیم:

```
for layer in base_model.layers:
    layer.trainable = False

lr_schedule = ExponentialDecay(initial_learning_rate=0.1, decay_steps=100000, decay_rate=0.002)

model.compile(loss='categorical_crossentropy',
              optimizer=optimizers.SGD(learning_rate=lr_schedule, momentum=0.9),
              metrics=['accuracy'])
```

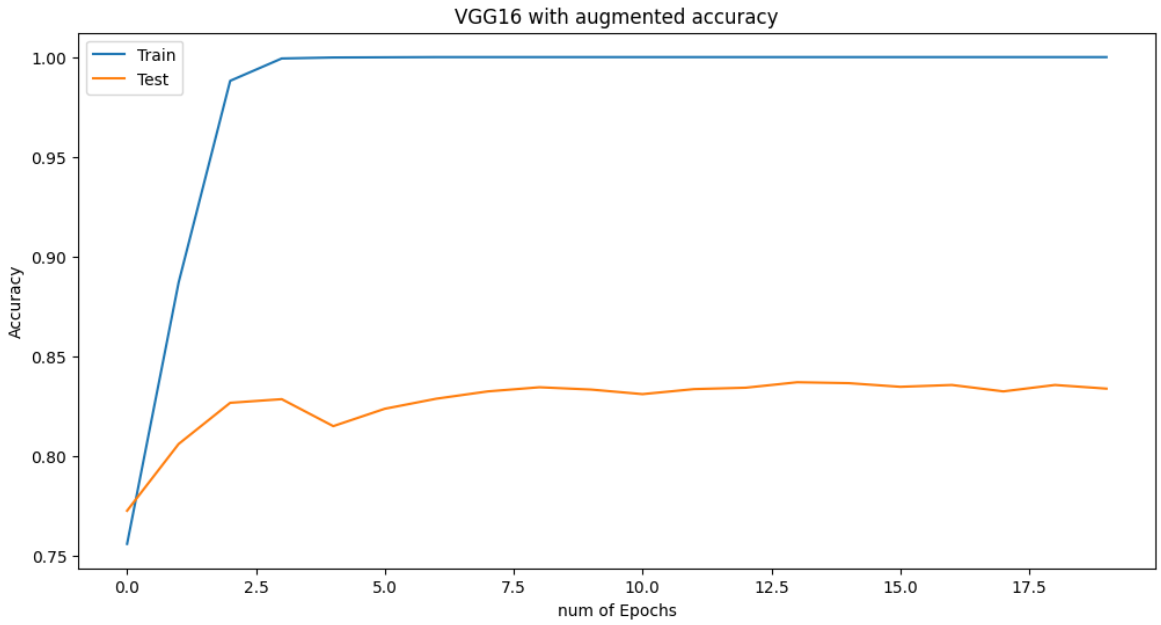
```
history = model.fit(
    augmented_train_generator,
    steps_per_epoch=len(augmented_train_generator),
    epochs=20,
    validation_data=augmented_validation_generator,
    validation_steps=len(augmented_validation_generator))
```

```
Epoch 1/20
80/80 [==============================] - 1719s 20s/step - loss: 0.8548 - accuracy: 0.7559 - val_loss: 0.5173 - val_accuracy: 0.7725
Epoch 2/20
80/80 [==============================] - 76s 943ms/step - loss: 0.2815 - accuracy: 0.8870 - val_loss: 0.4202 - val_accuracy: 0.8060
Epoch 3/20
80/80 [==============================] - 94s 1s/step - loss: 0.0393 - accuracy: 0.9881 - val_loss: 0.5075 - val_accuracy: 0.8266
Epoch 4/20
80/80 [==============================] - 75s 940ms/step - loss: 0.0064 - accuracy: 0.9993 - val_loss: 0.5102 - val_accuracy: 0.8285
Epoch 5/20
80/80 [==============================] - 75s 934ms/step - loss: 0.0030 - accuracy: 0.9998 - val_loss: 0.6416 - val_accuracy: 0.8150
Epoch 6/20
80/80 [==============================] - 75s 941ms/step - loss: 0.0020 - accuracy: 0.9999 - val_loss: 0.5993 - val_accuracy: 0.8237
Epoch 7/20
80/80 [==============================] - 94s 1s/step - loss: 0.0012 - accuracy: 1.0000 - val_loss: 0.5787 - val_accuracy: 0.8287
Epoch 8/20
```

