به نام خدا

دانشگاه تهران

پردیس دانشکده‌های فنی

دانشکده برق و کامپیوتر

درس تحلیل و طراحی شبکه های عصبی عمیق

تمرین شماره ۱

نام و نام خانوادگی و شماره دانشجویی اعضای گروه :

ماجده رضائی (۸۱۰۶۰۱۰۷۳)

محمد اخلاقی ۸۱۰۶۰۱۰۲۷

آبان ماه ۱۴۰۲

فهرست

مقدمه ۳

سوال اول : شاخص‌های هندسی ۳

الف)آموزش شبکه ALexnet با استفاده از MNIST ..... **Error! Bookmark not defined.**

ب) شاخص‌های هندسی SI ..... **Error! Bookmark not defined.**

ج) یافتن شاخص‌های هندسی برای لایه‌ی آخر(قبل از طبقه‌بند) ..... **Error! Bookmark not defined.**

د) ..... **Error! Bookmark not defined.**

و) SMI & Cross SMI برای داده‌ی fetch_california_housing ..... **Error! Bookmark not defined.**

ه)LDI & Reletive density برای داده‌گان mnist ..... **Error! Bookmark not defined.**

سوال دوم : FEATURE SELECTION ۸

الف) FEATURE SELECTION ۹

ب)آموزش لایه‌های طبقه‌بند مدل ALEXNET با ویژگیهای انتخاب شده ۱۱

سوال سوم : قویتر کردن مجموعه دادهها و ارزیابی دادهها ۱۳

الف) آموزش مدل با پنج روش تقویت داده ۱۳

ب) محاسبهی Cross SI ۲۱

مقدمه

در حل این تکلیف، از معماری شبکه‌ی عصبی AlexNet استفاده می‌شود.

همچنین دیتاست A ذکر شده در تکلیف، دیتاست Mnist و دیتاست B آن california housing است.

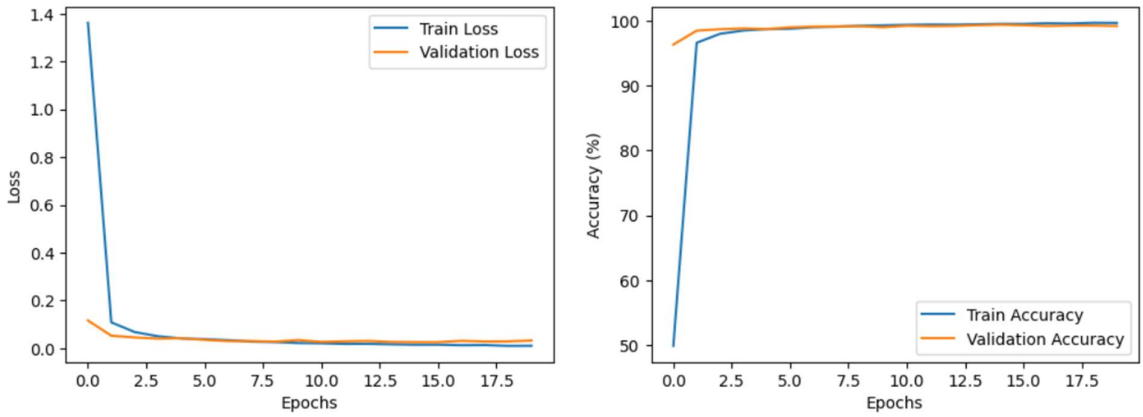
روش‌های Augmentation سوال سوم نیز، در گزارش همان سوال، ذکر شده‌اند.

سوال اول : شاخص‌های هندسی

الف)آموزش شبکه ALEXNET با استفاده از MNIST

در ابتدا معماری شبکه Alexnet پیاده سازی شد. برای این کار از معماری موجود در صفحه‌ای که در گیتهاب کاری مشابه انجام داده بود استفاده شد. لینک صفحه

سپس برای ایجاد دیتاست، مجموعه داده MNIST را از torchvision دریافت شد. با استفاده از تابع random_split ، مجموعه داده آموزش به دو بخش تقسیم شد. ۸۰٪ از داده‌ها را به عنوان مجموعه آموزش و ۲۰٪ دیگر به عنوان مجموعه اعتبارسنجی مشخص می‌شود. بنابراین، در کل در این مرحله ۴۸۰۰۰ داده‌ی آموزشی، ۱۲۰۰۰ داده‌ی ارزیابی و ۱۰۰۰۰ داده‌ی تست داریم. سپس برای هر سه دیتاست آموزش، آزمایش و اعتبارسنجی DataLoader به صورت مجزا ساخته شده و سایز batch ۶۴ و shuffle=True قرارداده شد. برای اینکه در صورت وجود GPU بتوانیم از آن استفاده کنیم device را torch.device("cuda" if torch.cuda.is_available() else "cpu") قرار گرفت و مدل را به آن انتقال یافت. تابع هزینه را CrossEntropyLoss و تابع بهینه‌ساز را SGD با نرخ یادگیری ۰,۰۱ و مومنتوم ۰,۹ قرار گرفت. حال این مدل با دیتاست توضیح داده شده را در یک حلقه‌ی for در ۲۰ گام آموزش می‌دهیم. برای این کار در ابتدا متغیرهایی مانند train_accuracy ایجاد می‌شوند تا دقت و خطاهای آموزش و ارزیابی را در هر دوره آموزش ذخیره کنند و در نهایت این مقادیر برای رسم نمودار استفاده میشوند. در آخر نمودار هزینه و دقت نمایش داده شد. در شکل زیر این نمودارها مشاهده می‌شود.



شکل ۱: نمودارهای دقت و هزینه در حین آموزش شبکه

سپس شبکه ذخیره شد و دقت شبکه بر داده های تست هم محاسبه شد. مقدار دقت و هزینه نهایی به شرح زیر است.

Test Loss: 0.0272 Test Accuracy: 99.31%

ب) شاخص‌های هندسی SI

در این قسمت برای استفاده از کتابخانه "SeprationIndex" داده‌های MNIST با تبدیل خطی به صورت بردار در آمدند. تابع "FlattenTransform" به منظور این تبدیل در کد تعریف شده است.

شاخص‌ها در این بخش برای دو دسته‌ی داده به دست آمد. دسته‌ی اول داده های قسمتی از کل داده‌های آموزشی و دسته‌ی دوم داده های تست. این کار به ما کمک میکند تا مشابهت داده‌های آموزشی و تست را در توضیع هندسی بررسی کنیم.

برای انتخاب بخشی از داده‌های آموزشی ۱۰۰۰ داده از هر کلاس انتخاب شد و سپس با دستور ConcatDataset دیتاستی ۱۰۰۰۰ تایی از داده‌های آموزش ایجاد شد. سپس مجموعه داده تصاویر در یک تانسور پایتورچ (torch.Tensor) ذخیره می‌شود. این کار با استفاده از تابع torch.stack صورت می‌گیرد. در مرحله دوم، برچسب‌های مربوط به تصاویر آموزشی/آزمایشی استخراج و در یک تانسور پایتورچ دیگر ذخیره می‌شود. ابتدا برچسب‌ها به صورت یک لیست پایتونی استخراج و سپس با استفاده از تابع torch.tensor، آنها به یک تانسور پایتورچ تبدیل می‌گردد.

در نهایت به عنوان ورودی به کتابخانه‌ی Kalhor_SeparationIndex، تصاویر و برچسب‌های آموزشی را می‌دهیم و در "instance_concatenated" ذخیره شد. سپس شاخص SI به دست آمد. SI مرتبه بالا مرتبه دو در نظر گرفته شد و برای SI مرتبه بالای سافت مرتبه ۳ در نظر گرفته شد. در نهایت برای داده های تست نیز مشابه داده های آموزشی مقادیر SI به دست آمد. در جدول زیر مقادیر به دست آمده قرار گرفته است.

