به نام خدا

دانشگاه تهران

دانشکده مهندسی برق و کامپیوتر

درس شبکه‌های عصبی و یادگیری عمیق

تمرین دوم

امیرعباس رضا سلطانی	610399205
نیما نیرومند	610399199

فهرست

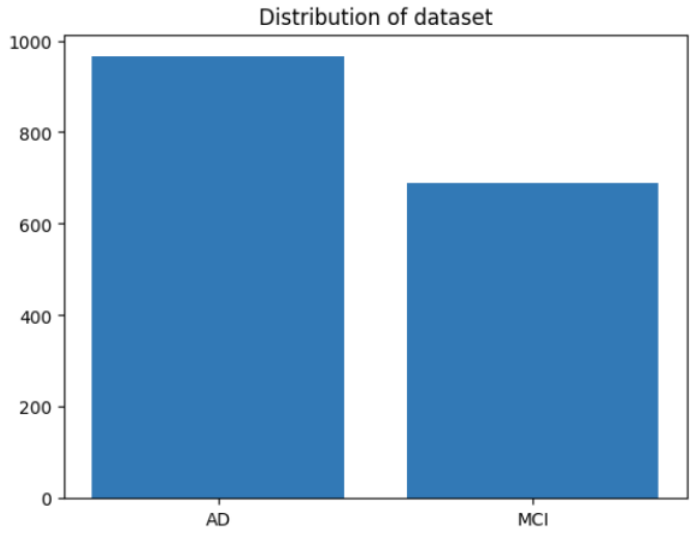
شکل‌ها

جدول‌ها

پرسش ۱.تشخیص آلزایمر با استفاده از تصویر برداری مغزی(ADNI)

۱-۲. پیش پردازش تصاویر

ابتدا مجموعه داده ADNI را می‌خوانیم که دارای ۱۶۵۴ عکس MRI از مغز می‌باشد و مشاهده می‌کنیم که ۹۶۵ تای آن‌ها از دسته‌ی AD و ۶۸۹ تای آن‌ها از دسته‌ی MCI می‌باشند



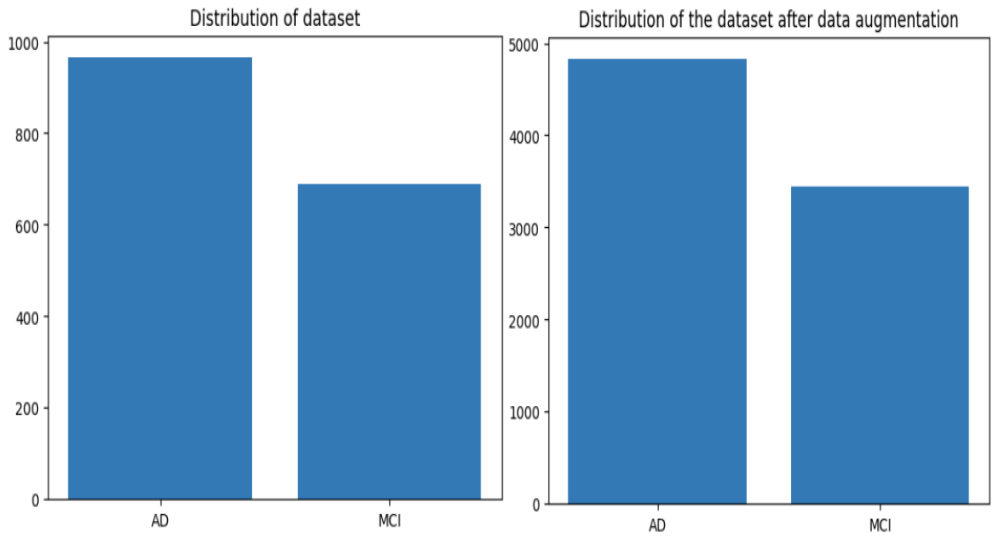
شکل 1. توزیع آماری کلاس‌های مجموعه داده

از آنجایی که تصاویر ما خاکستری هستند پس آن‌ها را به شکل grayscale و همچنین در سایر ۶۴در۶۴ می‌خوانیم پس یک ماتریس دوبعدی به اندازه ۶۴*۶۴ خواهیم داشت که در آن اعداد در بازه‌ی ۰ تا ۲۵۵ (که ۰ نشان دهنده سیاه و ۲۵۵ نشان دهنده سفید و باقی اعداد ترکیبی از این دو و خاکستری خواهند بود.) حال می‌خواهیم که داده‌های ورودی را نرمال‌سازی کنیم که یعنی تمامی مقادیر را به ۰ تا ۱ نگاشت کنیم. به این منظور از آنجایی که اعداد ورودی در بازه ۰ تا ۲۵۵ هستند پس همه را بر ۲۵۵ تقسیم می‌کنیم تا به بازه ۰ تا ۱ نگاشت شوند که با این کار آموزش مدل سریع‌تر می‌شود و زودتر می‌تواند به نقطه همگرا برسد چرا که می‌دانیم در الگوریتم backpropagation از لایه خروجی به سمت ورودی حرکت می‌کنیم و می‌دانیم که برای محاسبه گرادیان از ضرب مشتق‌های جزئی استفاده می‌کنیم و هرچه که به لایه‌های عقب تر می‌ریم گرادیان حاصل ضرب عناصری بیشتری خواهند بود و حاصل ضرب چند عدد بزرگ عدد بزرگی می‌شود و اگر خیلی به عقب برویم ممکن است مقدار بسیار بزرگی شود و به اصطلاح دچار انفجار گرادیان شویم و باعث شود که مدل به خوبی آموزش نبیند و همگرا شود

۱-۳. داده افزایی Data augmentation

اگر بخواهیم که مدل ما نتیجه خوبی داشته باشد ساختار آن بزرگ خواهد بود اما اگر داده‌های لیبل‌دار ما کم باشند مدل به خوبی آموزش نخواهد دید(واگر معماری کوچک باشد دچار بیش برازش می‌شویم) پس کمک روش‌های Data augmentation می‌توانیم تعداد داده‌ها را افزایش دهیم و علاوه بر اینکه مدل ما می‌تواند داده‌های بیشتری را ببیند و نتیجه بهتری ارائه دهد می‌توانیم که به علت اینکه در حالت‌های بیشتر و متفاوتی عکس های ورودی را ببیند از بیش برازش جلوگیری کند و مدل ما انعطاف پذیری بیشتری داشته باشد.