همانطور که در کد مشخص است مجددا مدل را از اول ایجاد کردم و گام ها را پیش رفتم اما در گام سوم و چهارم، لایه های دست نخورده را فریز کرده و اینبار با داده، augmented_train_generator مدل را fine-tune کردم دقت مدل در حدود یک درصد بهبود یافت و داشتم:

```
[ ] loss, accuracy = model.evaluate(test_generator)
    print(f'Test accuracy: {accuracy}, Test loss: {loss}')

    1/1 [==============================] - 2s 2s/step - loss: 0.6139 - accuracy: 0.8700
    Test accuracy: 0.8700000047683716, Test loss: 0.6138901710510254
```

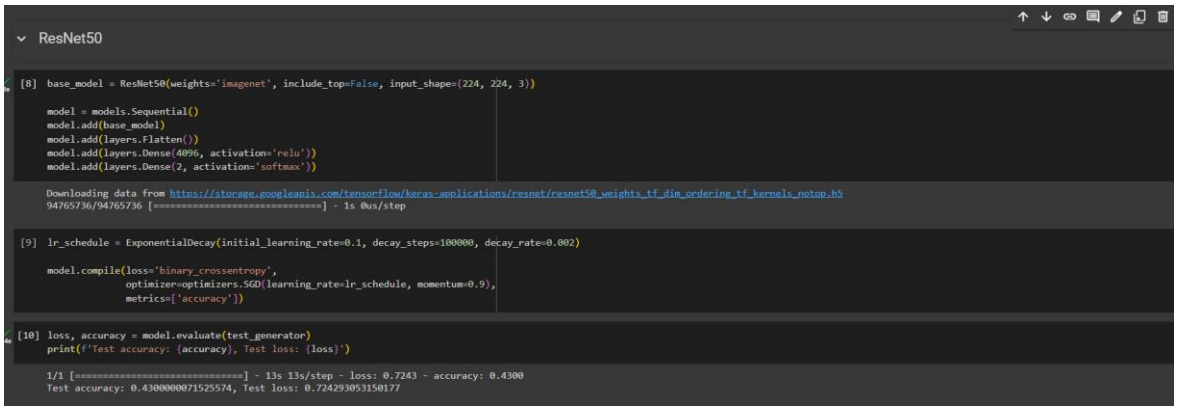
اما نکته مهم این بود که این بار با داده های افزون شده دقت بسیار سریع تر و در تعداد ایپاک کم به مقدار مد نظر رسید و نمودار دقت بدین صورت بود:



۱-۲-۳- ResNet50:

این شبکه‌ها با تفاوت کمی مانند VGG16 عمل کرده و داریم:

ابتدا بدون fine-tune کردن داریم:



سپس ابتدا لایه‌های مد نظر مقاله را freeze کرده و برروی داده‌های افزوده نشده (۷۰۲) آموزش می دهیم:

```
[11] for layer in base_model.layers:
        layer.trainable = False

    lr_schedule = ExponentialDecay(initial_learning_rate=0.01, decay_steps=100000, decay_rate=0.002)

    model.compile(loss='categorical_crossentropy',
                  optimizer=optimizers.SGD(learning_rate=lr_schedule, momentum=0.9),
                  metrics=['accuracy'])

history = model.fit(
    train_generator,
    steps_per_epoch=len(train_generator),
    epochs=20,
    validation_data=validation_generator,
    validation_steps=len(validation_generator))

Epoch 1/20
4/4 [==============================] - 31s 5s/step - loss: 110.6402 - accuracy: 0.5041 - val_loss: 0.6896 - val_accuracy: 0.5381
Epoch 2/20
4/4 [==============================] - 5s 1s/step - loss: 0.6882 - accuracy: 0.5528 - val_loss: 0.6839 - val_accuracy: 0.5000
Epoch 3/20
4/4 [==============================] - 5s 1s/step - loss: 0.8244 - accuracy: 0.5061 - val_loss: 0.7369 - val_accuracy: 0.5000
Epoch 4/20
4/4 [==============================] - 5s 1s/step - loss: 0.7139 - accuracy: 0.4980 - val_loss: 0.7011 - val_accuracy: 0.5048
Epoch 5/20
4/4 [==============================] - 6s 2s/step - loss: 0.7440 - accuracy: 0.5203 - val_loss: 0.7261 - val_accuracy: 0.5048
Epoch 6/20
4/4 [==============================] - 4s 1s/step - loss: 0.7264 - accuracy: 0.4919 - val_loss: 0.6891 - val_accuracy: 0.5000
Epoch 7/20
4/4 [==============================] - 6s 2s/step - loss: 0.7128 - accuracy: 0.5467 - val_loss: 0.7564 - val_accuracy: 0.5000
Epoch 8/20
4/4 [==============================] - 4s 1s/step - loss: 0.7401 - accuracy: 0.4776 - val_loss: 0.7118 - val_accuracy: 0.5000
Epoch 9/20
4/4 [==============================] - 6s 2s/step - loss: 0.7041 - accuracy: 0.5224 - val_loss: 0.6948 - val_accuracy: 0.5048
Epoch 10/20
```

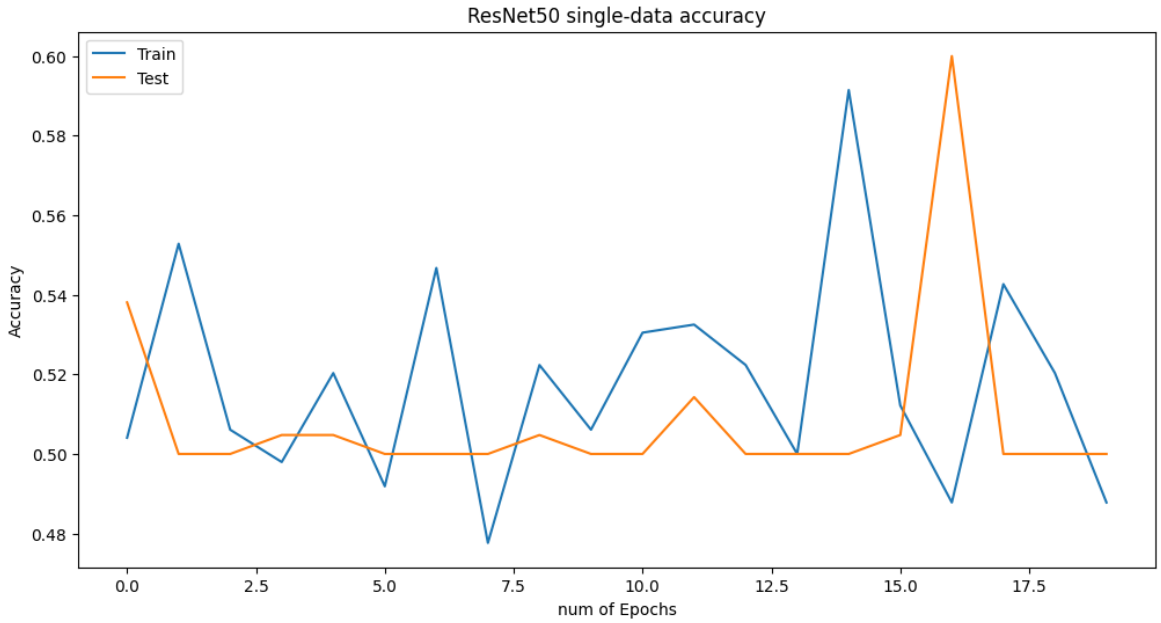
درادامه لایه های فریز شده باز کرده و مجدد ۳۰ ایپاک بعدی آموزش را تکرار می کنیم:

```
for layer in base_model.layers[-4:]:
    layer.trainable = True

model.compile(loss='categorical_crossentropy',
              optimizer=optimizers.SGD(learning_rate=lr_schedule, momentum=0.9),
              metrics=['accuracy'])

history = model.fit(
    train_generator,
    steps_per_epoch=len(train_generator),
    epochs=30,
    validation_data=validation_generator,
    validation_steps=len(validation_generator))

Epoch 2/30
4/4 [==============================] - 6s 2s/step - loss: 0.6904 - accuracy: 0.5427 - val_loss: 0.7043 - val_accuracy: 0.5000
Epoch 3/30
4/4 [==============================] - 5s 1s/step - loss: 0.9748 - accuracy: 0.4776 - val_loss: 0.6954 - val_accuracy: 0.5048
Epoch 4/30
4/4 [==============================] - 5s 1s/step - loss: 0.6906 - accuracy: 0.5915 - val_loss: 0.7024 - val_accuracy: 0.5048
Epoch 5/30
4/4 [==============================] - 5s 1s/step - loss: 0.6693 - accuracy: 0.5407 - val_loss: 0.7125 - val_accuracy: 0.5048
Epoch 6/30
4/4 [==============================] - 5s 1s/step - loss: 0.8318 - accuracy: 0.5041 - val_loss: 0.6929 - val_accuracy: 0.5095
Epoch 7/30
4/4 [==============================] - 5s 1s/step - loss: 0.6862 - accuracy: 0.5163 - val_loss: 0.6905 - val_accuracy: 0.5143
Epoch 8/30
4/4 [==============================] - 5s 1s/step - loss: 0.6751 - accuracy: 0.5915 - val_loss: 0.6754 - val_accuracy: 0.5524
Epoch 9/30
4/4 [==============================] - 6s 2s/step - loss: 0.6477 - accuracy: 0.6138 - val_loss: 0.7103 - val_accuracy: 0.5238
Epoch 10/30
4/4 [==============================] - 5s 1s/step - loss: 0.7105 - accuracy: 0.5549 - val_loss: 0.6771 - val_accuracy: 0.6143
Epoch 11/30
4/4 [==============================] - 5s 1s/step - loss: 0.6506 - accuracy: 0.6159 - val_loss: 0.6603 - val_accuracy: 0.6190
Epoch 12/30
```