جدول ۱: مقادیر SI برای داده های MNIST

شاخص	۱۰۰۰۰ داده آموزشی	داده های تست
SI	0.952	0.9558
High order SI(2)	0.9136	0.9188
High order soft SI(3)	0.9361333	0.9410333
Center-Based SI	0.8038	0.8229
Anti SI(2)	0.0263	0.0235

ج) یافتن شاخص‌های هندسی برای لایه‌ی آخر(قبل از طبقه‌بند)

در این بخش ابتدا شبکه ذخیره شده در قسمت ۱ بازخوانی شد. مانند قسمت ب قسمتی از داده‌های آموزشی جدا شد. برای مقایسه بهتر ابتدا تلاش بر این بود که ۱۰۰۰۰ داده آموزشی بخش قبل استفاده شود اما محدودیت ram در این بخش اجازه نمیداد. از هر کلاس ۵۰۰ داده و در مجموع ۵۰۰۰ داده جدا شد. سپس مدل در حالت اولیوشن قرار گرفت و برای استخراج فیچرهای لایه‌ی آخر تابع hook تعریف شد و تابع register_forward_hook استفاده شد. فیچرها خطی سازی شده و به همراه لیبل‌هایی که از قبل در تنسورlabels ذخیره شده بودند به Kalhor_SeparationIndex وارد شدند. در نهایت مقادیر زیر برای شاخص‌های SI به دست آمد:

جدول ۲: مقادیر شاخص‌های SI برای لایه آخر مدل

شاخص	۵۰۰۰ داده آموزشی
SI	0.9794
High order SI(2)	0.9642
High order soft SI(3)	0.9746
Center-Based SI	0.9518
Anti SI(2)	0.0106

مشاهده میشود که تمامی شاخص‌ها بهبود یافته اند.(در صورت انتخاب ۱۰۰۰۰ داده شاخص ها اندکی بهتر نیز می‌شدند)

د) CROSS SI برای داده MNIST

از همان Instace که در بخش ب تعریف شد یعنی با داده های جدا شده از آموزش استفاده شد. داده های تست به عنوان ورودی دیگر تابع اضافه شد و Cross SI محاسبه شد. نتایج زیر به دست آمد.

Cross Separation Index for Classes: [0.99081635 0.99559474 0.9302326 0.9306931 0.9327902 0.9450673
0.9791232 0.9357977 0.8880904 0.9326065]
Cross Separation Index: 0.9466

نتایج به دست آمده نشان میدهد توزیع هندسی دو دسته داده شباهت بسیاری دارد زیرا مقدار Cross SI نزدیک به یک است. از Cross SI کلاسها نیز میتوان فهمید که اعداد یک و دو ۶ بیشترین جدایش را از بقیه‌ی کلاس‌ها دارند.

و) SMI & CROSS SMI برای داده‌ی FETCH_CALIFORNIA_HOUSING

در این بخش کتابخانه "SmoothnessIndex" خوانده شد و داده‌های کالیفرنیا هازینگ از کتابخانه مربوطه فراخوانی شد.

مانند بخش های قبل divice تعریف شد. ابتدا دادگان و لیبل ها در تنسورها قرار گرفت و مانند مثال موجود در گیتهاب به Kalhor_SmoothnessIndex داده شد. و در ادامه SMI به دست آمد.

Linear Smoothness Index for the California housing dataset is: 0.7351261

برای به دست آوردن Cross SMI داده ها را به دو بخش train و test تقسیم کردیم. به کمک train_test_split ۸۰ درصد داده ها آموزشی و ۲۰ درصد تست انتخاب شدند. سپس دادگان و لیبل ها در تنسورها قرار گرفت و مانند مثال موجود در گیتهاب به Kalhor_SmoothnessIndex داده شد. در نهایت به کمک تابع cross_smi_linear مقدار Cross SMI به دست آمد.

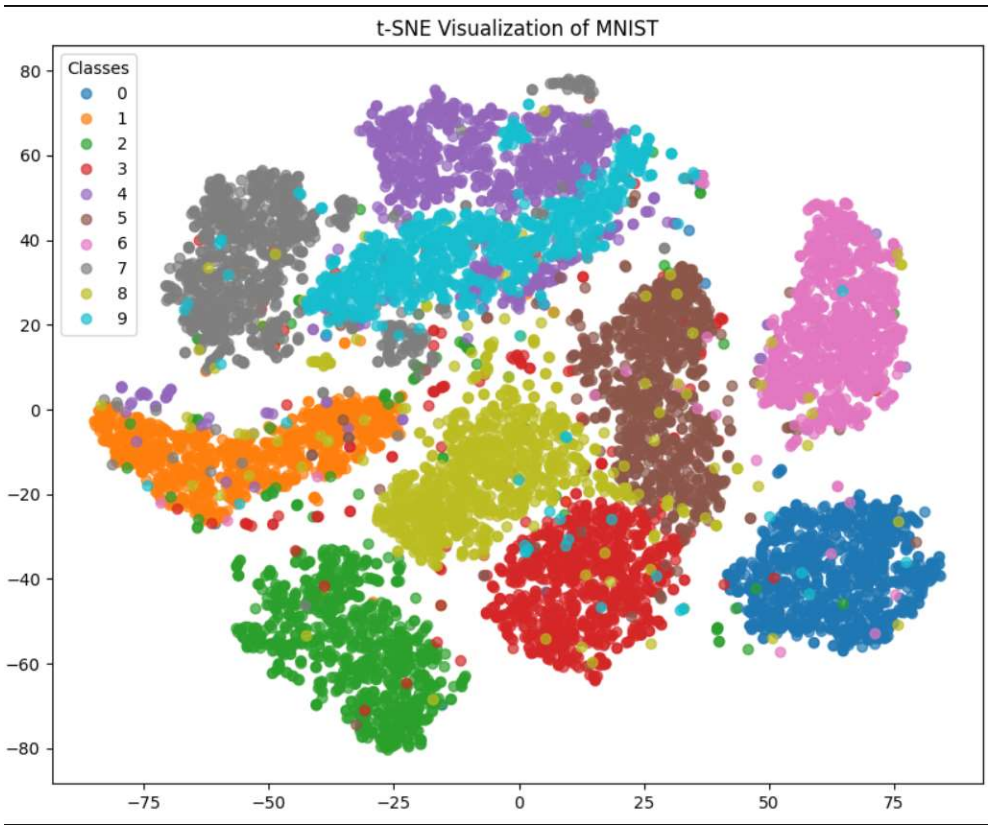
Cross Smoothness Index for the California housing dataset is: 0.7328244

ه)LDI & RELETIVE DENSITY برای داده‌گان MNIST

فراخوانی کتابخانه LDI انجام شد و قسمتی از دادگان MNIST مانند بخش های قبل جداسازی شد و دادگان به شکل تنسور در data و labels قرار گرفت. مانند مثال موجود در گیتهاب kmeans_repeat=20 و حداکثر کلاستر نیز ۲ برابر کلاسها یعنی ۲۰ قرار گرفت. دادگان به Kalhor_LinearDensityIndex داده شد و تعداد کلاستر ها برای دادگان به دست آمد. اما متاسفانه تابع تنها ۱ کلاستر پیدا کرد. این مسئله میتواند به دلیل عدم خوشه خوشه بودن باشد.

Number of clusters is: 1 sum of linear density is: 1893.3639

به دلیل این مشکل در داده ها به کمک روش t-SNE داده ها را دو بعدی کرده و این روش را برای داده‌های تغییر شکل یافته انجام دادیم. برای این کار از ماژول TSNE کتابخانه sklearn.manifold استفاده شد. حال که داده ها دو بعدی هستند میتوان آنها را به شکل زیر میتوان رسم کرد.



شکل ۲: شمای دادگان MNIST پس از کاهش بعد

در این حالت با قرار دادن n_max_clusters = 80 و دادن دادگان جدید به Kalhor_LinearDensityIndex

۲۰ خوشه به دست آمد و مجموع leanier density به دست آمد.