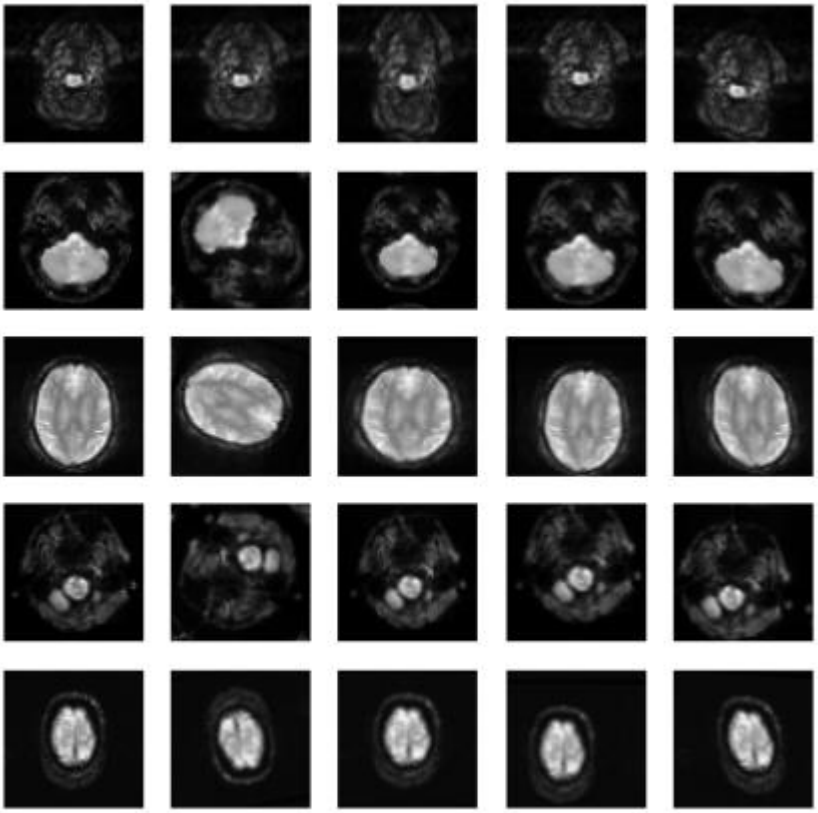
پس بدین منظور با روش‌های افزایش داده از هر تصویر،۴تصویر دیگر به وجود می‌آوریم و در مجموع تعداد تصاویر ما ۵ برابر خواهد شد



شکل۲. توزیع آماری کلاس‌های مجموعه داده قبل و بعد از تقویت داده‌ها

همانطور که انتظار داشتیم به علت اینکه تصاویر ما ۵ برابر شدند پس توزیع آماری تصاویر بین کلاس‌های ما یکسان خواهند ماند.

حال به صورت تصادفی ۵ تا از داده‌های اصلی را انتخاب و آن‌ها تحت روش های مختلف داده افزایی استفاده شده نمایش می‌دهیم.



شکل۳. تصاویر ۵ تا دیتا تصادفی بعد از تقویت داده ها

۱-۴. پیاده‌سازی

ابتدا در مورد مقداردهی اولیه Glorot توضیح می‌دهیم می‌دانیم که پارامترهای شبکه به صورت تصادفی وزن‌دهی می‌شود و وزن لایه‌های محتلف نیز یکی از پارامترهای شبکه می‌باشد حال فرض کنیم برای مثال که شبکه ما ۲۵۶ ورودی داشته باشد و همه یک باشند حال یک نورون از لایه پنهان اول را در نظر می‌گیریم و فرض کنیم که میانگین وزن‌های اولیه تصادفی ما ۰ و انحراف از معیار ۱ حال بدیهتا نورون ما خودش یک توزیع نرمال با میانگین نزدیک به صفر خواهد بود اما واریانس آن ۲۵۶ و در نتیجه انحراف معیر آن ۱۶ خواهد بود که بسیار بیشتر از انحراف معیار اولیه ۱ خواهد بود پس در این مدل انحراف معیار خروجی هر لایه بیشتر از ورودی آن خواهد بود و هرچی به جلو برویم این واریانس بیشتر خواهد شد و این واریانس زیاد باعث می‌شود که مقدار ما بسیار کوچکتر یا بسیار بزرگتر از یک شود و مثلا از تابع فعال ساز ما سیگموید باشد آنگاه چون مقدار ما یا خیلی بزرگ هستند یا کوچک پس مقدار ان به ۱ یا ۰ نزدیک می شود که میدانیم مشتق در آن نقاط نزذیک صفر می شود و این امر باعث می شود در روش آموزش backpropagation گرادیان ما نزدیک به صفر شود و أمورشی صورت نگیرد مخصوصا در لایه‌های عقب تر و گرادیان محو شود حال در روش glorot همانظور که دیدیم واریانس خروجی از ورودی تاثیر می‌گیرد پس اگر می‌خواهیم واریانس خروجی را کوچک کنیم باید واریانس این پارامترها را کوچک کنیم و یکی از

راه‌های آن این است که از تعداد ورودی یک لایه و خروجی یک لایه میانگین بگیریم و مقدار واریانس پارامترها را یک بر روی این مقدار قرار دهیم.

حال به سراغ پیاده سازی مدل‌ها میرویم(که در آن‌ها dropout و وزن دهی اولیه glorot نباشه را در نظر می‌گیریم.)

