به نام خدا

دانشگاه تهران

دانشکده مهندسی برق و کامپیوتر

هوش مصنوعی قابل اعتماد

تمرین اول

نام و نام خانوادگی	محمدرضا سلیمی
شماره دانشجویی	810102178

فهرست

سؤال اول

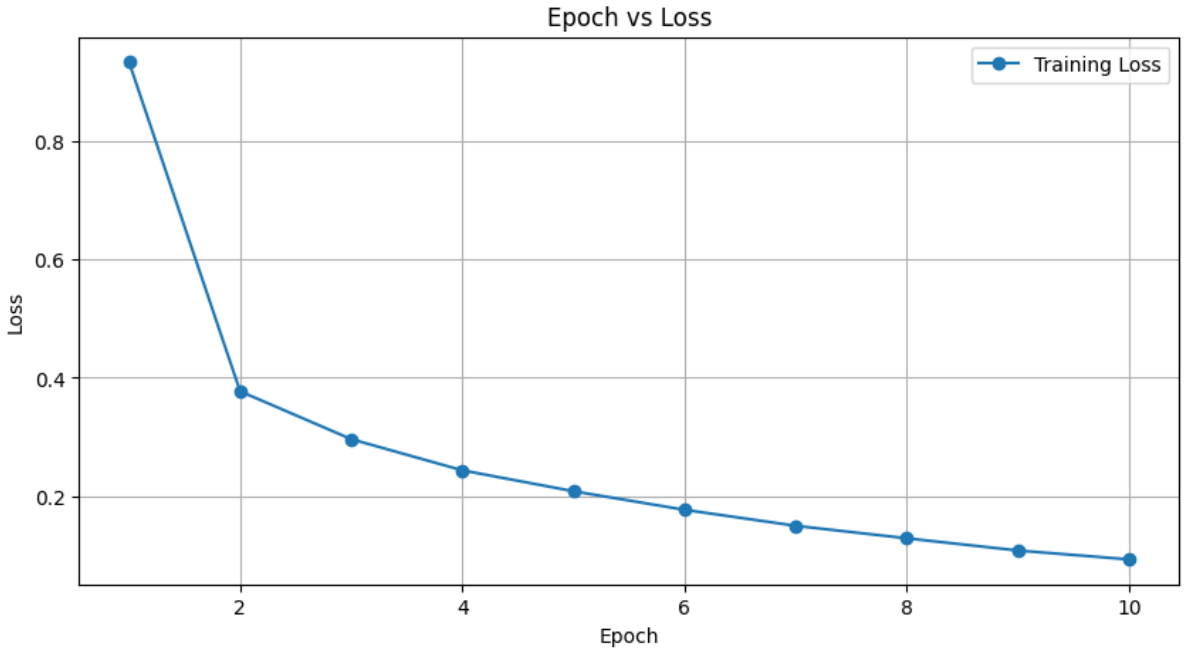
آموزش مدل اولیه

- کدهای این قسمت را در یک فایل jupyter notebook به نام base_resnet قرار داده ام .

در این قسمت یک مدل به صورت دستی برای Resnet18 طراحی کرده ام و و با داده های آموزشی دیتاست SVHN آموزش داده ام و در ادامه مدل را بر روی داده های تست دیتاست SVHN و mnist ارزیابی کرده ام

از آنجا که تصاویر MNIST به صورت خاکستری (یک کانال) هستند تبدیلی را استفاده کرده ام که تصاویر یک کاناله را به سه کانال تبدیل می‌کند. این کار برای همخوانی با مدل ما که برای آموزش از داده های SVHN که سه بعدی است استفاده میکنیم ضروری است .

مدل را با بهینه ساز SGD with momentum در 10 ایپاک آموزش داده ام که نمودار تغییرات خطای آن به این شکل شد .



نمودار تغییرات خطا برای داده های آموزشی مدل پایه رزنت

با توجه به نمودا خطا با گذشت هر دوره آموزشی به طور مستمر کاهش یافته است. این نشان می‌دهد که مدل در حال یادگیری از داده‌های آموزشی است

همچنین دقت مدل روی داده های تست SNHN 92 درصد شد و روی داده های تست MNIST 64 درصد شد

درصد هایی که از داده های تست بدست آمد کاملا منطقی است دقت روی داده های تست SVHN 92 درصد شد چون که ما مدل را روی همین داده ها آموزش داده ایم و روی داده های MNIST 64 درصد شد و با اینکه ما از روش بهبود تعمیم پذیر خاصی استفاده نکردیم ولی دقت نسبتا مناسبی بدست آمد .

MODEL ARCHITECTURE

Batch Normalizationاز طریق کاهش Internal Covariate Shift به بهبود تعمیم‌پذیری در شبکه‌های عصبی کمک می‌کند Internal Covariate Shift .به تغییرات در توزیع ورودی‌های هر لایه در طول آموزش اشاره دارد که این مسئله می‌تواند منجر به کند شدن فرآیند آموزش، نیاز به نرخ‌های یادگیری پایین‌تر، و دشواری در رسیدن به بهینه‌سازی شود Batch Normalization .با نرمال‌سازی ورودی‌های هر لایه در طول آموزش، توزیع ورودی‌ها را ثابت نگه می‌دارد، که این امر می‌تواند به شبکه کمک کند تا سریع‌تر آموزش ببیند و بهتر تعمیم دهد

این روش به ویژه با اجازه دادن به استفاده از نرخ‌های یادگیری بالاتر بدون خطر از دست دادن همگرایی و کاهش نیاز به تنظیم‌های دقیق مقدار اولیه پارامترها، روند آموزش را تسریع می‌بخشد. علاوه بر این، Batch Normalizationبا کاهش وابستگی گرادیان‌ها به مقیاس پارامترها یا مقادیر اولیه آن‌ها، به جریان بهتر گرادیان‌ها در شبکه کمک می‌کند. این امر به نوبه خود امکان استفاده از نرخ‌های یادگیری بیشتر را فراهم می‌آورد و باعث می‌شود آموزش نسبت به مقیاس پارامترها مقاوم‌تر باشد

با تثبیت توزیع ورودی‌های هر لایه و حفظ توانایی شبکه برای نمایش تابع هویت (از طریق پارامترهای قابل یادگیری که در هر لایه اضافه می‌شوند)، Batch Normalizationبه شبکه امکان می‌دهد که بهینه‌سازی را با ثبات بیشتری انجام دهد. این کار به ویژه برای شبکه‌های عمیق که ممکن است در غیر این صورت با مشکلاتی نظیر گرادیان‌های ناپدید شونده یا منفجر شونده مواجه شوند، مفید است.