Number of clusters is: 20

sum of linear density is: 146.00197

در ادامه در بخش Reletive density کتابخانه mmt_5data_scoring فراخوانی شد. داده های بخش قبل خطی سازی شد و به کمک تابع module_data_scoring_unsupervised داده شد. و مقادیر امتیاز در mnist_scores ذخیره شد. این امتیازات بین ۰-۱ برای هر داده در این تنسور قرار دارد. بخشی از آن به عنوان نمونه در زیر نوشته شده است. همچنین در این قسمت تعداد خوشه ۴۸ یافت شد.

mnist_scores=[0.4052, 0.4705, 0.8325, ..., 0.5337, 0.1717, 0.7875]

the predicted number of clusters is: 48

سوال دوم : FEATURE SELECTION

در این سوال هدف استفاده از شاخص پیچیدگی SI برای انتخاب ویژگی و کاهش بعد در مدل از پیش آموزش داده سوال یک است.

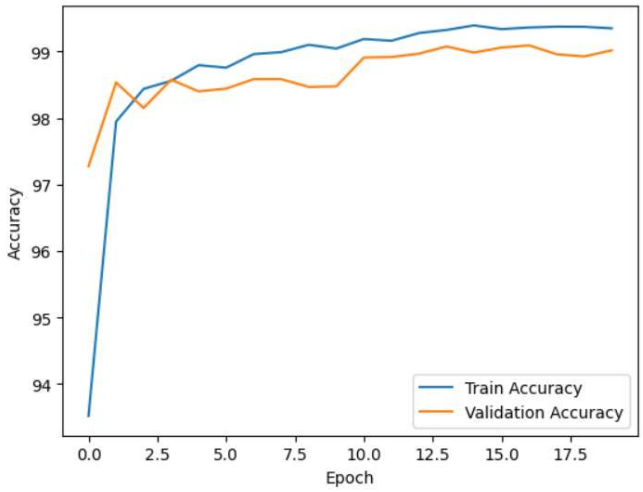
در این قسمت می‌خواهیم مجموعه داده A را به عنوان ورودی به شبکه از پیش از آموزش داده شده سوال اول بدهیم و سپس FEATURE SELECTION انجام دهیم.

قبل از بررسی روند حل این سوال و نتایج آن، به موضوعی می‌پردازیم.

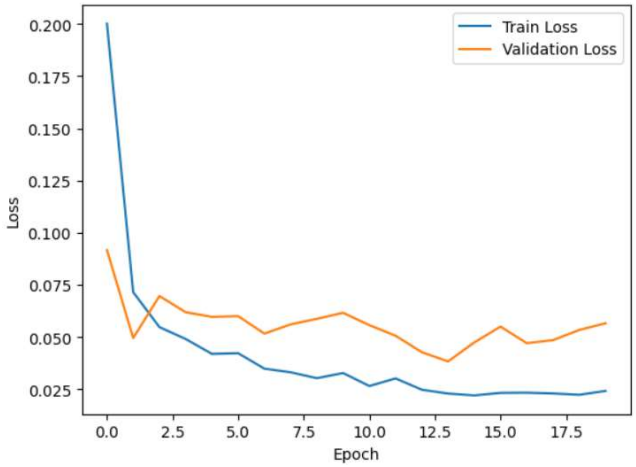
با توجه به اینکه forward_feature_ranking_si حجم محاسباتی بالایی دارد و سیستم مورد استفاده ما (google colab) نیز دارای محدودیت حافظه RAM , GPU است. در صورت استفاده از مدل سوال اول که تعداد ویژگی‌های زیادی تولید می‌کند(۴۶۰۸ ویژگی)، فقط قادر به استفاده از حداکثر ۱۵۰ داده‌ی آموزشی می‌شدیم. پس برای اینکه بتوانیم از دیتاست با تعداد داده‌ی بیشتری استفاده کنیم، مدل AlexNet سوال اول را با تعداد لایه‌های pooling بیشتری (بعد از هر لایه‌ی کانولوشن یک لایه‌ی poolingg با kernel size برابر ۲) تعریف می‌کنیم و دوباره آن را آموزش می‌دهیم. این تغییر باعث می‌شود سایز ورودی اولین لایه‌ی طبقه‌بند که نمایانگر تعداد ویژگی‌ها است نیز تغییر کند.

با این تغییر در شبکه تعداد ویژگی تولیدی آن به ۲۵۹۲ ویژگی می‌رسد.

نتایج حاصل از آموزش و ارزیابی این مدل با دیتاست Mnist در ۲۰ گام و آزمایش آن در ادامه آمده است:



شکل۳ - دقت آموزش و ارزیابی مدل با افزودن لایه‌های pooling در حین آموزش مدل



شکل ۴- خطای آموزش و ارزیابی مدل با افزودن لایه‌های **pooling** در حین آموزش مدل

دقت نهایی شبکه (مدل (مدل AlexNet با افزودن لایه‌های pooling)بر روی داده های تست: ٪۹۹,۲۳

حال این مدل را با فرمت pth. برای استفاده در مراحل بعدی، ذخیره می‌کنیم.

الف) FEATURE SELECTION

برای انجام این سوال در مرحله‌ی اول کتابخانه‌های لازم و درایو (برای بارگیری مدل(۲) ذخیره شده) را فراخوانی می‌کنیم.

در گام بعد دستگاه محاسباتی مورد استفاده در سوال را **torch.device("cuda:0" if torch.cuda.is_available() else "cpu")** قرار می‌دهیم تا در صورت وجود GPU از آن و در غیر این صورت از حافظه CPU استفاده کند.

سپس کلاس **Kalhor_SeparationIndex** را می‌سازیم که شامل تابع **forward_feature_ranking_si** است.

برای استفاده از مدل پیش آموزش دیده، مدل را مجدد تعریف می‌کنیم و به وسیله‌ی **model.load_state_dict(model_checkpoint)** وزن‌های آموزش دیده‌ی آن را استفاده می‌کنیم.

حال نیاز ا ست دیتا ست **mnist** را برای ورود به شبکه و تولید ویژگی‌ها از **torchvision.datasets** بارگیری کنیم. هنگام این کار از تبدیل **transforms.ToTensor** استفاده می‌کنیم تا دادگان به فرمت تنسور ذخیره شوند.

همانطور که گفته شد با توجه به حجم بالای محاسباتی **feature raniking** و محدودیت‌های سیستم مورد استفاده توانایی استفاده از تمامی دادگان را نداریم. برای این کار نیازمند جداسازی بخشی از دادگان هستیم. باتوجه به متوازن بودن دیتاست، از هر کلاس ۱۲۲ داده جدا می‌کنیم و با آن دیتاست جدید شامل ۱۲۲۰ عضو را می‌سازیم.

حال برای جداسازی تصاویر و برچسب‌ها و ذخیره‌ی آنان در متغیرهای مجزا، از تمام نمونه‌های موجود در مجموعه داده تصاویر استخراج و در یک تانسور پایتورچ (**torch.Tensor**) ذخیره می‌شود. این کار با استفاده از تابع **torch.stack** صورت می‌گیرد. در مرحله دوم، برچسب‌های مربوط به تصاویر استخراج و در یک تانسور پایتورچ دیگر ذخیره می‌شود. ابتدا

برچسب‌ها به صورت یک لیست پایتونی استخراج و سپس با استفاده از تابع torch.tensor، آنها به یک تانسور پایتورچ تبدیل می‌گردد. سپس با استفاده از تابع view، ابعاد تانسور برچسب‌ها از یک بعد به دو بعد تبدیل می‌شود. (torch.Size([1220])) به (torch.Size([1220, 1]))

در این مرحله می‌خواهیم ویژگی‌های قبل از لایه‌های کاملاً متصل (fully connected)، که شبکه از پیش آموزش داده شده از دیتاست ایجاد شده، تولید می‌کند را جداسازی کنیم.

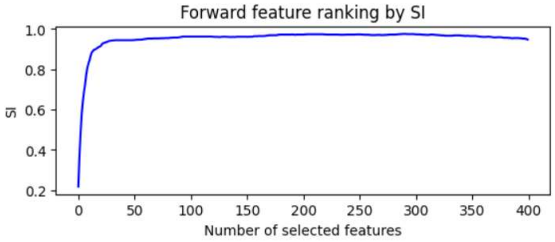
برای این کار ابتدا متغیر features را تعریف می‌کنیم و مقدار آن را None قرار می‌دهیم. این متغیر برای ذخیره کردن ویژگی‌های استخراج شده استفاده می‌شود. سپس یک تابع به نام hook تعریف می‌کنیم که سه ورودی module، input و output را دریافت می‌کند. این تابع، output را در متغیر features ذخیره می‌کند.