Layer (type)	Output Shape	Param #
conv2d_32 (Conv2D)	(None, 62, 62, 32)	320
batch_normalization_50 (BatchNormalization)	(None, 62, 62, 32)	128
activation_60 (Activation)	(None, 62, 62, 32)	0
conv2d_33 (Conv2D)	(None, 60, 60, 32)	9,248
batch_normalization_51 (BatchNormalization)	(None, 60, 60, 32)	128
activation_61 (Activation)	(None, 60, 60, 32)	0
max_pooling2d_18 (MaxPooling2D)	(None, 30, 30, 32)	0
conv2d_34 (Conv2D)	(None, 28, 28, 32)	9,248
batch_normalization_52 (BatchNormalization)	(None, 28, 28, 32)	128
activation_62 (Activation)	(None, 28, 28, 32)	0
conv2d_35 (Conv2D)	(None, 26, 26, 32)	9,248
batch_normalization_53 (BatchNormalization)	(None, 26, 26, 32)	128
activation_63 (Activation)	(None, 26, 26, 32)	0
max_pooling2d_19 (MaxPooling2D)	(None, 13, 13, 32)	0
flatten_10 (Flatten)	(None, 5408)	0
dense_28 (Dense)	(None, 128)	692,352
batch_normalization_54 (BatchNormalization)	(None, 128)	512
activation_64 (Activation)	(None, 128)	0
dense_29 (Dense)	(None, 64)	8,256
batch_normalization_55 (BatchNormalization)	(None, 64)	256
activation_65 (Activation)	(None, 64)	0
dense_30 (Dense)	(None, 2)	130
activation_66 (Activation)	(None, 2)	0

شکل۴. خلاصه **proposed_model**

Layer (type)	Output Shape	Param #
conv2d_36 (Conv2D)	(None, 62, 62, 32)	320
batch_normalization_56 (BatchNormalization)	(None, 62, 62, 32)	128
activation_67 (Activation)	(None, 62, 62, 32)	0
max_pooling2d_20 (MaxPooling2D)	(None, 31, 31, 32)	0
conv2d_37 (Conv2D)	(None, 29, 29, 32)	9,248
batch_normalization_57 (BatchNormalization)	(None, 29, 29, 32)	128
activation_68 (Activation)	(None, 29, 29, 32)	0
max_pooling2d_21 (MaxPooling2D)	(None, 14, 14, 32)	0
flatten_11 (Flatten)	(None, 6272)	0
dense_31 (Dense)	(None, 128)	802,944
batch_normalization_58 (BatchNormalization)	(None, 128)	512
activation_69 (Activation)	(None, 128)	0
dense_32 (Dense)	(None, 2)	258
activation_70 (Activation)	(None, 2)	0

شکل۵. خلاصه **testin_model_1**

Layer (type)	Output Shape	Param #
conv2d_38 (Conv2D)	(None, 62, 62, 32)	320
batch_normalization_59 (BatchNormalization)	(None, 62, 62, 32)	128
activation_71 (Activation)	(None, 62, 62, 32)	0
conv2d_39 (Conv2D)	(None, 60, 60, 32)	9,248
batch_normalization_60 (BatchNormalization)	(None, 60, 60, 32)	128
activation_72 (Activation)	(None, 60, 60, 32)	0
max_pooling2d_22 (MaxPooling2D)	(None, 30, 30, 32)	0
flatten_12 (Flatten)	(None, 28800)	0
dense_33 (Dense)	(None, 128)	3,686,528
batch_normalization_61 (BatchNormalization)	(None, 128)	512
activation_73 (Activation)	(None, 128)	0
dense_34 (Dense)	(None, 64)	8,256
batch_normalization_62 (BatchNormalization)	(None, 64)	256
activation_74 (Activation)	(None, 64)	0
dense_35 (Dense)	(None, 2)	130
activation_75 (Activation)	(None, 2)	0

شکل۶. خلاصه testin_model_2

Layer (type)	Output Shape	Param #
conv2d_40 (Conv2D)	(None, 62, 62, 32)	320
batch_normalization_63 (BatchNormalization)	(None, 62, 62, 32)	128
activation_76 (Activation)	(None, 62, 62, 32)	0
dropout_6 (Dropout)	(None, 62, 62, 32)	0
conv2d_41 (Conv2D)	(None, 60, 60, 32)	9,248
batch_normalization_64 (BatchNormalization)	(None, 60, 60, 32)	128
activation_77 (Activation)	(None, 60, 60, 32)	0
max_pooling2d_23 (MaxPooling2D)	(None, 30, 30, 32)	0
conv2d_42 (Conv2D)	(None, 28, 28, 32)	9,248
batch_normalization_65 (BatchNormalization)	(None, 28, 28, 32)	128
activation_78 (Activation)	(None, 28, 28, 32)	0
dropout_7 (Dropout)	(None, 28, 28, 32)	0
conv2d_43 (Conv2D)	(None, 26, 26, 32)	9,248
batch_normalization_66 (BatchNormalization)	(None, 26, 26, 32)	128
activation_79 (Activation)	(None, 26, 26, 32)	0
max_pooling2d_24 (MaxPooling2D)	(None, 13, 13, 32)	0
flatten_13 (Flatten)	(None, 5408)	0
dense_36 (Dense)	(None, 128)	692,352
batch_normalization_67 (BatchNormalization)	(None, 128)	512
activation_80 (Activation)	(None, 128)	0
dense_37 (Dense)	(None, 64)	8,256
batch_normalization_68 (BatchNormalization)	(None, 64)	256
activation_81 (Activation)	(None, 64)	0
dropout_8 (Dropout)	(None, 64)	0
dense_38 (Dense)	(None, 2)	130
activation_82 (Activation)	(None, 2)	0