در نتیجه، Batch Normalization با کاهش Internal Covariate Shift و بهبود جریان گرادیان‌ها، آموزش را تسریع بخشیده و به شبکه‌های عصبی کمک می‌کند تا با دقت بالاتر و در زمان کوتاه‌تری به تعمیم بهتری دست یابند

طبق مقاله Dropout یک تکنیک قدرتمند برای کاهش بیش‌برازش در شبکه‌های عصبی و در نتیجه افزایش توانایی تعمیم‌دهی آن‌ها است. اصل اساسی Dropout ، "رها کردن" یا نادیده گرفتن موقتی زیرمجموعه‌ای از نورون‌ها در شبکه حین آموزش است. این بدان معناست که در هر مرحله آموزش، انتخاب تصادفی از نورون‌ها از محاسبات حذف می‌شوند، که در واقع یک نسخه باریک‌شده از شبکه را ایجاد می‌کند. نورون‌های حذف شده موقتاً در نظر گرفته می‌شوند که در شبکه وجود ندارند. در زمان آزمون، هیچ نورونی حذف نمی‌شود و تمام نورون‌ها در محاسبات شرکت می‌کنند. با این حال،

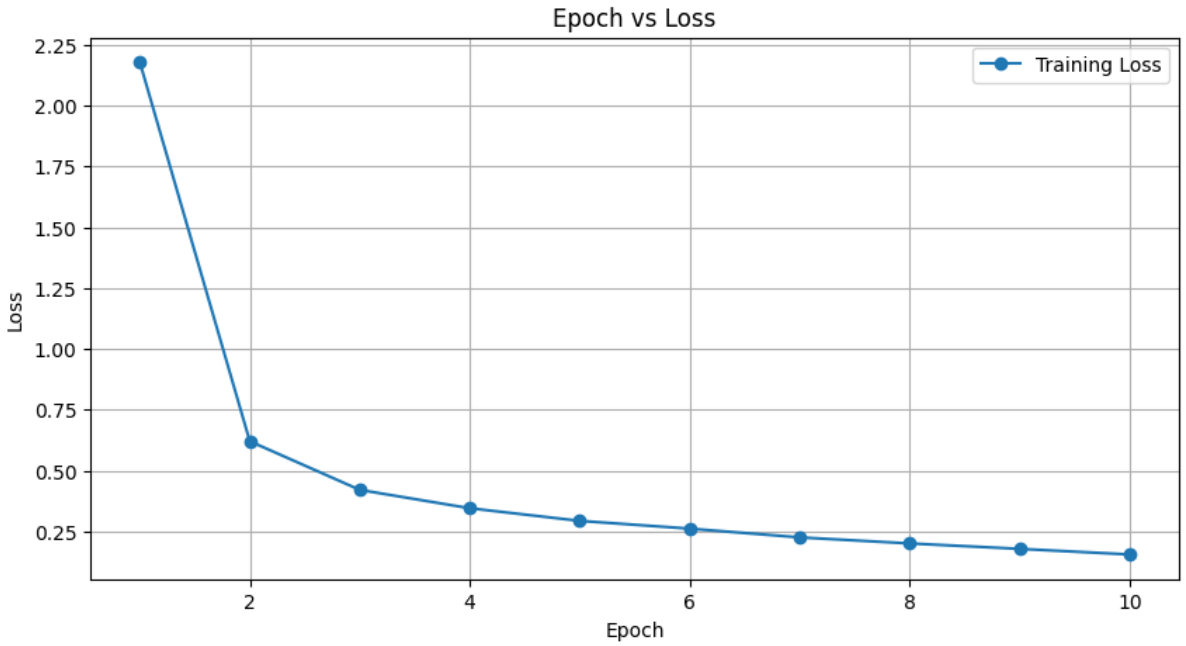
برای جبران تفاوت بین تعداد نورون‌های فعال در طول آموزش و آزمون، خروجی نورون‌ها در زمان آزمون کاهش می‌یابد (معمولاً با ضریبی که برابر با احتمال حفظ نورون در طول آموزش است، ضرب می‌شود). این کار تضمین می‌کند که انتظارات تولیدی شبکه در طول آموزش و آزمون مطابقت داشته باشد

Batch normalization

کد های این بخش را درون یک فایل جوپیتر نوت بوک به نام resnet_without_bach_normalization قرار داده ام .

در این بخش داده ها را روی مدل resnet18 که قبلا پیاده سازی کردیم منتهی بدون Batch normalization آموزش و ارزیابی کرده ام که نتایج زیر حاصل شده است .

نمودار تغییرات خطا روی داده های آموزشی در این حالت اینگونه شد



همچنین دقت مدل روی داده های تست SVHN 91 درصد شد وروی داده های تست MNIST 68 درصد شد

Accuracy on SVHN Test Set: 91.3759987707437%

Accuracy on MNIST Test Set: 68.96%

LOSS FUNCTION

تکنیک نرم‌سازی برچسب یا Label Smoothing Regularization یک روش تنظیمی (Regularization) است که به مدل کمک می‌کند تا نسبت به داده‌های آموزشی بیش از حد اطمینان پیدا نکند و به این ترتیب، از افراط آموزش یا Overfitting جلوگیری کند. این روش در شبکه‌های عصبی دسته‌بندی کاربرد دارد و به ویژه هنگام استفاده از تابع خطای Cross-Entropy مفید است.

در یک مدل طبقه بندی که از Cross-Entropy استفاده می‌کند، برچسب‌های واقعی (True Labels) معمولا به صورت one-hot encoded هستند. یعنی برای هر نمونه، یک برچسب درست دارای مقدار ۱ است و بقیه مقادیر صفر هستند. نرم‌سازی برچسب‌ها با تغییر دادن برچسب‌های one-hot به گونه‌ای عمل می‌کند که به جای استفاده از ۱ کامل، از یک مقدار کمتری استفاده می‌کند و بقیه احتمالات را به صورت مساوی بین سایر کلاس‌ها توزیع می‌کند.

کد های این قسمت را در یک فایل جوپیتر نوت بوک به نام Resnet_With_Label_Smoothing نوشته ام

در کد این بخش از تابع خطای label smoothing استفاده کرده ام و آن را به صورت دستی نوشته ام .

```
class LabelSmoothingCrossEntropy(nn.Module):
    def __init__(self, epsilon=0.25):
        super(LabelSmoothingCrossEntropy, self).__init__()
        self.epsilon = epsilon

    def forward(self, input, target):
        num_classes = input.size(-1)
        log_preds = F.log_softmax(input, dim=-1)
        targets = torch.zeros_like(log_preds).scatter_(1, target.unsqueeze(1), 1)
        targets = (1 - self.epsilon) * targets + self.epsilon / num_classes
        loss = (-targets * log_preds).sum(dim=-1).mean()
        return loss
```

کد بالا را به این شکل پیاده سازی کرده ا م

:log_preds = F.log_softmax(input, dim=-1) لاگ احتمالات نرم شده (Log-Softmax) برای پیش‌بینی‌های مدل را محاسبه می‌کند.