سپس از register_forward_hook به صورت model.feature[-1].register_forward_hook(hook) استفاده می‌کنیم. تابع register_forward_hook در PyTorch به ما امکان می‌دهد تا یک پیش‌فرض (forward hook) را به یک لایه از یک ماژول مشخص اضافه کنیم. با استفاده از register_forward_hook می‌توانیم یک تابع را به یک لایه از شبکه عصبی متصل کنیم. هر زمان که داده از لایه مورد نظر عبور می‌کند، تابع مربوطه فراخوانی می‌شود و قادر خواهیم بود ویژگی‌های مورد نظر را در هر مرحله از فرآیند پیش‌روی ثبت کنیم یا تغییر دهیم.

در اینجا، تابع hook به لایه‌ی آخر قسمت اول از مدل(feature) اضافه شده است. هنگامی که داده از این لایه عبور می‌کند، تابع hook فراخوانی می‌شود و ویژگی‌های output لایه را در متغیر features ذخیره می‌کند.

سپس features.view(features.size(0),-1) را بر روی این ویژگی‌ها اعمال می‌کنیم تا تغییر فرمت ویژگی‌ها انجام شود.(تغییر از شکل [۱۲۲۰،۳۲،۹،۹] به شکل [۱۲۲۰،۲۵۹۲])

با انجام این امور یک متغیر شامل ویژگی‌ها به فرمت (torch.Size([1220, 2592]) می‌رسیم. حال از بین این ۲۵۹۲ ویژگی، ۴۰۰ ویژگی را انتخاب می‌کنیم تا نیاز به پردازش و حجم محاسباتی کمتر شود و بتوان از ظرفیت google colab استفاده کرد. حال در این مرحله به عنوان ورودی به Kalhor_SeparationIndex، ویژگی‌های جداسازی شده و برچسب‌ها را می‌دهیم و از آن forward_feature_ranking_si را فراخوانی می‌کنیم. که در نتیجه به ما رنک ویژگی‌ها و SI آن‌ها را می‌دهد. نمودار SI ویژگی‌ها بر حسب تعداد آن‌ها در شکل زیر آمده است. در ابتدا با افزایش ویژگی-ها SI افزایش قابل توجهی داشته است. اما پس از آن افزایش SI به شدت آهسته و کم شده است و این افزایش بسیار اندک تا تعداد ۲۸۸ ویژگی ادامه داشته و پس از آن ثابت شده و حتی مقداری کاهش داشته است.



شکل ۵- نمودار SI ویژگی‌های مدل AlexNet با لایه‌های pooling بر حسب تعداد آن‌ها

ب)آموزش لایه‌های طبقه‌بند مدل ALEXNET با ویژگی‌های انتخاب شده

در این بخش قصد داریم طبقه بند را روی ویژگی‌هایی که از قسمت (الف) بدست آوردیم اعمال کنیم و طبقه بند را آموزش دهیم.

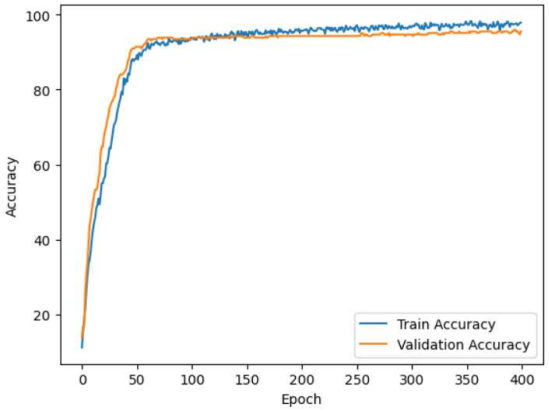
برای آموزش این مدل دوم(مدل طبقه بند) در ابتدا نیاز است دیتاست آن را ایجاد کنیم. با توجه به اینکه قصد داریم فقط طبقه بند مدل را آموزش دهیم، به عنوان ورودی، از ویژگی‌های تولید و جداسازی شده برای هر داده در مرحله (الف) و به عنوان خروجی، برچسب‌های دادگان را انتخاب و به وسیله‌ی TensorDataset از کتابخانه‌ی torch.utils.data

از آن‌ها یک دیتاست می‌سازیم و سپس به کمک random_split از همین کتابخانه، ۸۰ درصد دادگان را برای آموزش و ۲۰ درصد آن‌ها را برای ارزیابی مدل در حین آموزش، به صورت تصادفی، جداسازی و در نهایت به وسیله‌ی DataLoader همچنان از همان کتابخانه، از دیتاست‌های ایجاد شده، دیتالور با batch_size = 64 می‌سازیم.

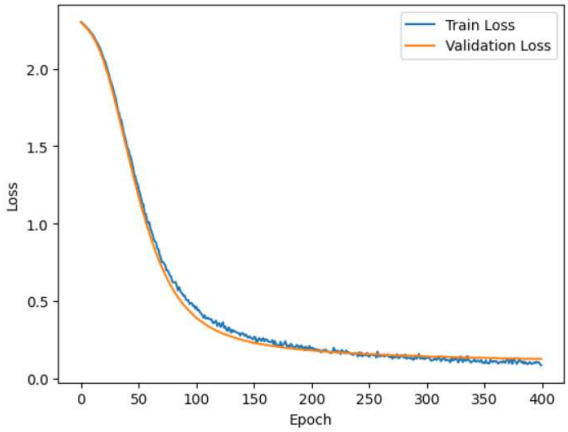
پس از ایجاد دیتالودرهای مد نظر، باید مدل را بسازیم و آموزش دهیم. برای این کار قسمت طبقه‌بند مدل را از مدل اول جدا می‌کنیم و ورودی آن را متناسب با تعداد ویژگی‌های انتخاب شده تغییر می‌دهیم. (یعنی لایه‌ی اول این مدل را به صورت (nn.Linear(max_index,2048 می‌نویسیم که max_index در آن نشاندهنده‌ی تعداد ویژگی‌های موثر در آموزش مدل که در قسمت (الف) به دست آوردیم است.)

برای آموزش مدل تابع هزینه را برابر CrossEntropyLoss و تابع بهینه‌ساز را Adam با نرخ یادگیری ۰,۰۰۱ قرار می‌دهیم. حال با نوشتن یک حلقه‌ی for که در آن خروجی‌های شبکه برای ورودی‌ها را پیش‌بینی، تغییر شکل‌های لازم ورودی و خروجی‌های آموزشی را بر روی آن‌ها اعمال، خروجی‌های واقعی را با خروجی‌های مدل با تابع هزینه تعریف شده مقایسه، و در نهایت ضرایب و پارامترها را به روزرسانی می‌شود، مدل را آموزش می‌دهیم. روند ذکر شده در حلقه‌ی for برای دادگان آموزشی را در همان حلقه با غیرفعال‌سازی کردن به روز رسانی ضرایب و پارامترها، برای مجموعه دادگان ارزیابی نیز انجام می‌دهیم تا عملکرد شبکه حین آموزش را بررسی کنیم. مدل را در ۴۰۰ گام آموزش می‌دهیم.

نمودار دقت و خطای آموزشی و ارزیابی در حین آموزش مدل در اشکال زیر آمده است.



شکل۶ - دقت آموزش و ارزیابی مدل طبقه بند



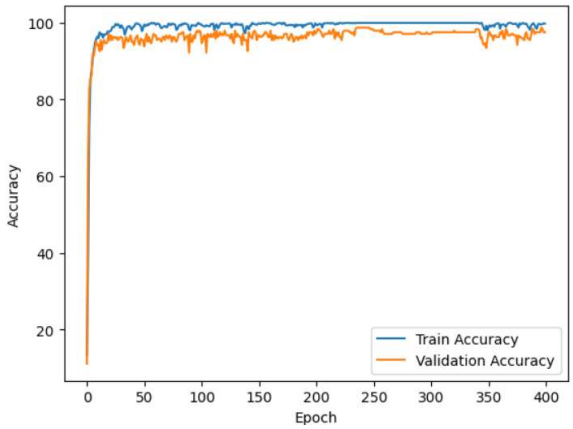
شکل ۷-خطای آموزش و ارزیابی مدل طبقه بند

پس از آموزش مدل باید توسط داده‌های آزمایش، آن را ارزیابی کنیم. برای این کار در ابتدا همانند بخش (الف)، ویژگی-های دادگان آزمایشی را از مدل اصلی از پیش آموزش داده شده، استخراج و با این ویژگی‌ها و برچسب‌های مجموعه داده‌ی آزمایشی، دیتاست و دیتالودر آزمایشی را می‌سازیم و با استفاده از مدل طبقه بند دقت مدل را محاسبه می‌کنیم.