شکل۷. خلاصه **proposed_model_with_dropout**

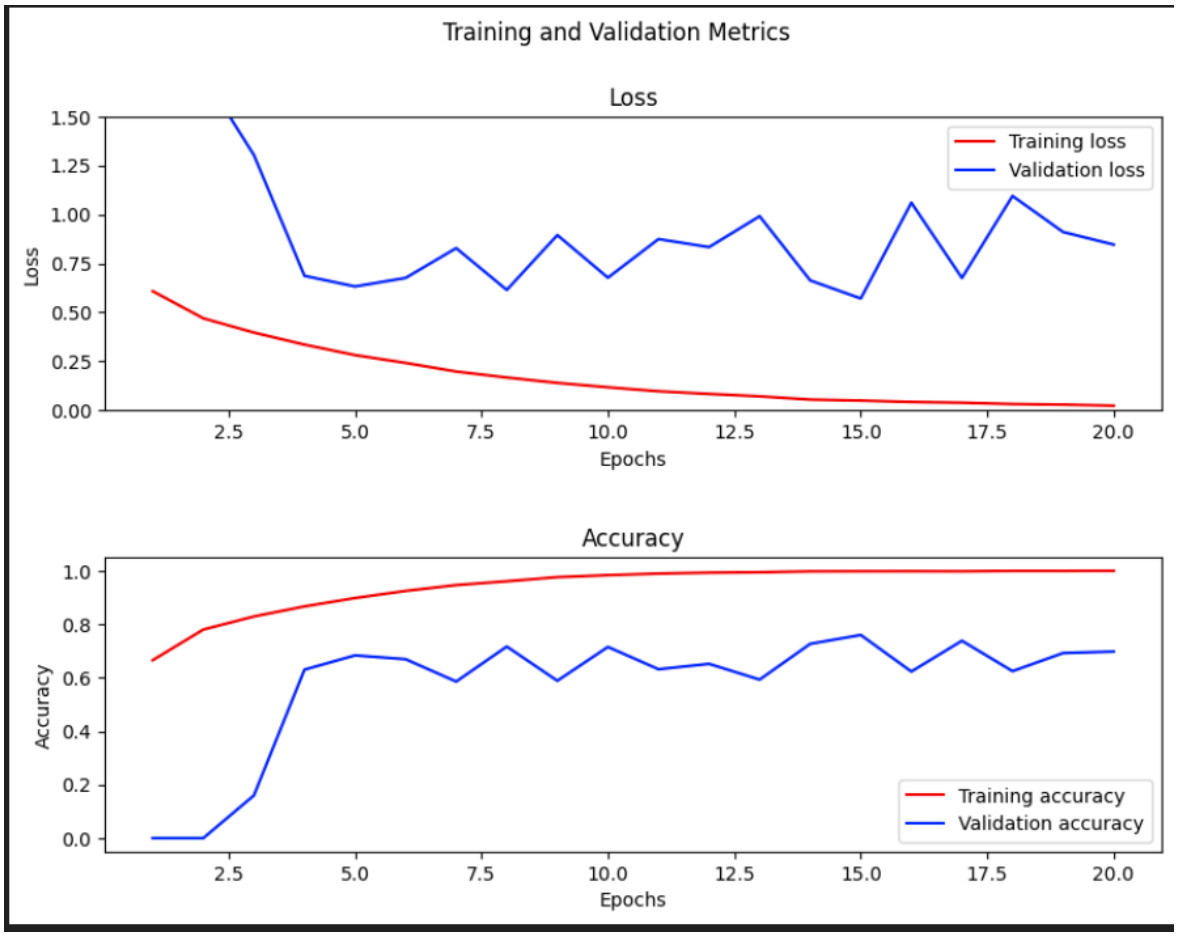
Layer (type)	Output Shape	Param #
conv2d_44 (Conv2D)	(None, 62, 62, 32)	320
batch_normalization_69 (BatchNormalization)	(None, 62, 62, 32)	128
activation_83 (Activation)	(None, 62, 62, 32)	0
conv2d_45 (Conv2D)	(None, 60, 60, 32)	9,248
batch_normalization_70 (BatchNormalization)	(None, 60, 60, 32)	128
activation_84 (Activation)	(None, 60, 60, 32)	0
max_pooling2d_25 (MaxPooling2D)	(None, 30, 30, 32)	0
conv2d_46 (Conv2D)	(None, 28, 28, 32)	9,248
batch_normalization_71 (BatchNormalization)	(None, 28, 28, 32)	128
activation_85 (Activation)	(None, 28, 28, 32)	0
conv2d_47 (Conv2D)	(None, 26, 26, 32)	9,248
batch_normalization_72 (BatchNormalization)	(None, 26, 26, 32)	128
activation_86 (Activation)	(None, 26, 26, 32)	0
max_pooling2d_26 (MaxPooling2D)	(None, 13, 13, 32)	0
flatten_14 (Flatten)	(None, 5408)	0
dense_39 (Dense)	(None, 128)	692,352
batch_normalization_73 (BatchNormalization)	(None, 128)	512
activation_87 (Activation)	(None, 128)	0
dense_40 (Dense)	(None, 64)	8,256
batch_normalization_74 (BatchNormalization)	(None, 64)	256
activation_88 (Activation)	(None, 64)	0
dense_41 (Dense)	(None, 2)	130
activation_89 (Activation)	(None, 2)	0

شکل۸. خلاصه **proposed_model_without_glorot**

حال برای کامپایل کردن مدل‌ها تابع زیانی که در نظر می‌گیریم تابع زیانه Categorical Cross Entorpy است که همان طور که از نام آن مشخص هست مناسب مسائل چند کلاسه می‌باشد که احتمالا پیش بینی هر دسته را می‌گیرد و با دسته حقیقی مقایسه و طبق اختلاف آن‌ها جریمه مدل را مشخص می‌کند اما باید لیبیل‌های ما one-hot باشند و همچنین تابع فعال ساز ما softmax که به صورت احتمالاتی از دسته‌های ما خروجی می‌دهد

همچنین بهینه‌ساز adam با نرخ یادگیری ۰.۰۰۱ را استفاده می‌کنیم.