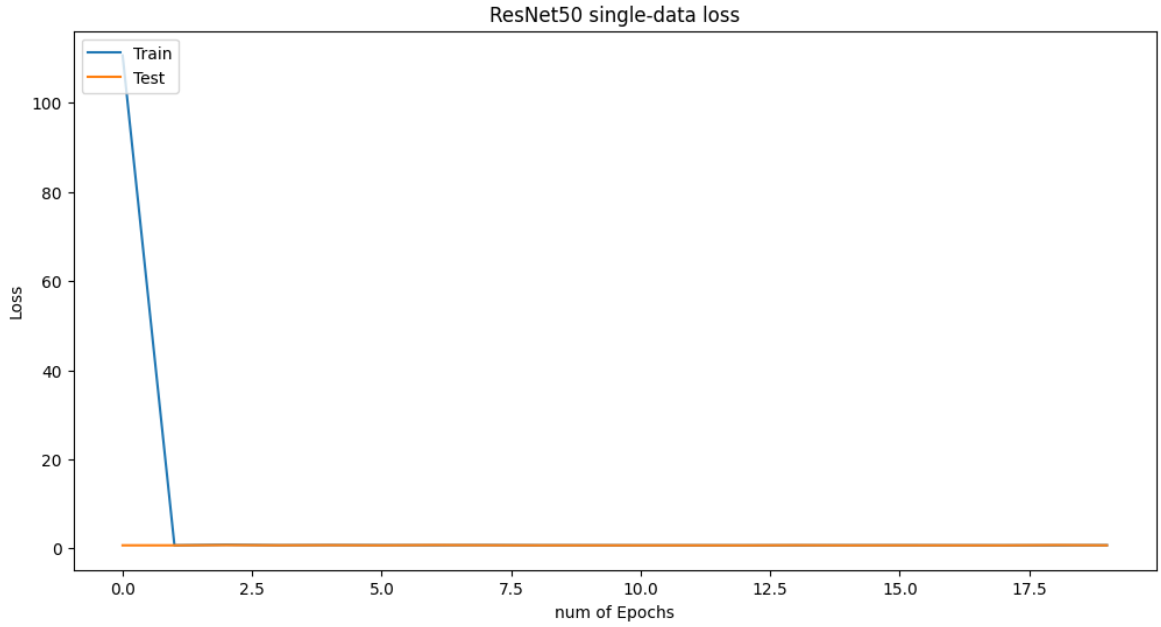
که دقت برروی داده های تست بدین صورت در آمد:

```
[13] loss, accuracy = model.evaluate(test_generator)
     print(f'Test accuracy: {accuracy}, Test loss: {loss}')

     1/1 [==============================] - 1s 854ms/step - loss: 0.6955 - accuracy: 0.5000
     Test accuracy: 0.5, Test loss: 0.6955238580703735
```