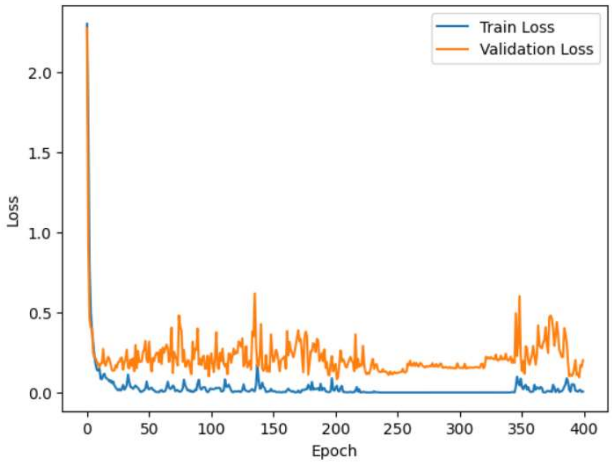
دقت نهایی شبکه بر روی داده های تست: ۹۶,۱۴ %

با توجه به اینکه مدلی که آموزش و نتایج ضرایب آن را ذخیره کرده بودیم، بر روی تمامی دیتاست **Mnist** آموزش دیده شده بود، نمی‌توانیم نتیجه این مدل طیقه بند که با بخشی از دیتاست، آموزش دیده مقایسه کنیم.

پس برای یکسان سازی شرایط مقایسه و دیدن تاثیر استراتژی جداسازی ویژگی‌ها، مدل **AlexNet** با لایه‌های **Pooling** که در همین سوال قبل از بخش (الف) آن را معرفی کردیم را مجددا با دیتاستی هم اندازه‌ی دیتاست مدل طبقه بند، یعنی با ۱۲۲۰ داده و در ۴۰۰ گام آموزش می‌دهیم که نتایج آن در ادامه آمده است.



شکل۸ - دقت آموزش و ارزیابی مدل با افزودن لایه‌های **pooling** با دیتاست کوچک آموزشی و ارزیابی در حین آموزش مدل



شکل ۹- خطای آموزش و ارزیابی مدل با افزودن لایه‌های pooling با دیتاست کوچک آموزشی و ارزیابی در حین آموزش مدل

دقت نهایی شبکه (مدل AlexNet با افزودن لایه‌های pooling با دیتاست کوچک آموزشی و ارزیابی) بر روی داده های تست: ٪۹۶,۷۸

همانطور که از نتایج به دست آمده (از آموزش مدل‌ها با دیتاست کوچک) مشخص است با کاهش تعداد ویژگی‌ها دقت مدل ۹۶,۱۴ درصد و دقت مدل اصلی ۹۶,۷۸ بوده است. با وجود اینکه دقت در حالت کاهش ویژگی‌ها مقداری کاهش داشته است ولی همانطور که از نمودارهای دقت و خطا مشخص است، در حالتی که مدل را فقط با ویژگی‌های موثر آموزش دهیم، نوسان دقت و خطا در حین آموزش کمتر است و روند آموزش پایداری بیشتری دارد و همچنین نمودارهای خطا و دقت ارزیابی و آموزش نیز در این حالت به هم نزدیک‌ترند و تفاوت بسیار ناچیزی دارند، که این نشان می‌دهد مدل در این حالت قدرت تعمیم‌پذیری بهتری دارد. همچنین با این کار، حجم محاسباتی مدل طبقه‌بند را کاهش می‌دهیم که این نیز یک مزیت مهم به حساب می‌آید.

سوال سوم : قوی‌تر کردن مجموعه داده‌ها و ارزیابی داده‌ها

در این سوال با استفاده از کتابخانه‌ی Albumentations داده‌ها را تنوع داده‌ایم و با این کار عملکرد مدل را بهبود می‌دهیم.

الف) آموزش مدل با پنج روش تقویت داده

برای این کار در ابتدا کتابخانه‌های لازم را فراخوانی می‌کنیم. سپس مدل AlexNet که در سوال یک آن را ساخته بودیم را مجددا تعریف می‌کنیم.(جزئیات مربوط به آن در گزارش سوال اول آورده شده است.)

سپس یک کلاس به نام AlbumentationsTransform تعریف می‌کنیم که یک تبدیل Albumentations را می‌گیرد و برای تصاویر اجرا می‌کند.

این کلاس، یکی از تبدیل‌های موجود در کتابخانه Albumentations را به عنوان ورودی دریافت می‌کند و آن را در ویژگی transform خود ذخیره می‌کند. (در این جا ما قصد داریم ۵ نوع از روش‌های تقویت داده را بررسی کنیم. پس در هر مرحله یکی از روش‌ها را در transform = albumentations.Compose() قرار می‌دهیم.)

در تابع __call__ این کلاس، تصویر ورودی به صورت یک آرایه NumPy تبدیل می‌شود و سپس تبدیل Albumentations با استفاده از self.transform(image=img)["image"] روی آن اعمال می‌شود. نتیجه به صورت یک تصویر NumPy است. سپس، تصویر به یک تنسور PyTorch تبدیل می‌شود و در نهایت به عنوان نتیجه تابع بازگردانده می‌شود.

(نحوه استفاده از این کتابخانه با مطالعه و گرفتن راهنمایی از این لینک ایده گرفته شده است:

حال برای ایجاد دیتاست، مجموعه داده MNIST را از torchvision دریافت می‌کنیم (هر دو دیتاست آزمایش و آموزش را دانلود می‌کنیم.) و برای اعمال Albumentations بر روی دادگان، متغیر transform در torchvision.datasets.MNIST را برابر کلاس AlbumentationsTransform(transform) که در مرحله قبل ساختیم قرار می‌دهیم. لازم به ذکر است ما در اینجا کلاس تقویت دادگان را برای هر دو مجموعه آموزش و آزمایش اعمال می‌کنیم. این کار باعث می‌شود علاوه بر اینکه مدل بر روی دادگان متنوع‌تری آموزش ببیند و قوی‌تر شود، باعث می‌شود که مدل بر روی دادگان متنوع‌تری نیز ارزیابی شود و درصد دقتی که از آزمایش آن می‌گیریم قابل قبول‌تر شود. البته می‌توانستیم این تقویت دادگان را فقط بر روی دادگان آموزش اعمال کنیم.

سپس با استفاده از تابع random_split، مجموعه داده آموزش را به دو بخش تقسیم می‌کنیم. ۸۰٪ از داده‌ها را به عنوان مجموعه آموزش و ۲۰٪ دیگر به عنوان مجموعه اعتبارسنجی مشخص می‌شود.

(یعنی در کل در این مرحله ۴۸۰۰۰ داده‌ی آموزشی، ۱۲۰۰۰ داده‌ی ارزیابی و ۱۰۰۰۰ داده‌ی آزمایشی داریم.)

در نهایت برای هر سه دیتاست آموزش، آزمایش و اعتبارسنجی یک DataLoader می‌سازیم و سایز batch را ۶۴ و shuffle=True قرار می‌دهیم.

برای اینکه در صورت وجود GPU بتوانیم از آن استفاده کنیم device را torch.device("cuda" if torch.cuda.is_available() else "cpu") قرار می‌دهیم و مدل را به آن انتقال می‌دهیم.(در زمان آموزش مدل، تصاویر و برچسب‌ها را نیز به آن انتقال می‌دهیم.)