۱-۵. تحلیل نتایج

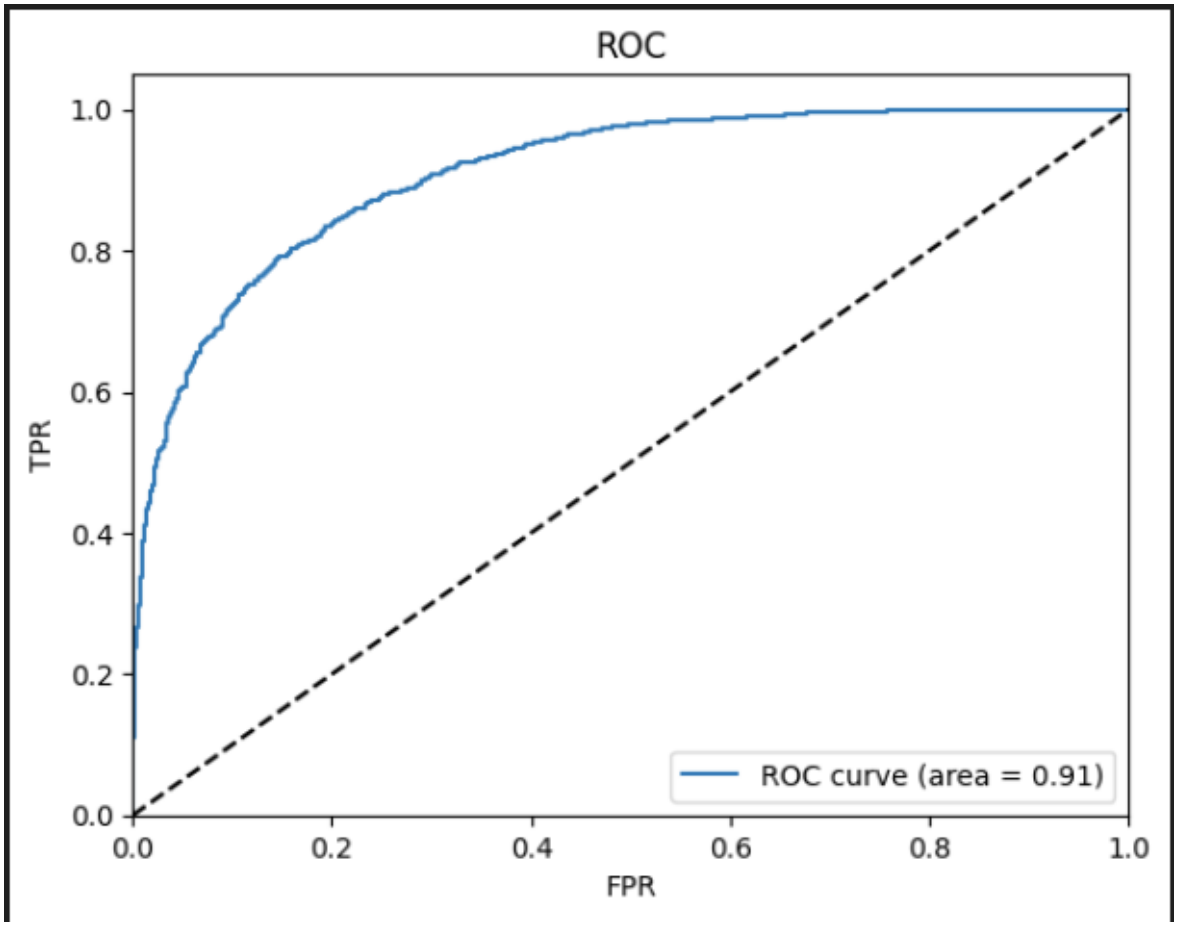


شکل۹. نمودار دقت و خطا در دادگان آموزشی و ارزیابی

همانطور که دیده مدل ما دچار بیش برازش میشود و زیان ما روی داده‌های ترین کم می‌شود اما روی ارزیابی زیاد که یعنی آموزش را بیش از حد خوب یاد گرفته و عمومیت خودشو از دست داده است همچنین دقت برای زمانی که کلاسای ما توزیع یکسنان ندارند نمیتواند توضیح خوبی باشد برای عملکرد مدل

ROC صرفا برای مسائل دوکلاسه است و نشان می‌دهد مدل ما چقدر تونسته بین کلاسای مختلف تمایز قائل شود و به درستی آن‌ها را تشخیص دهد از آنجایی که در این مسئله داده‌های ما نامتوازن هستند پس دقت معیار خوبی نیست معیار ارزیابی ROC_AUC ، از دو مفهوم TPR و FPR ایجاد شده که به ترتیب نشان‌دهنده این هستند که چقدر از نمونه‌های کلاس به درستی مثبت و چقدر از نمونه‌های منفی به اشتباه مثبت اعلام شده اند حال ما در اینجا از تابع فعال ساز softmax استفاده کرده‌ایم و نشان‌دهنده این هست که به صورت احتمالی می‌توانیم نشان دهیم که خروجی مربوط به کدام کلاس است و ما در نمودار roc-auc ، TPR در مقابل FPR را رسم میکنیم و میتواند نشان دهد که کجا حد

آستانه بهتری برای تمایز بین دو کلاس است(یعنی اگر از یه احتمالی بیشتر بودن یک کلاس در خروجی ، خروجی را آن کلاس در نطر بگیریم) و AUC مساحت زیر نمودار است و عددی بین یک و صفر است و هرچقدر که که بیشتر باشد یعنی مدل ما بهتر عمل کرده و توانسته تمایز بهتری بین دو کلاس در نظر بگیرد و مشاهده می‌کنیم که مساحت زیر نمودار ما ۸۱ صدم هست که یعنی مدل ما نسبتا در تمایز بین کلاس‌ها خوب عمل کرده است



شکل۱۰. نمودار **proposed_model ROC**

```
Confusion Matrix:
[[ 939  302]
 [ 236 1501]]
Classification Report:
              precision    recall  f1-score   support

          AD       0.80      0.76      0.78      1241
         MCI       0.83      0.86      0.85      1737

    accuracy                           0.82      2978
   macro avg       0.82      0.81      0.81      2978
weighted avg       0.82      0.82      0.82      2978

AUC:
0.8103907141203656
```