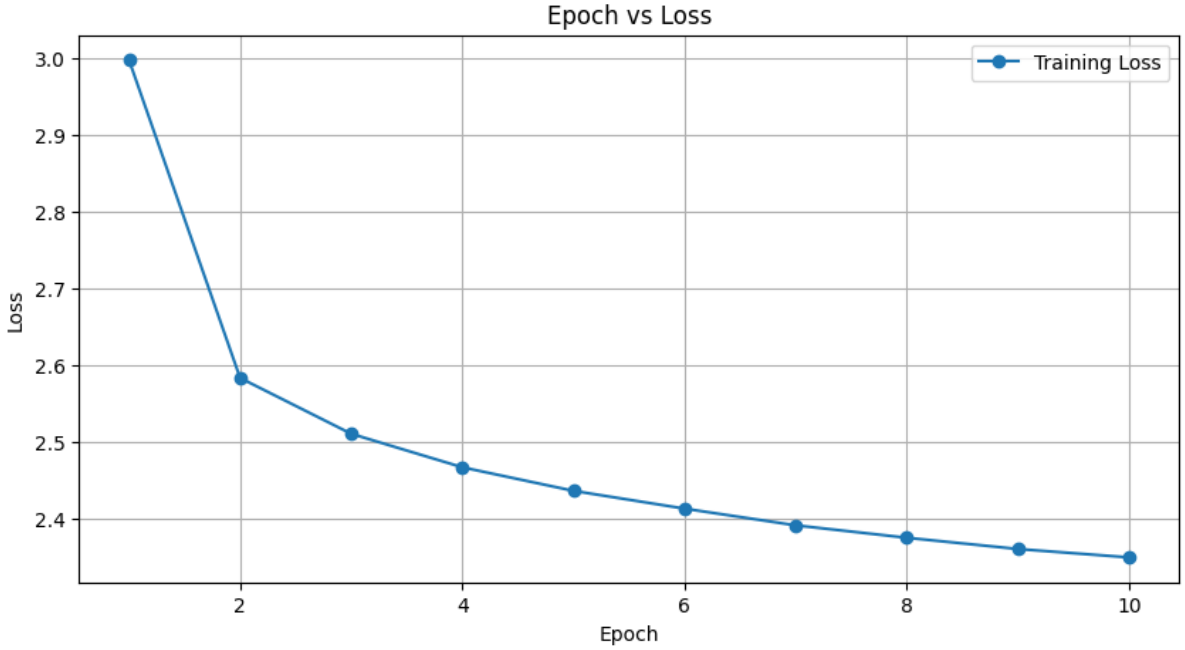
:targets = torch.zeros_like(log_preds).scatter_(1, target.unsqueeze(1), 1) برچسب‌های one-hot را می‌سازد. این کار با ایجاد یک تنسور صفر که همان شکل ورودی را دارد

و سپس استفاده از تابع scatter_ برای قرار دادن ۱ در ایندکس‌های مربوط به کلاس‌های واقعی انجام می‌شود.

targets = (1 - self.epsilon) * targets + self.epsilon / num_classes: نرم‌سازی برچسب‌ها با توزیع یک مقدار کوچک epsilon به کلاس‌های دیگر و حفظ بقیه اطمینان برای کلاس صحیح.

loss = (-targets * log_preds).sum(dim=-1).mean(): خطای نهایی را محاسبه می‌کند. این از طریق ضرب هر برچسب نرم‌سازی شده در لاگ احتمال مربوطه و محاسبه میانگین آن برای تمام نمونه‌ها انجام می‌شود.

بقیه پارامتر ها را مانند مدل پایه گذاشته ام و فقط تابع خطا را عوض کرده ام . نمودار تغییرات خطا به شکل زیر شد .



نمودار تغییرات خطا با تابع خطای label smoothing croos entropy

نمودار نشان میدهد که خطا به خوبی کاهش یافته است و مدل در حال آموش است .

همچنین مقدار خطا برای داده های SVHN 92 درصد و برای داده های MNIST 70 درصد شد که نشان میدهد مدل تعمیم دهی خوبی نسبت به حالت اولیه که دقت 64 درصد بود پیدا کرده است .

DATA AUGMENTATION

کد های این قسمت را در یک فایل jupytet notebook به نام
resnet_Data_agumentation قرار داده ام .

با توجه به زمینه سوال ، برخی از انواع افزایش داده ممکن است به دلیل ماهیت داده‌ها مناسب نباشند مانند :

۱. افزایش مبتنی بر رنگ: از آنجایی که MNIST یک مجموعه داده سیاه و سفید است و SVHN رنگی است اما عمدتاً بر روی ارقام عددی تمرکز دارد، افزایش‌های مبتنی بر رنگ (مانند تغییرات جزئی در رنگ) ممکن است خیلی مفید نباشند. برای MNIST، افزایش‌های رنگی بی‌ربط هستند و برای SVHN، می‌توانند در صورت استفاده بیش از حد، از یادگیری شکل ارقام حواس‌پرتی کنند