تابع هزینه را CrossEntropyLoss و تابع بهینه‌ساز را Adam با نرخ یادگیری ۰٫۰۰۱ قرار می‌دهیم.

حال این مدل با دیتاست توضیح داده شده را در یک حلقه‌ی for در ۲۰ گام آموزش می‌دهیم. برای این کار در ابتدا متغیرهایی مانند train_accs، val_accs،train_losses و val_losses ایجاد می‌شوند تا دقت و خطاهای آموزش و ارزیابی را در هر دوره آموزش ذخیره کنند.

با قرار دادن model.train، مدل را به حالت آموزش در می‌آوریم و سپس برای هر دسته داده در داده‌های آموزش، ورودی‌ها و برچسب‌ها را به دستگاه محاسباتی(CPU, GPU) انتقال می‌دهیم. ورودی‌ها را از مدل عبور می‌دهیم و خروجی‌ها را محاسبه می‌کنیم و در نهایت تابع خطای تخمین را برای خروجی‌ها و برچسب‌ها به وسیله‌ی تابع هزینه انتخابی، محاسبه می‌کنیم.

در نهایت گرادیان‌ها را برای پارامترهای مدل محاسبه و پارامترها را با استفاده از بهینه‌ساز به‌روز می‌کنیم.

برای محاسبه‌ی دقت مدل در حین آموزش، برچسب‌های پیش‌بینی شده را با برچسب‌های واقعی مقایسه کرده و تعداد پیش‌بینی‌های صحیح را محاسبه می‌کنیم.

در گام بعدی آموزش مدل، برای ارزیابی مدل حین روند آموزش، مدل را به وسیله‌ی model.eval به حالت ارزیابی در می‌آوریم و برای هر دسته داده در داده‌های ارزیابی، ورودی‌ها و برچسب‌ها را به دستگاه محاسباتی انتقال و سپس ورودی‌ها را از مدل عبور می‌دهیم و خروجی‌ها را محاسبه می‌کنیم. سپس تابع خطای تخمین برای خروجی‌ها و برچسب‌ها محاسبه و خطای فعلی به val_loss اضافه می‌شود.

برای محاسبه‌ی دقت مدل در حین ارزیابی در زمان آموزش مدل، برچسب‌های پیش‌بینی شده را با برچسب‌های واقعی مقایسه کرده و تعداد پیش‌بینی‌های صحیح را محاسبه می‌کنیم.

در نهایت خطا و دقت آموزش و ارزیابی‌ای که در طی مراحل فوق یعنی آموزش مدل به دست آورده و در متغیرها ذخیره کرده بودیم را توسط کتابخانه‌ی matplotlib.pyplot رسم می‌کنیم.

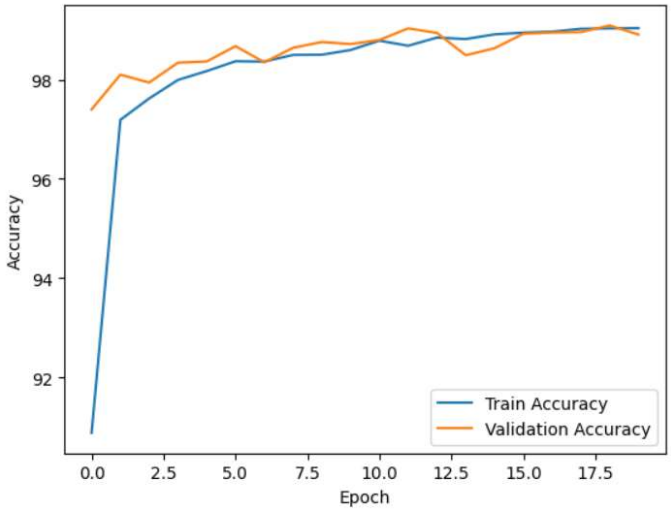
در مرحله‌ی نهایی برای آزمایش مدل آموزش دیده، خروجی داده‌های آزمایش را توسط مدل پیش بینی می‌کنیم و با خروجی‌های واقعی مقایسه می‌کنیم و دقت و خطای نهایی مدل را به دست می‌آوریم.

مراحل توضیح داده شده‌ی فوق را برای پنج روش تقویت داده انجام داده‌ایم که در ادامه نتایج حاصل از آن آمده است:

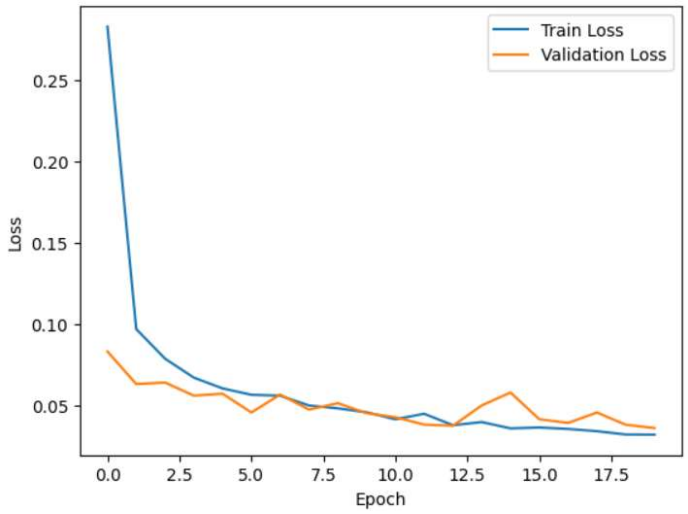
(لازم به ذکر است که برای استفاده از این پنج روش، پارامترهای پیش‌فرض آن را تغییر نداده‌ایم.)

۱- albumentations.augmentations.blur.transforms.Blur:

روش Blur در کتابخانه Albumentations یک تبدیل (transform) است که برای اعمال اثر ماتی (Blur) بر تصاویر استفاده می‌شود. این تبدیل از فیلترهای مات مختلفی برای میزان مات کردن تصویر استفاده می‌کند.



شکل ۱۰- دقت آموزش و ارزیابی مدل با روش تقویت داده Blure

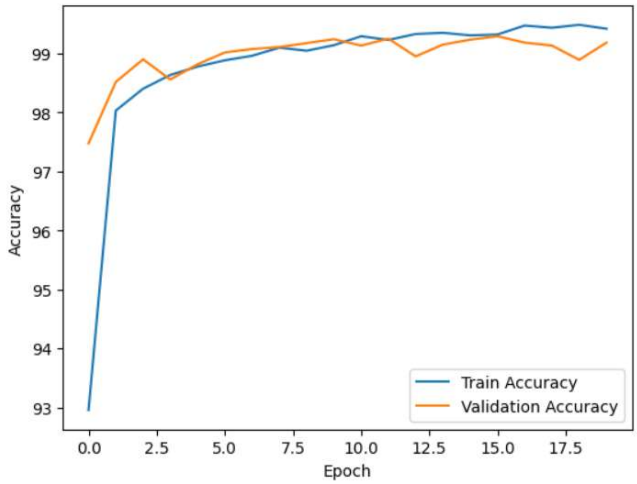


شکل ۱۱-خطای آموزش و ارزیابی مدل با روش تقویت داده Blure

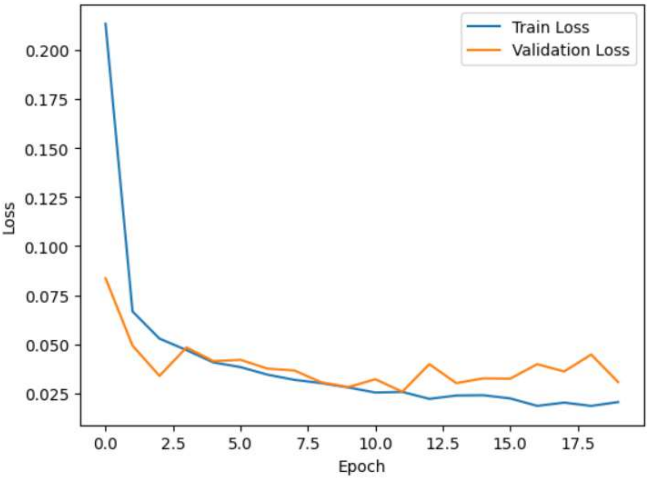
دقت نهایی شبکه بر روی دادگان تست روش تقویت داده Blure: ۹۹,۰۶ %

۲- albumentations.augmentations.transforms.ToRGB:

روش ToRGB در کتابخانه Albumentations یک تبدیل (transform) است که برای تبدیل فضای رنگی تصاویر به حالت RGB استفاده می‌شود. این تبدیل معمولاً بر روی تصاویری اعمال می‌شود که در فضای رنگی دیگری مانند BGR یا Grayscale هستند و می‌خواهیم آن‌ها را به فضای رنگی RGB تبدیل کنیم.