شکل۱۱ صحت عملکرد **proposed_model**

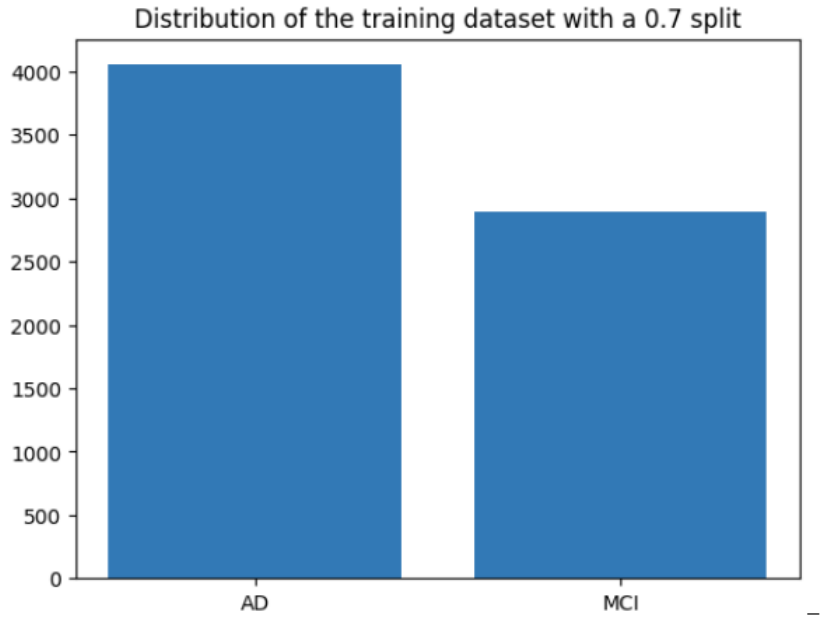
مشاهد می‌شود که ۹۳۹ تا کلاس مثبت درست، ۳۰۲ تا به اشتباه مثبت اعلام شده و ۲۳۶ تا به اشتباه منفی و ۱۵۰۱ تا به درست منفی اعلام شده است .

Precision نشان می‌دهد که چقدر از پیش بینی های کلاس مثبت درست هستند و Recall نشان میدهد که چقدر از نمونه‌هایی که واقعا مثبت هستند مثبت تشخیص داده شده اند اما هردوی این ویژگی ها به تنهایی ممکن است اطلاعات مفیذی نداشته باشند برای مثال اگر مدل ما همه چیز را کلاس مثبت اعلام کند آنگاه ریکال ۱ میشود اما مدل خوبی نداریم برای همین اگر بخواهیم هردوی این اطلاعات سود مند را ترکیب کنیم . از هردوی آنها استفاده کنیم و تعادل بیشتری داشته باشیم f1 که از ترکیب صحت و ریکال (میانگین هارمونیک آنها)است بطور میانگین ۸۲ درصد شده است که مشاهده میکنیم به طور کلی مدل ما رو داده‌های تست خوب عمل کرده است و همچنین AUC ، ۸۱ درصد شده است که نشان میدهد مدل ما عملکرد خوبی را دارد.

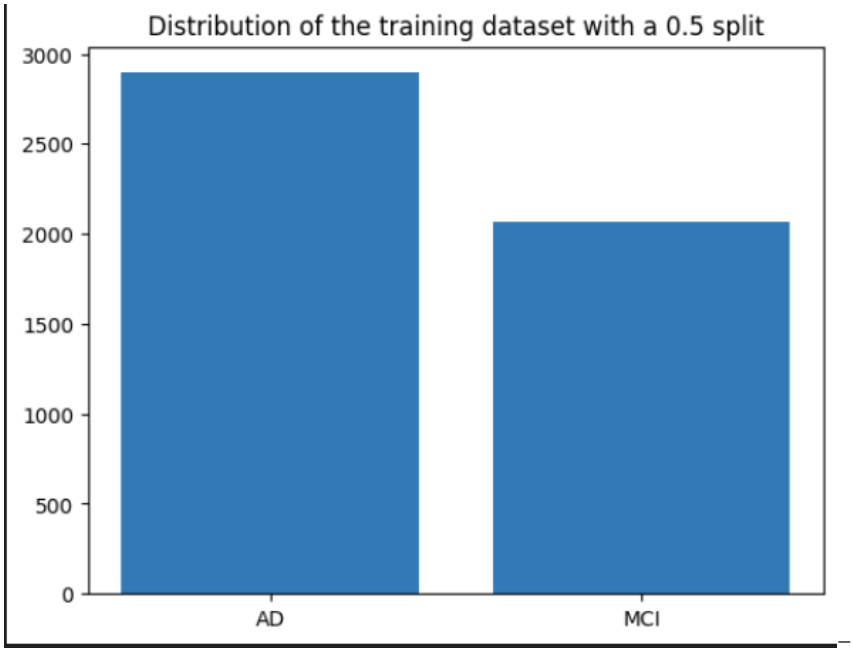
۱-۶. مقایسه نتایج

-ابتدا داده‌های تست را ۰.۳ . سپس .۵ در نظر می‌گیریم و همانطور که در نمودارا میبینیم توزیع کلی آماری بین دو کلاس تغییری نکرده است

حال auc ما در حالت اول ۸۱ و در حالت دوم ۷۹ که همانطور که انتظار داشتیم زمانی که داده‌های بیشتری برای اموزش داشته باشیم نتیجه بهتری را می‌گیریم.



شکل۱۲. نمودار توزیع آماری داده‌ها ترین پس از تقسیم ۰.۷ شدن



شکل۱۳. نمودار توزیع آماری داده‌ها ترین پس از تقسیم ۰.۵ شدن

- پس از اعمال حذف تصادفی auc ۸۳ میشود و همین چنین f1 معیار از ۸۲ به ۸۳ میرسد که هامنطور که در تحلیل نمودار گفتیم دچار بیش برازش شدیم و با اعمال حذف تصادفی نتیجه ما بهتر باید بشود روی تست و عمومیت بیشتری داشته باشد و نتیجه بهتری را بگیرد

- با حذف وزن دهی اولیه گلوروت و وزن دهی اولیه نرمال با میانیک ۰ و انحراف معیار ۰.۰۵ همانطور که انتظار داشتیم واریانس خروجی ها زیاد و در نتیجه گرادیان در لایه‌های عقب تر بدتر عمل میکند و مدل به وخبی آموزش نمیبیند و مطابق انتظار ما auc ۸۰ و f1 ما از ۸۲ به ۸۱ میرسد و مدل بدتر عمل میکند

- auc به ترتیب ۸۱و ۸۰.۸ و ۸۰ میشود که همانطور که انتظار داشتیم پی‌چیدگی بیشتر مدل باعث نتیجه بهتر شده است و مدل بهتر یاد می‌گیرد(به ترتیب مدل‌های proposed model و testing model 2 و testing model 1)

.

پرسش ۲ – بررسی تاثیر افزایش داده بر عملکرد شبکه های کانولوشنیFine-Tuneشده

۲-۲ پیش پردازش تصاویر.