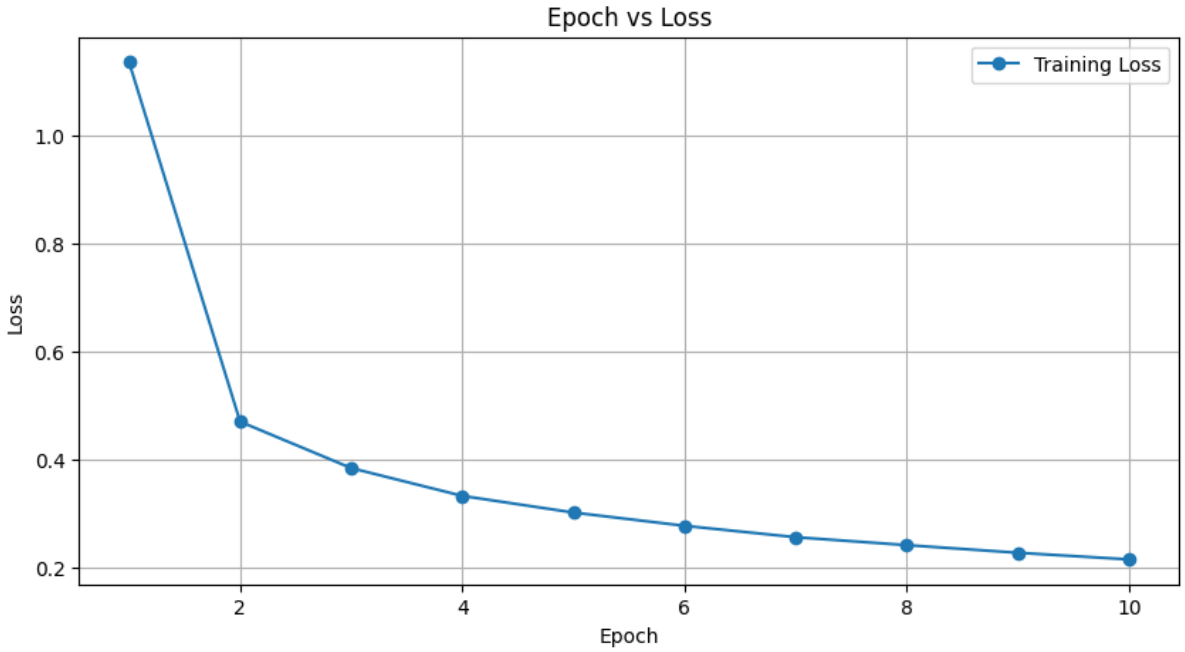
۲. تغییرات هندسی پیچیده: در حالی که چرخش‌های جزئی و مقیاس‌بندی‌ها می‌توانند به مدل کمک کنند تا به این تغییرات بی‌تفاوت شود، تغییرات هندسی شدید می‌تواند مسئله را به طور قابل توجهی سخت‌تر کند. به عنوان مثال، چرخش‌های بزرگ یا مقیاس‌بندی‌های شدید می‌توانند ارقام عددی را فراتر از شناسایی تحریف کنند، که یادگیری نگاشت‌های صحیح توسط مدل را دشوار می‌کند. اعداد جهت هایی دارند که برای شناسایی آن‌ها مهم است (به عنوان مثال، '6' با چرخش ۱۸۰ درجه به '9' تبدیل می‌شود)، بنابراین چرخش‌های بزرگ می‌توانند گمراه کننده باشند.

۳. چرخش افقی: این نوع افزایش می‌تواند معنای ارقام را تغییر دهد (به عنوان مثال، چرخاندن '6' به صورت افقی می‌تواند آن را شبیه '9' کند و بالعکس)، که افزایش مطلوبی برای مجموعه داده‌ای که بر روی شناسایی رقم متمرکز است، نیست.

در این قسمت با توجه به روش های data augmentation سعی کرده ام تعمیم پذیر مدل را افزایش دهم به همین خاطر با سعی و خطا های مختلف روش های زیر را برای data augmentation اعمال کردم.

اعمال تغییرات افاین تصادفی شامل چرخش ملایم تصاویر (تا 5 درجه)، جابجایی تصاویر تا 5% از کل ابعاد تصویر به هر سمت، و مقیاس‌بندی (تغییر اندازه تصویر به مقیاس‌های بین 95% تا 105% از اندازه اصلی). این نوع از تغییرات به مدل کمک می‌کند تا از وابستگی به موقعیت، اندازه و جهت ارقام فراتر رود و بهتر تعمیم دهد.

مدل را آموزش داده ام و نمودار تغییرات خطا برای آن به این شکل شد .



نمودار تغییرات خطا با data augmentation

طبق نمودار تغییرات خطا کاهشی و پیوسته است که نشان میدهد مدل به خوبی در حال آموزش است

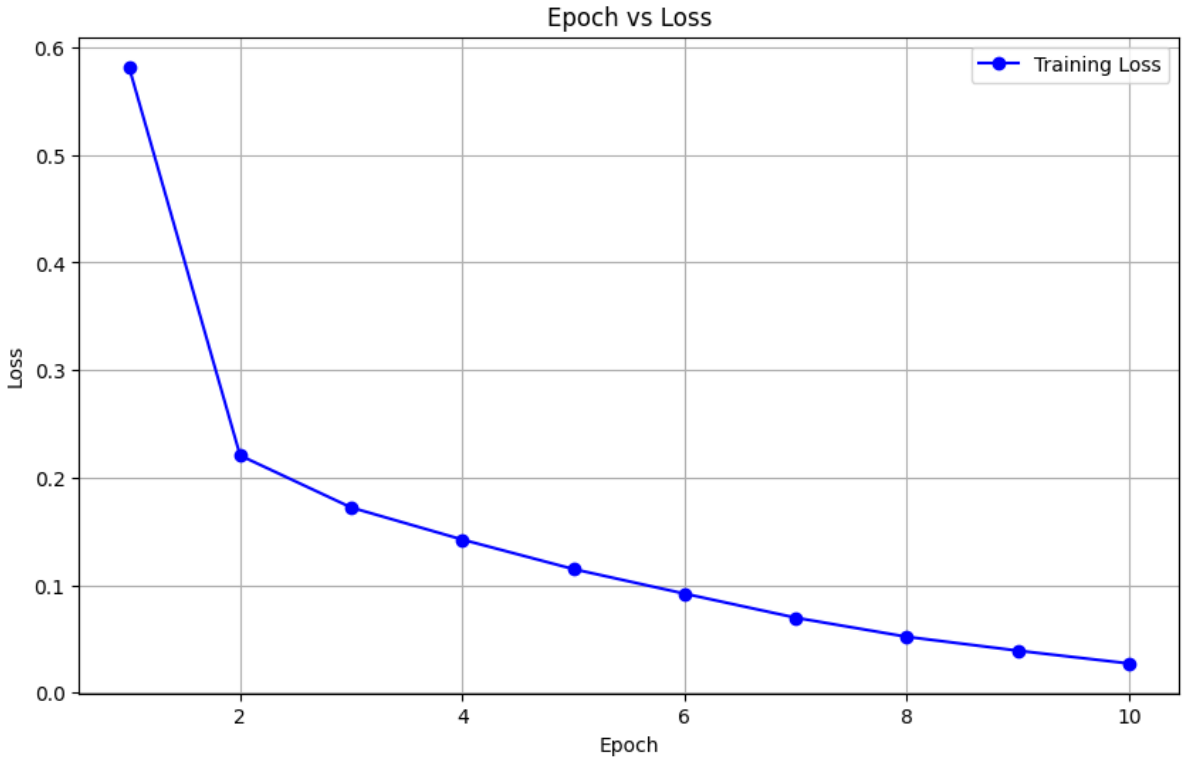
همچنین دقت مدل روی داده های SVHN 93 درصد و دقت روی داده های MNIST 72 درصد شد که نشان میدهد Generalization مدل نسبت به حالتی که از data augmentation استفاده نکردیم بیشتر شده و روش هایی که اعمال کردیم برای افزایش داده مناسب بوده است .