شکل ۱۲ - دقت آموزش و ارزیابی مدل با روش تقویت داده TORGB



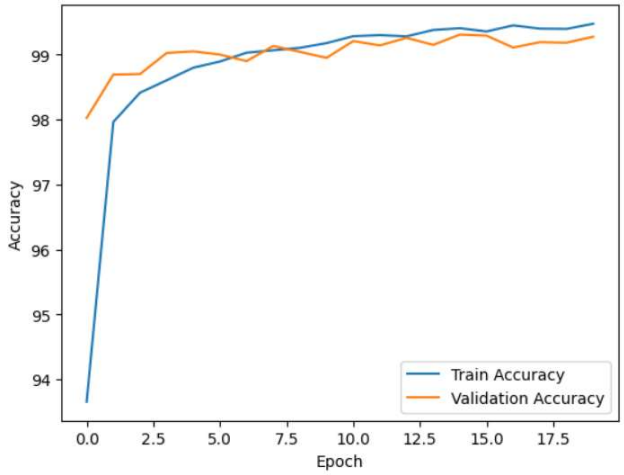
شکل ۱۳- خطای آموزش و ارزیابی مدل با روش تقویت داده TORGB

دقت نهایی شبکه بر روی دادگان تست با روش تقویت داده TORGB : % ۹۹,۳۴

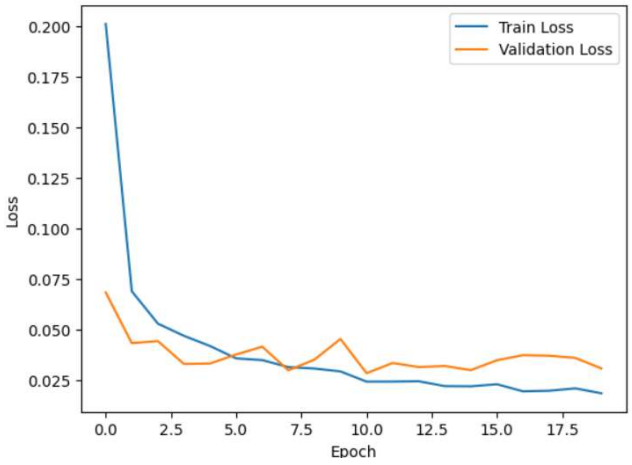
۳- albumentations.augmentations.transforms.GaussNoise:

روش GaussNoise در کتابخانه Albumentations یک تبدیل (transform) است که برای افزودن نویز گوسی به تصاویر استفاده می‌شود. نویز گوسی یک نوع از نویزهای تصادفی است که به طور تصادفی به تصاویر افزوده می‌شود و مقادیر پیکسل‌ها را تغییر می‌دهد.

استفاده از تبدیل GaussNoise می‌تواند به تنوع و تقویت داده‌ها کمک کند. با افزودن نویز گوسی به تصاویر، تغییرات تصادفی در مقادیر پیکسل‌ها ایجاد می‌شود. این تغییرات می‌تواند به مدل کمک کند تا به تصاویر با شرایط متفاوت و سخت‌تر عادت کند و در نتیجه، بهبود عملکرد مدل در شرایط ناهنجار و واقعی را فراهم کند.



شکل۱۴ - دقت آموزش و ارزیابی مدل با روش تقویت داده GaussNoise

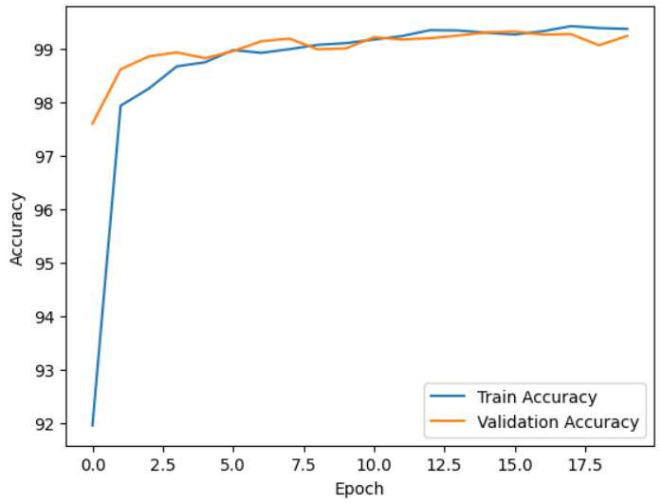


شکل ۱۵- خطای آموزش و ارزیابی مدل با روش تقویت داده GaussNoise

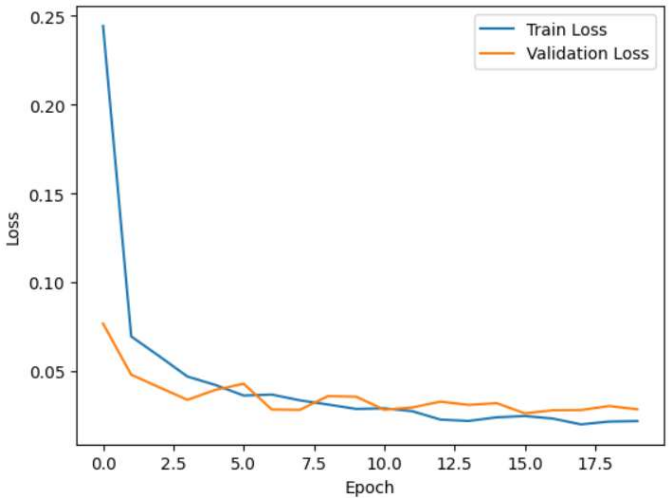
دقت نهایی شبکه بر روی دادگان تست با روش تقویت داده GaussNoise: ۹۹,۳۶ %

۴- albumentations.augmentations.transforms.Sharpen:

روش Sharpen در کتابخانه Albumentations یک تبدیل (transform) است که برای تیز کردن (شارپ کردن) تصاویر استفاده می‌شود. این تبدیل با اعمال فیلترهایی بر روی تصاویر، باعث افزایش وضوح و تمرکز تصویر می‌شود. این می‌تواند به مدل کمک کند تا الگوها و ویژگی‌های مهم تصاویر را بهتر تشخیص دهد و در نتیجه، عملکرد مدل را بهبود بخشد.