در دیتاست داده شده، ۳۵۰ عکس گربه و ۳۵۰ عکس سگ موجود است. تمام عکس‌ها را به ابعاد (۳ **,۱۲۸ ,۱۲۸) تغییر بُعد** می‌دهیم.**۱۰۰ عکس سگ و ۱۰۰ عکس گربه از پیش به عنوان داده‌های تست** گزیده شده‌اند. از میان ۵۰۰ عکس باقی‌مانده، **۳۰٪ را عنوان داده‌های** validation و ۷۰٪ را به عنوان داده‌های **train** به صورت رندوم برمی‌گزینیم. از سه روش **بازتاب افقی، زوم کردن در بازه (۲۵٪+، ۲۵٪-)، و چرخاندن (۳۰+، ۳۰-) درجه** برای **augment** کردن داده‌های **train** استفاده می‌کنیم. برای **augment** کردن داده‌ها از لایه‌های مرتبط در کتابخانه **keras** استفاده می‌کنیم. در واقع داده ورودی از این لایه‌ها گذشته و به صورت رندوم در بازه‌های معین شده مورد تغییر قرار می‌گیرند. از آنجا که تعداد **epoch** ها ۲۵ قبل از **fine-tune** کردن و ۲۵ هنگام از **fine-tune** است، و همچنین تعداد داده‌های آموزش، ۳۵۰ است، بنابراین ۳۵۰ **x** ۵۰ یعنی **۱۷۵۰۰ داده مختلف** بعد از **augment** شدن به شبکه آموزش داده می‌شود (با این فرض که یک داده در دو **epoch** متفاوت، دچار **augment** یکسان نمی‌شود).

۲-۳. پیاده سازی

مدل VGG16:

از مدل VGG16 در کتابخانه keras استفاده می‌کنیم. قسمت fully connected این شبکه را حذف و از باقی مانده شبکه (لایه‌های pooling و convolution) برای استخراج ویژگی‌ها استفاده می‌کنیم. خروجی شبکه cnn را با استفاده از لایه flatten به صورت بردار درمی‌آوریم. سپس از دو لایه fully connected با ۱۵ نورون و تابع فعال‌ساز relu استفاده می‌کنیم. بعد از اعمال هر لایه fully connected، یک لایه dropout و یک لایه batch normalizer به کار می‌بریم. در آخر از یک لایه با یک نورون با تابع فعال‌ساز sigmoid استفاده می‌کنیم که خروجی آن به‌صورت احتمال تعلق به دسته سگ تعبیر می‌شود. از بهینه‌ساز sgd با پارامترهای درون جدول برای پیداکردن وزن‌های مناسب در شبکه استفاده می‌کنیم. همچنین تابع هزینه binary_crossentropy می‌باشد. تعداد epochها و batchsize نیز در جدول۱ معین شده‌است. همچنین از معیار binaryaccuracy برای مقایسه عملکرد شبکه‌ها استفاده می‌کنیم.

ابتدا مطابق هایپرپارامترها در جدول۱، با فریز کردن مدل VGG16، یادگیری را انجام می‌دهیم تا بردار وزن‌های قسمت fully connected بدست آید. سپس لایه آخر کانولوشن VGG16 را unfreeze کرده و مجدد فرایند یادگیری را انجام می‌دهیم تا شبکه fine-tune شود.

مدل ResNet50:

از مدل ResNet50 در کتابخانه keras استفاده می‌کنیم. قسمت fully connected این شبکه را حذف و از باقی مانده شبکه (بلوک‌های residual) برای استخراج ویژگی‌ها استفاده می‌کنیم. خروجی شبکه cnn را با استفاده از لایه flatten به صورت بردار درمی‌آوریم. سپس از دو لایه fully connected با ۱۵ نورون و تابع فعال‌ساز relu استفاده می‌کنیم. بعد از اعمال هر لایه fully connected، یک لایه dropout و یک لایه batch normalizer به کار می‌بریم. در آخر از یک لایه با یک نورون با تابع فعال‌ساز sigmoid استفاده می‌کنیم که خروجی آن به‌صورت احتمال تعلق به دسته سگ تعبیر می‌شود. از بهینه‌ساز sgd با پارامترهای درون جدول برای پیداکردن وزن‌های مناسب در شبکه استفاده می‌کنیم. همچنین تابع هزینه binary_crossentropy می‌باشد. تعداد epochها و batchsize نیز در جدول۱ معین شده‌است. همچنین از معیار binaryaccuracy برای مقایسه عملکرد شبکه‌ها استفاده می‌کنیم.

ابتدا مطابق هایپرپارامترها در جدول۱، با فریز کردن مدل ResNet50، یادگیری را انجام می‌دهیم تا بردار وزن‌های قسمت fully connected بدست آید. سپس بلوک residual آخر ResNet50 را unfreeze کرده و مجدد فرایند یادگیری را انجام می‌دهیم تا شبکه fine-tune شود.

استفاده از augmentation:

با استفاده از ۳ لایه horizontal flip، random zoom و random rotate در keras، ورودی شبکه را augment کرده و خروجی آن‌ها را به شبکه‌های اشاره شده در بالا به عنوان ورودی می‌دهیم.

عدم استفاده از augmentation:

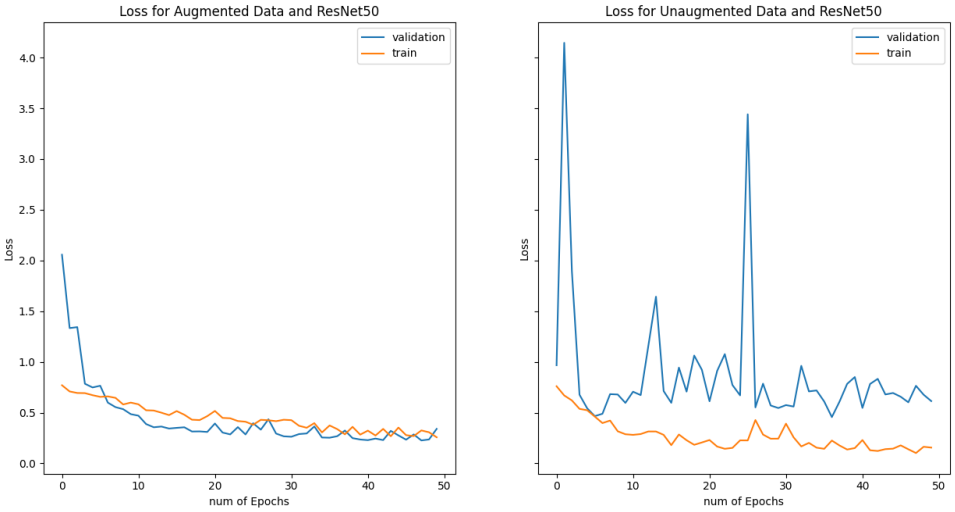
عکس‌ها بعد از مرحله preprocessing و به صورت خام را به صورت مستقیم به شبکه‌های اشاره شده در بالا می‌دهیم.