و برای نمودار دقت برای fine-tune رو داده های اندک در ۲۰ ایپاک اولیه داریم:



و برای نمودار loss در داده‌های اندک داریم:



در ادامه مجددا مدل را برای داده های augmented شده ایجاد کردم و طبق گام سوم لایه های از پیش آموزش دیده را فریز کرده:

```python
base_model = ResNet50(weights='imagenet', include_top=False, input_shape=(224, 224, 3))

model = models.Sequential()
model.add(base_model)
model.add(layers.Flatten())
model.add(layers.Dense(4096, activation='relu'))
model.add(layers.Dense(2, activation='softmax'))
```

```python
lr_schedule = ExponentialDecay(initial_learning_rate=0.01, decay_steps=100000, decay_rate=0.002)

model.compile(loss='binary_crossentropy',
              optimizer=optimizers.SGD(learning_rate=lr_schedule, momentum=0.9),
              metrics=['accuracy'])
```

```python
for layer in base_model.layers:
    layer.trainable = False

lr_schedule = ExponentialDecay(initial_learning_rate=0.01, decay_steps=100000, decay_rate=0.002)

model.compile(loss='categorical_crossentropy',
              optimizer=optimizers.SGD(learning_rate=lr_schedule, momentum=0.9),
              metrics=['accuracy'])
```

و بر روی داده های augmented شده fine-tune کردم:

```python
history = model.fit(
    augmented_train_generator,
    steps_per_epoch=len(augmented_train_generator),
    epochs=20,
    validation_data=augmented_validation_generator,
    validation_steps=len(augmented_validation_generator))
```