مقایسه : دقت در حالت پایه 64 و در این حالت 72 درصد شد .

INPUT FEATURES (FEATURE EXTRACTION)

در این بخش از مدل پیش آموزش دیده resnet18 که روی داده های imagw net آموزش دیده است استفاده کرده ام و داده ها را روی آن آموزش داده ام که نتایج آن اینگونه شد .

نمودار تغییرات خطا برای این بخش به شکل زیر شد



نمودار تغییرات خطا با مدل از پیش آموزش دیده resnet 18

همچنین دقت روی داده های SVHN 95 درصد و دقت روی داده های MNIST 75 درصد شد .

Accuracy on SVHN Test Set: 95.86%

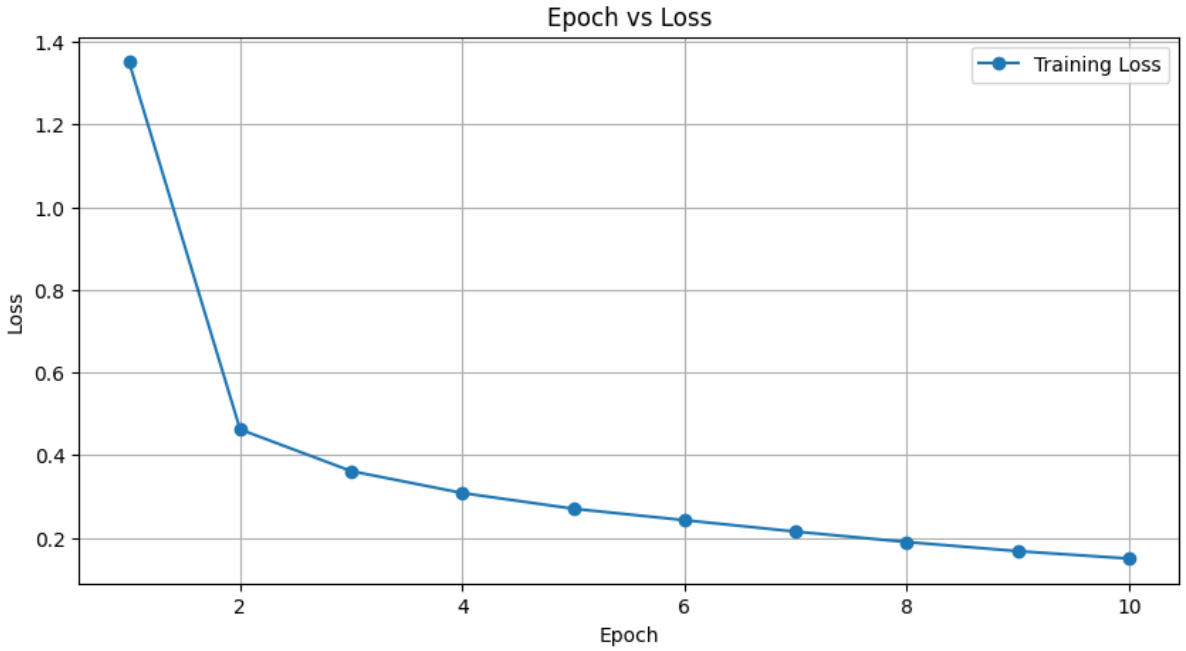
Accuracy on MNIST Test Set: 75.30%

در این بخش نتایج بسیار خوبی و بهتری نسبت به روش های قبل بدست آوردیم به علت اینکه مدل پیش آموزش دیده resnet روی دیتاست بزرگ image net آموزش دیده وبسیاری از ویژگی ها را یاد گرفته و روی این دیتاست ها هم به خوبی جواب میدهد و قدرت تعمیم دهی بسیار خوبی دارد .

OPTIMIZER

در این بخش تنظیمات اولیه را تغییر نداده ام و فقط از تابع بهینه ساز آدادم استفاده کرده ام

نمودار تغییرات خطا با این روش به شکل زیر شد



طبق نمودار خطا با گذشت زمان کاهش کاهش پیدا میکند که نشان میدهد مدل در حال یادگیری است

دقت مدل روی داده های تست SVHN 92 و روی داده های تست MNIST 69 درصد شد که نشان میدهد زمانی که در اینجا که ما از بهینه ساز Adam استفاده کردیم به Generalization بهتری دست پیدا کردیم .

آموزش معکوس مدل

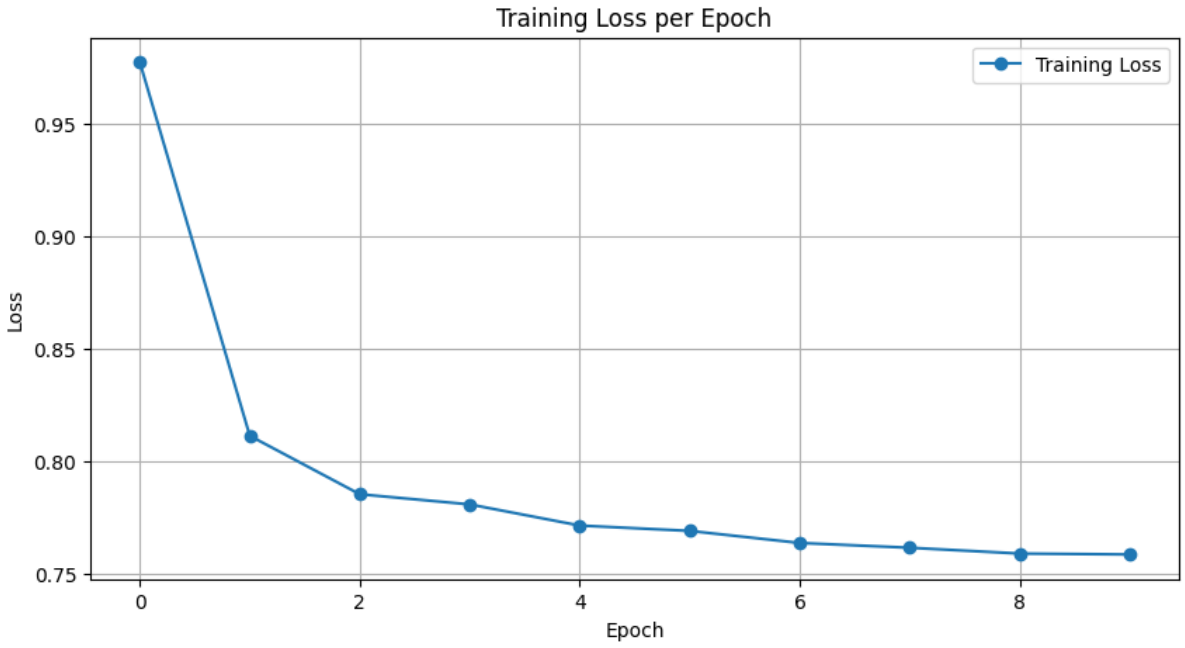
UNSUPERVISED

کد های قسمت unsupervised و supervise را در یک فایل جوپیتر نوت بوک به نام Unsupervise_Supervise قرار داده ام