جدول۱. هایپرپارامترهای شبکه‌ها

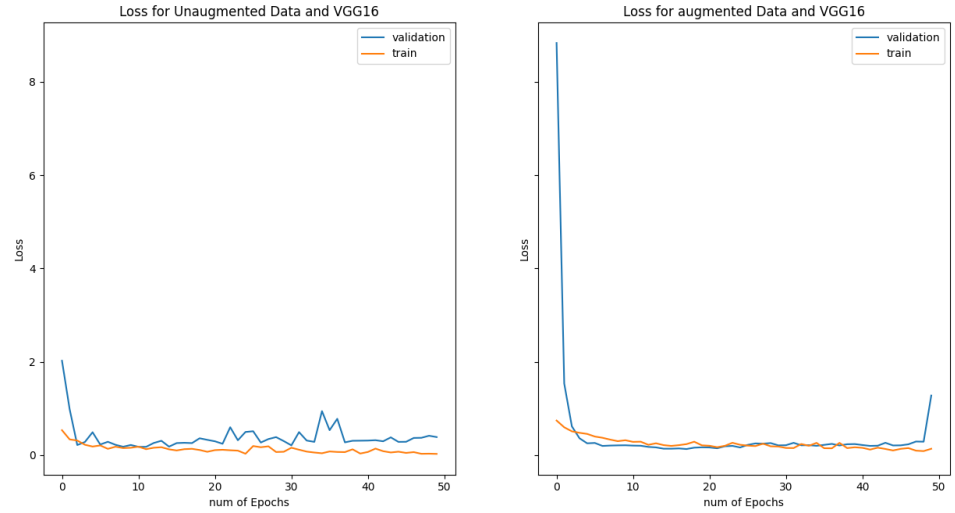
initial learning rate (sgd)	momentum (sgd)	decay (sgd)	batch size	epochs
0.1	0.9	0.002	10	50

۴-۲ نتایج و تحلیل آن.

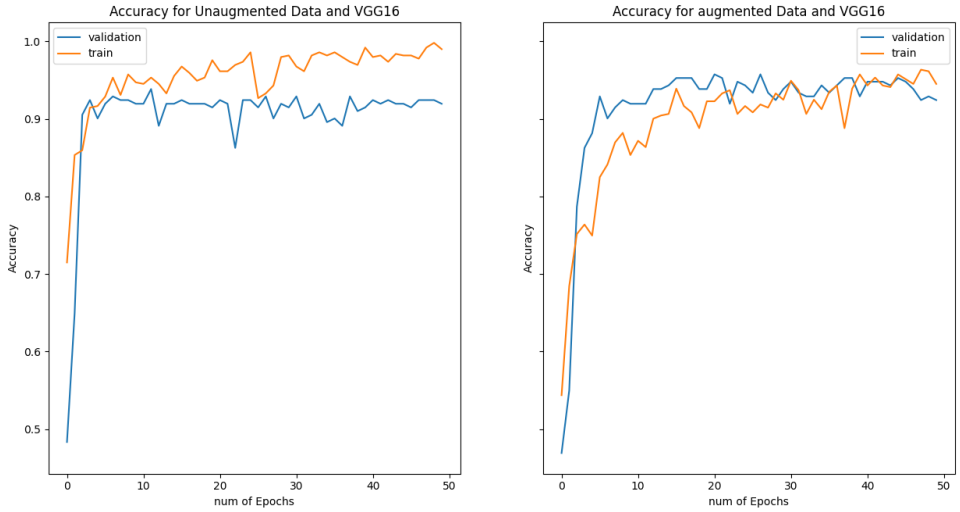
تصویر۱۴. دقت ResNet50



تصویر۱۵. loss شبکه ResNet50



تصویر۱۶. loss شبکه VGG16



تصویر۱۷. دقت VGG16

جدول۲. دقت شبکه‌ها

Model	VGG16 with Augmentation	VGG16 without Augmentation	ResNet50 with Augmentation	ResNet50 without Augmentation
Training acc	٪95	٪99	٪89	٪93
Validation acc	٪92	٪92	٪86	٪83
Test acc	٪94	٪95	٪83	٪75

همانطور که از نتایج درون جدول پیداست، دقت هر دو شبکه روی داده‌های test و validation، پس از augment کردن داده‌ها، اغلب افزایش می‌یابد. علت آن این است که با تعداد پارامتر مشخص، داده‌های بیشتر باعث افزایش قدرت تعمیم شبکه می‌شوند و مانع از overfit شدن شبکه می‌شوند (همانطور که در تصاویر ۲ و ۳ پیداست، شبکه با داده‌های augment شده، overfit نمی‌کند ولی شبکه با داده‌های عادی overfit می‌کند).

برخلاف مقاله، در این پیاده‌سازی، در تمام موارد، vgg16 از دقت بهتری برخوردار است. زیرا سخت‌افزار بکار رفته نسبت به مقاله، پیشرفته‌تر می‌باشد و به ازای تعداد epoch مساوی از train کردن دو شبکه، چون تعداد پارامترهای resnet50 بیشتر است، نتیجه ممکن است ضعیف‌تر باشد. این شهود از نمودارها

نیز بدست میاید. زیرا که در نمودار مربوط به resnet50، شیب دقت train و validation، هنگام متوقف شدن، مثبت است و شبکه همچنان ظرفیت پیشرفت دارد.

.