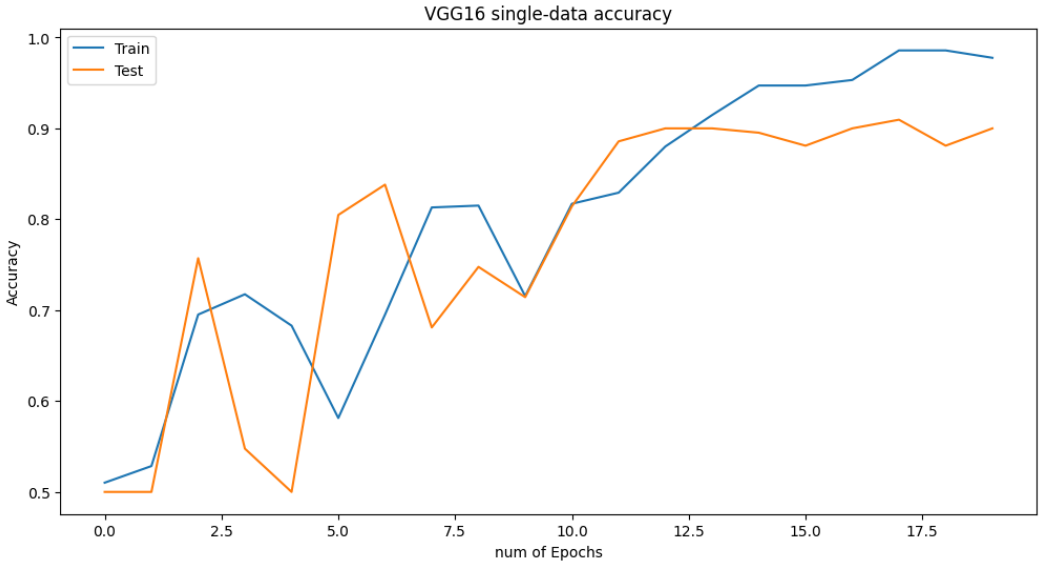
```
Epoch 1/20
192/192 [==============================] - 224s 1s/step - loss: 1.4552 - accuracy: 0.5276 - val_loss: 0.6995 - val_accuracy: 0.4994
Epoch 2/20
192/192 [==============================] - 184s 960ms/step - loss: 0.6933 - accuracy: 0.5289 - val_loss: 0.6787 - val_accuracy: 0.5109
Epoch 3/20
192/192 [==============================] - 212s 1s/step - loss: 0.6910 - accuracy: 0.5107 - val_loss: 0.6848 - val_accuracy: 0.5014
Epoch 4/20
192/192 [==============================] - 212s 1s/step - loss: 0.6922 - accuracy: 0.5069 - val_loss: 0.6928 - val_accuracy: 0.5013
Epoch 5/20
192/192 [==============================] - 212s 1s/step - loss: 0.6905 - accuracy: 0.5143 - val_loss: 0.6936 - val_accuracy: 0.4986
Epoch 6/20
192/192 [==============================] - 212s 1s/step - loss: 0.6906 - accuracy: 0.5048 - val_loss: 0.6786 - val_accuracy: 0.6491
Epoch 7/20
192/192 [==============================] - 191s 991ms/step - loss: 0.6853 - accuracy: 0.5320 - val_loss: 0.6763 - val_accuracy: 0.5618
Epoch 8/20
192/192 [==============================] - 185s 962ms/step - loss: 0.6866 - accuracy: 0.5257 - val_loss: 0.6932 - val_accuracy: 0.5014
Epoch 9/20
192/192 [==============================] - 219s 1s/step - loss: 0.6931 - accuracy: 0.5004 - val_loss: 0.6931 - val_accuracy: 0.5014
Epoch 10/20
```

۴-۲- نتایج و تحلیل:

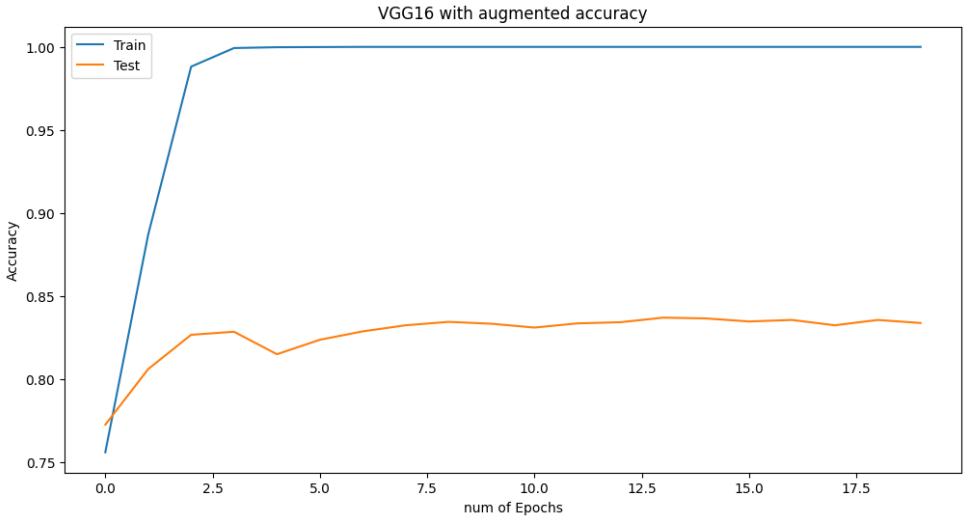
برای **دقت** داریم:

VGG16:

دیتاست **augment نشده**:

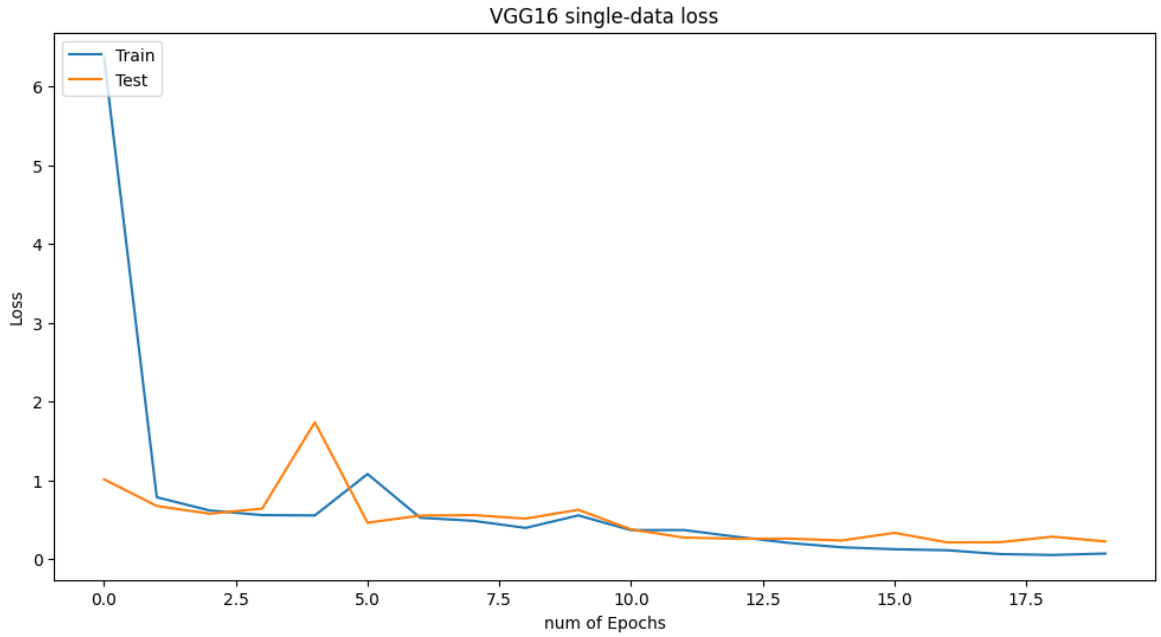


و برروی دیتاست **augment شده**:



و برای **Loss** در هر یک از دو دیتاست داریم:

برروی دیتاست augment نشده:

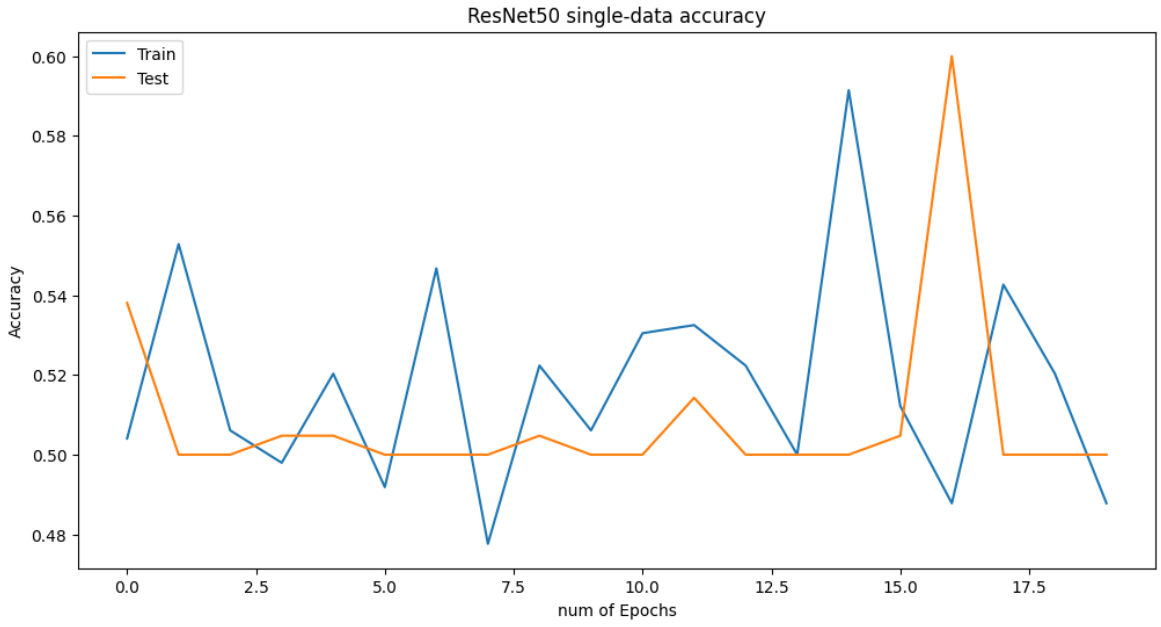


ResNet50:

برای مدل ResNet50 نیز داریم:

برای نمودار **دقت**:

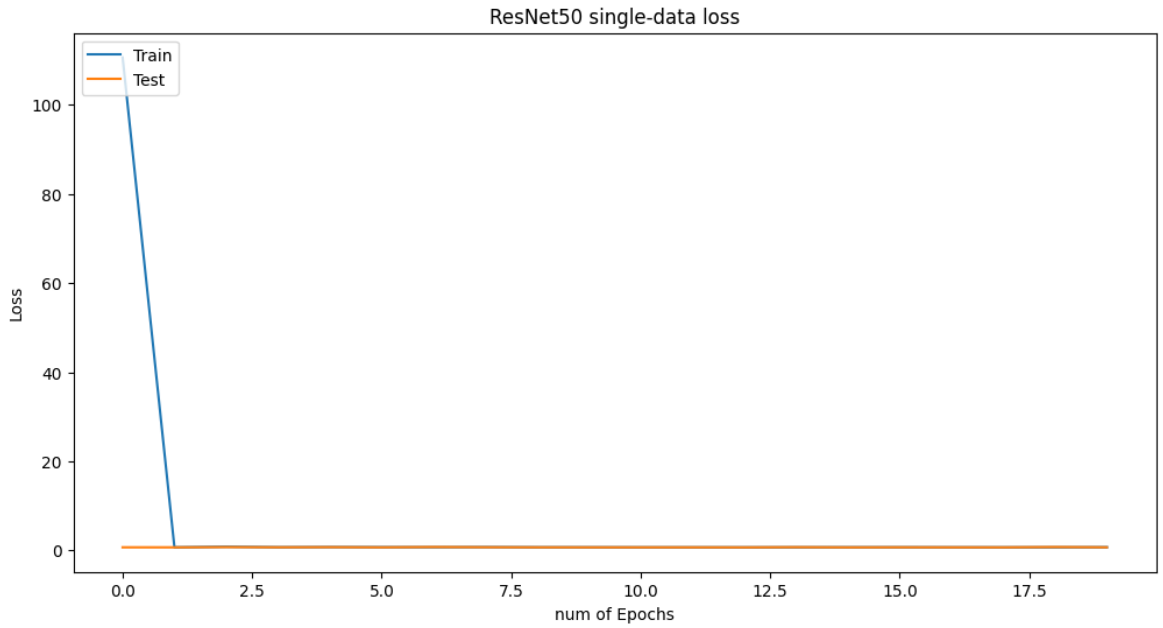
برای داده **augment نشده**:



برای داده **augment شده**:

برای نمودار Loss:

برای داده augment نشده



برای داده augment شده

جدول مقایسه:

Model	VGG16		ResNet50	
	Augmented	Not Augmented	Augmented	Not Augmented
Training Accurecy	100%	98.58%	69.06%	74.4%
Validation Accurecy	83.5%	90.95%	64.9%	60%
Test Accurecy	87%	89.9%	63.2%	69.5%