در این بخش بال توجه به نتایج بخش های قبلی از بهترین setting ممکن استفاده کرده ام ولی اینبار مدل را روی داده های mnist آموزش و روی داده های SVHN ارزیابی کرده ام

بهترین setting هایی که بدست آوردم : مدل از پیش آموزش دیده resnet18 همراه با Data augmentation و تابع خطای Label Smoothing Cross Entropy و بهینه ساز Adam

نمودار تغییرات خطا به این شکل شد



همچنین دقت روی داده های تست mnist 92 درصد و روی داده های تست SVHN حدود 12 در صد شد

Accuracy on MNIST Test Set: 91.86%

Accuracy on SVHN Test Set: 11.89%

تحلیل :

با توجه به اینکه ما اینبار مدل را روی داده های MNIST آموزش داده ایم و این دیتاست در مقابل SVHN دیتاست ساده تری است زمانی که مدل را ارزیابی میکنیم دقت روی داده های تست SVHN بالا نیست ولی زمانی که روی داده های SVHN مدل را آموزش میدهیم SVHN به طور کلی پیچیده‌تر و متنوع‌تر از MNIST هستند. این بدان معناست که مدلی که روی SVHN آموزش می‌بیند، باید یاد بگیرد که چگونه با تنوع بیشتری از شرایط تصویری، از جمله رنگ‌ها، سایه‌ها و پس‌زمینه‌های ناهمگن کنار بیاید. که کمک میکند دقت روی داده های تست MNIST بالا رود.

SUPERVISED

در این حالت از همان مدل حالت قبل استفاده کرده ایم تنها با این تفاوت که میخواهیم وقتی مدل روی داده های MNIST آموزش دید لایه های کانولوشنی را Freeze کنیم و تعدادی از داده های SVHN را روی Classifier نهایی Fine tune کنیم و در این حالت دقت مدل را اندازه گیری کنیم

برای این کار 1000 تا از داده های SVHN را در مرحله Fine tune استفاده کرده ام و این بار دوباره روی داده های تست mnist و SVHN ارزیابی را انجام داده ام که نتایج زیر حاصل شده است

Accuracy on MNIST Test Set: 69.09%

Accuracy on SVHN Test Set: 28.42%

همانطور که انتظار داشتیم شد چون که در این حالت مدل توزیع داده های SVHN را دیده است بنابراین دقت مدل روی داد های SVHN افزایش یافته و24 درصد به رسیده ولی روی داده کاهش MNIST یافته و به 69 درصد رسیده است . ولی در کل Generalization مدل افزایش یافته است.

سؤال دوم

بخش الف

FGSM یک روش سریع و کارآمد برای تولید نمونه‌های متخاصم است. این روش با اعمال یک اغتشاش کوچک اما تأثیرگذار در جهت گرادیان تابع خطا نسبت به داده‌های ورودی کار می‌کند. ایده این است که خطا را با تنظیم داده‌های ورودی به میزان کوچک اپسیلون (ϵ) در جهتی که خطا را افزایش می‌دهد، به حداکثر برساند، در نتیجه منجر به یک پیش‌بینی نادرست شود.

$$x' = x + \epsilon \cdot sign(\nabla_x J(\theta, x, y))$$

که در آن $\nabla_x J(\theta, x, y)$ گرادیان تابع زیان J نسبت به داده‌های ورودی x ، θ نماینده پارامترهای مدل، و y برچسب واقعی و x' نمونه متخاصم است .

PGD یک نسخه تکراری از FGSM است و می‌توان آن را به عنوان اجرای چندین بار FGSM با اندازه قدم‌های کوچک در نظر گرفت. ایده پشت PGD این است که چندین قدم کوچک در جهت گرادیان تابع خطا بردارد و اگر لازم باشد نمونه اغتشاش یافته را به محدوده اطراف نمونه اصلی برگرداند .

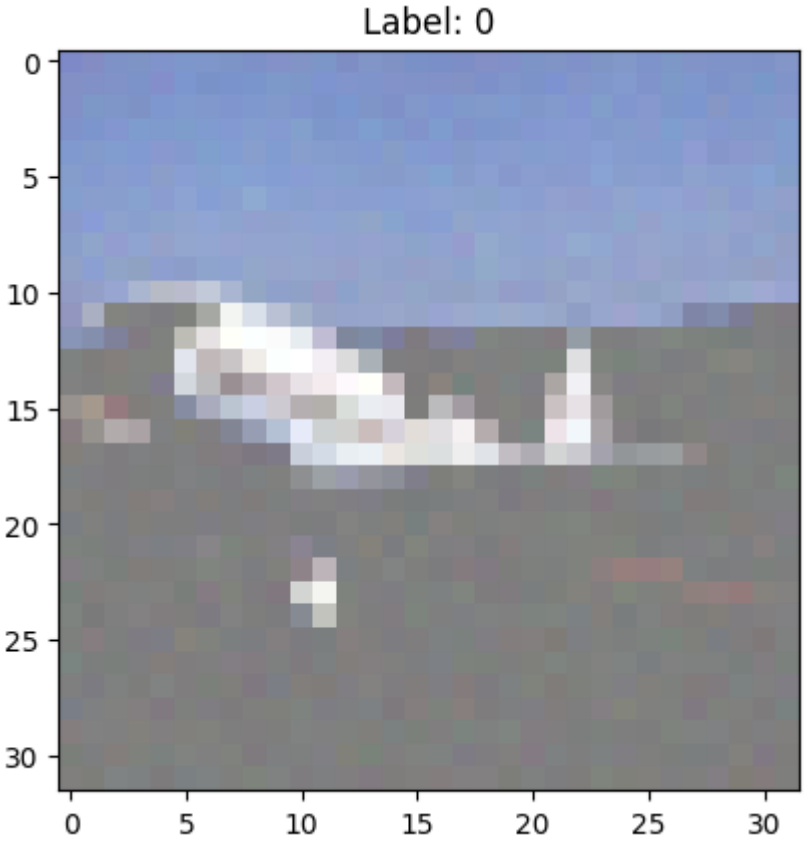
نمونه متخاصم به صورت زیر به‌روزرسانی می‌شود:

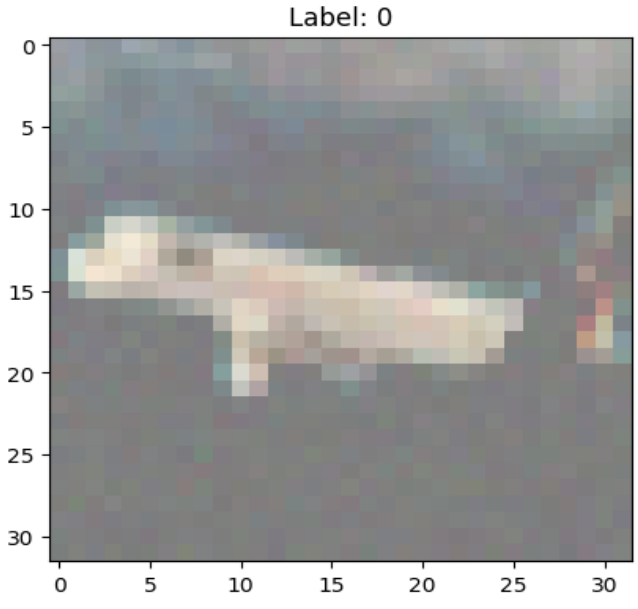
$$x'_{t+1} = Proj_{x+S}(x'_t + \alpha \cdot sign(\nabla_x J(\theta, x'_t, y)))$$

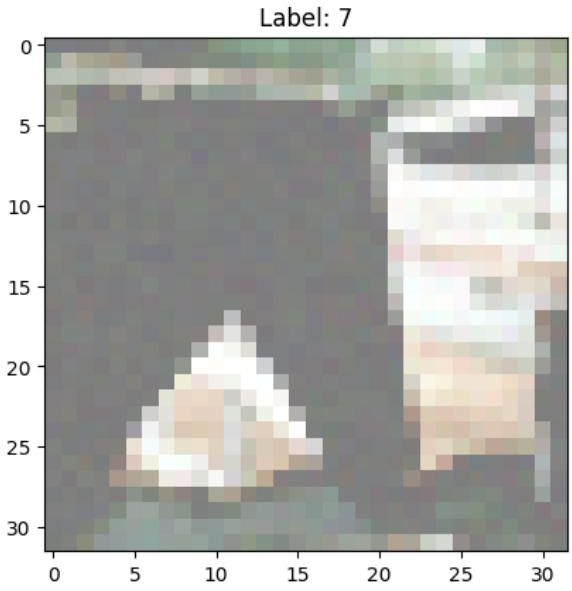
که در آن $x't$ نمونه متخاصم در تکرار t و α اندازه قدم است

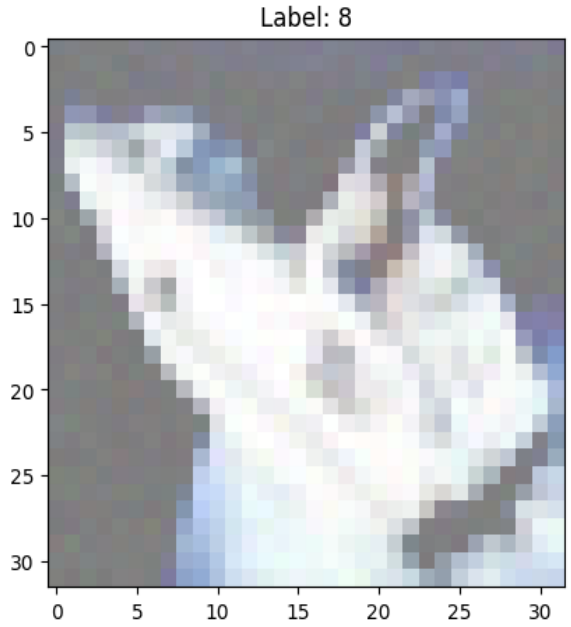
داده های تست را با روش FGSM به نمونه های Adversarial با پارامتر 0.1 تبدیل کرده ام و به آن ها نویز اضافه کرده ام که در ادامه تعدادی از این تصاویر را مشاهده میکنیم .

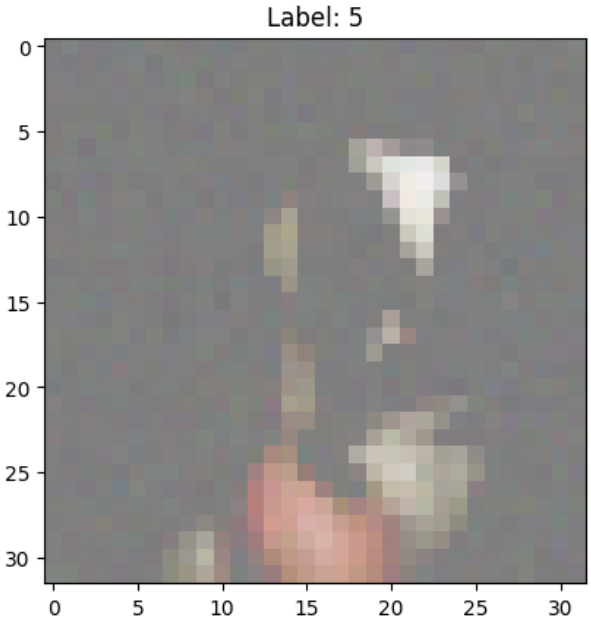
(چون که تصاویرCIFAR 10 در ابعاد 32 در 32 هستند و من در اینجا در ابعاد بالاتر نمایش داده ام کمی تصاویر ناواضح به نظر میرسد)