شکل ۱۶-دقت آموزش و ارزیابی مدل با روش تقویت داده Sharpen

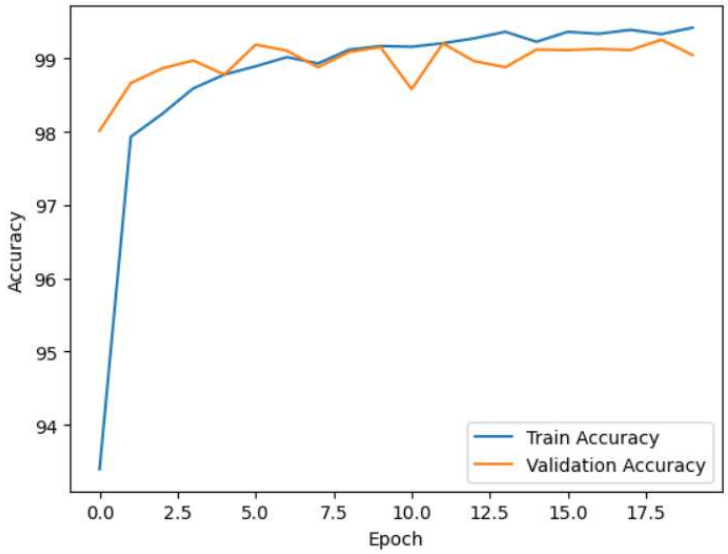


شکل ۱۷- خطای آموزش و ارزیابی مدل با روش تقویت داده Sharpen

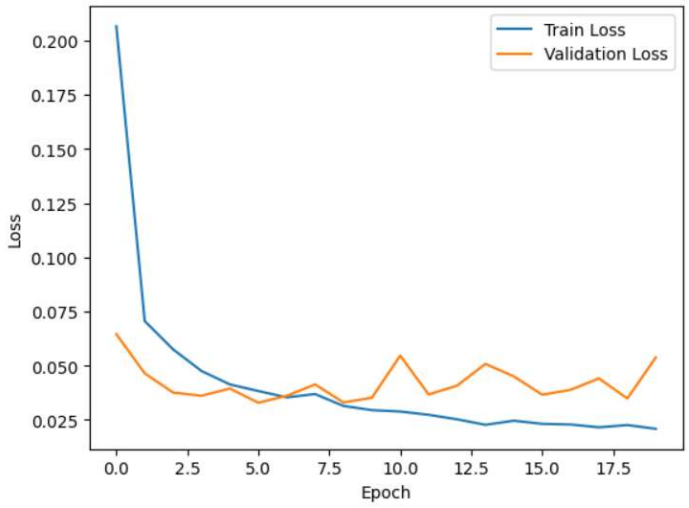
دقت نهایی شبکه بر روی دادگان تست با روش تقویت داده Sharpen: ٪ ۹۹,۳۵

۵- albumentations.augmentations.blur.transforms.ZoomBlur:

روش ZoomBlur در کتابخانه Albumentations یک تبدیل (transform) است که برای اعمال افکت مات و حرکت به تصاویر استفاده می‌شود.



شکل۱۸ - دقت آموزش و ارزیابی مدل با روش تقویت داده ZoomBlur



شکل ۱۹ - خطای آموزش و ارزیابی مدل با روش تقویت داده ZoomBlur

دقت نهایی شبکه بر روی دادگان تست با روش تقویت داده ZoomBlur: ۹۹,۱۸ %

جدول ۳- مقایسه نتایج دقت مدل بر روی دادگان آزمایش با استفاده پنج روش تقویت دادگان

نام روش تقویت دادگان	دقت بر روی دادگان آزمایش(درصد)
Blur	۹۹.۰۶
TORGB	۹۹.۳۴
Gauss Noise	۹۹.۳۶
Sharpen	۹۹.۳۵
Zoom Blur	۹۹.۱۸

همانطور که از نتایج فوق مشخص است، بر روی دیتاست **mnist** و شبکه‌ی **AlexNet**، روش **Gauss Noise** با اختلاف کمی نسبت به روش‌های **Sharpen** و **TORGB** در بین پنج روش امتحان شده، بهترین نتیجه را داشته است و نتیجه آن از دقت همین مدل بدون اعمال تقویت دادگان که در سوال اول آمده، بهتر شده است. البته این بدان معنا نیست که این روش در همه‌ی شبکه‌ها و دیتاست‌ها بهترین عملکرد را دارد. میزان تاثیر این روش‌های تقویت دادگان بر روی نتایج کاملا به دیتاست و شبکه بستگی دارد و چه بسا در برخی نقاط اثر کاملا منفی یا خنثی داشته باشد.)

ب) محاسبه‌ی CROSS SI

در این قسمت قصد داریم مقدار **Cross SI** را برای هر پنج حالت تقویت داده محاسبه کنیم. برای این کار از قسمت **cross_si** از کتابخانه‌ی **Kalhor_SeparationIndex** استفاده می‌کنیم.

در گام اول همانند بخش‌های قبلی دیتاست را فراخوانی می‌کنیم و سپس برای آن‌ها به وسیله‌ی **torch.utils.data.DataLoader** دیتالودر می‌سازیم.

حال قصد داریم برچسب‌ها و تصاویر دیتاست **mnist** را در هر دو دسته‌ی آزمایش و آموزش از هم جدا و در متغیرهای مجزا ذخیره می‌کنیم.

برای این کار در مرحله اول، از تمام نمونه‌های موجود در مجموعه داده آموزش/آزمایش تصاویر استخراج و در یک تانسور پایتورچ (**torch.Tensor**) ذخیره می‌شود. این کار با استفاده از تابع **torch.stack** صورت می‌گیرد. در مرحله دوم، برچسب‌های مربوط به تصاویر آموزشی/آزمایشی استخراج و در یک تانسور پایتورچ دیگر ذخیره می‌شود. ابتدا برچسب‌ها به صورت یک لیست پایتونی استخراج و سپس با استفاده از تابع **torch.tensor**، آنها به یک تانسور پایتورچ تبدیل می‌گردد.

سپس با استفاده از تابع view، ابعاد تانسور برچسب‌ها را از یک بعد به دو بعد (برای مثال برای دادگان آزمایش از (torch.Size([10000]) به (torch.Size([10000, 1]) و ابعاد تصاویر را از ۴ بعد به دو بعد(برای مثال برای دادگان آموزش از (torch.Size([48000, 1, 28, 28]) به (torch.Size([48000, 784]) تغییر می‌دهیم.

در گام بعدی برای استفاده از حافظه‌ی GPU (در صورت وجود) برای انجام محاسبات، دستگاه را برابر torch.device("cuda" if torch.cuda.is_available() else "cpu") قرار می‌دهیم.(در کد کتابخانه‌ی Kalhor_SeparationIndex نیز کد self.data.device = self.device1 را اضافه می‌کنیم که دستگاه در داخل کلاس مشخص شود و بتواند از آن استفاده کند.)

در نهایت به عنوان ورودی به کتاب‌خانه‌ی Kalhor_SeparationIndex، تصاویر و برچسب‌های آموزشی را می‌دهیم و نتیجه را در یک متغیر ذخیره می‌کنیم.

حال به cross_si این متغیر، تصاویر و برچسب‌های آزمایشی را به عنوان ورودی می‌دهیم تا cross_si را محاسبه کند.

این مقدار را برای هر پنج روش تقویت داده محاسبه کرده‌ایم که در جدول زیر نتایج آن آمده است:

جدول ۴- مقایسه نتایج Cross SI بر روی دادگان تقویت شده با استفاده پنج روش تقویت دادگان

نام روش تقویت دادگان	Cross SI
Blur	۰.۹۴۸۴
TORGB	۰.۹۶۶۲
Gauss Noise	۰.۹۶۶۹
Sharpen	۰.۹۶۱۱
Zoom Blur	۰.۹۶۲۶

** لازم به ذکر است با توجه به اینکه روش TORGB حجم دادگان را افزایش می‌دهد، باتوجه به محدودیت-های سخت افزاری امکان استفاده از کل مجموعه دادگان آموزشی را نداشتیم و از بین ۶۰۰۰۰ داده‌ی آموزشی ۵۵۰۰۰ تا را به صورت متوازن جدا کردیم. و سپس از بین این ۵۵۰۰۰ داده، دیتاست آموزشی و ارزیابی ساخته‌ایم. (که البته دیتاست ارزیابی در این بخش سوال استفاده نمی‌شود.اما جهت هماهنگی تعداد دادگان آموزشی با بخش (الف) این تقسیم دادگان به آموزشی و ارزیابی در این قسمت نیز باقی مانده است.). همچنین از بین دادگان ۱۰۰۰۰ داده‌ی آزمایشی نیز ۹۰۰۰ داده را به صورت متوازن جدا کرده‌ایم.

Cross SI شاخص جداسازی دامنه آزمایشی مجموعه داده را بر اساس دامنه آموزشی مجموعه داده اندازه گیری می‌کند و هرچه این مقدار بیشتر باشد بدین معناست که مدل قدرت تعمیم بیشتری دارد.

همـانطور کـه انتظـار مـی‌رفـت بـالاترین Cross SI متعلـق بـه دادگـان تقویـت شـده بـا روش Gauss Noise بـوده‌انـد که در بخش قبلی نیز بیشترین دقت را از خود نشان دادند.