Label: 0
0
5
10
15
20
25
30
0
5
10
15
20
25
30


Label: 0
0
5
10
15
20
25
30
0
5
10
15
20
25
30


Label: 7
0
5
10
15
20
25
30
0
5
10
15
20
25
30

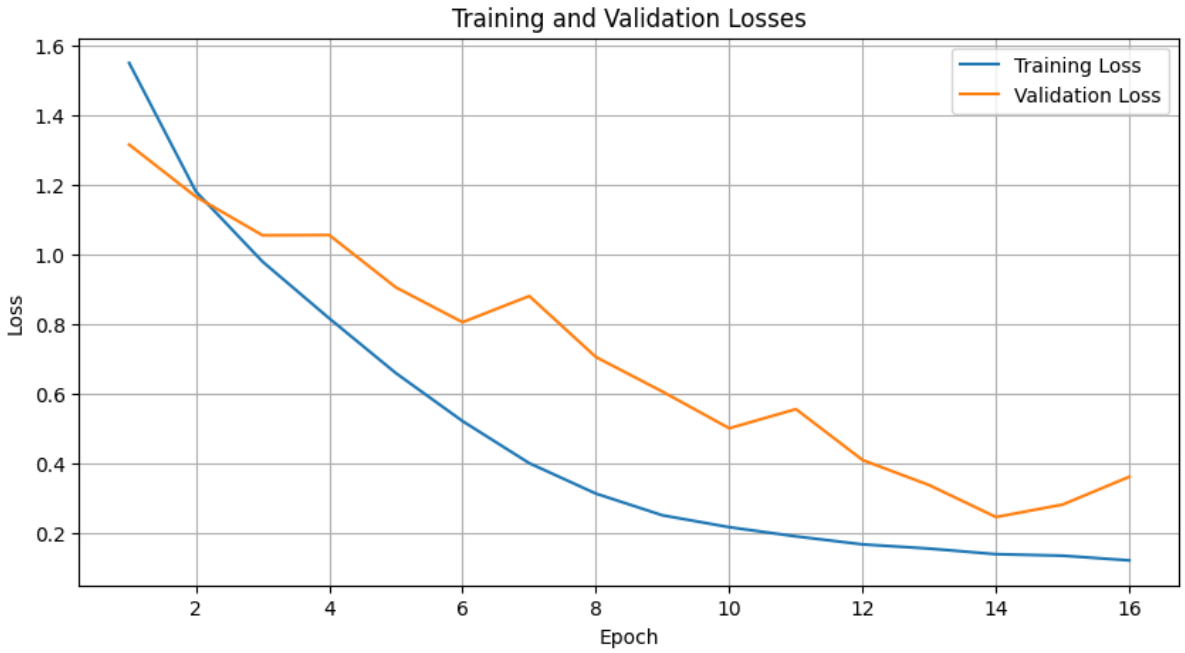

Label: 8
0
5
10
15
20
25
30
0
5
10
15
20
25
30



در ابتدا داده های را به سه دسته آموزش و ارزیابی و تست تقسیم کردم . برای این کار 80 درصد داده ها را برای آموزش و 20 درصد برای تست و 20 درصد از داده های آموزشی را برای ارزیابی قرار داده ام .

در ادامه resnet18 را آپلود کرده ام و برای آموزش از بهینه ساز Adam استفاده کرده ام داده ها را در 16 ایپاک آموزش داده ام (به دلیل اینکه در ایپاک های بیشتر مدل overfit میکرد early stopping زدم و به همین تعداد ایپاک بسنده کردم)

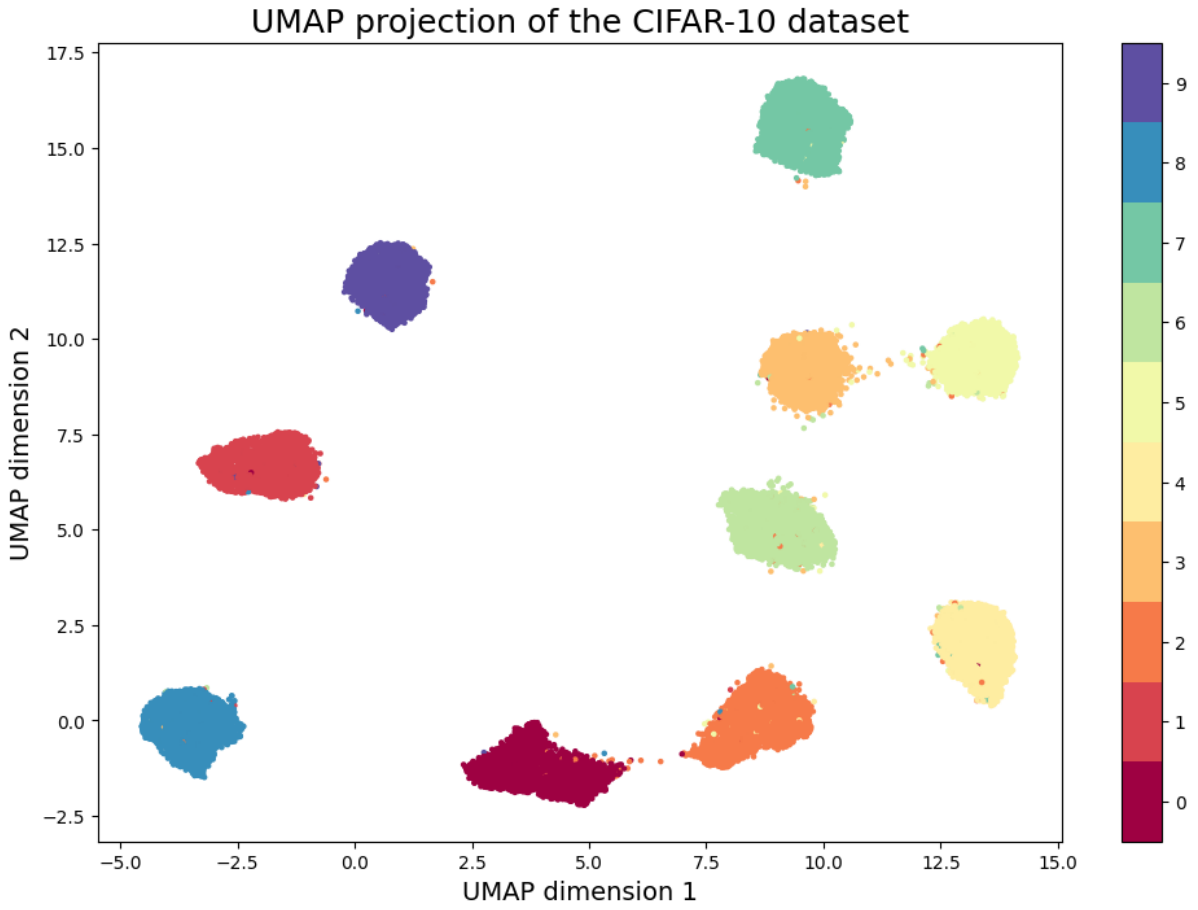
نمودار تغییرات خطا برای داده های آموزشی و ارزیابی مطابق شکل زیر شد



نمودار تغییرات خطا برای بخش الف

با توجه به نمودار خطای آموزش در طول دوره‌ها به طور پیوسته کاهش می‌یابد، که نشان‌دهنده یادگیری و بهبود عملکرد مدل در داده‌های آموزشی است.

همچنین نمودار UMAP را برای داده های آموزش ترسیم کرده ام که به این شکل شد .



نمودار UMAP برای داده های آموزشی بخش الف

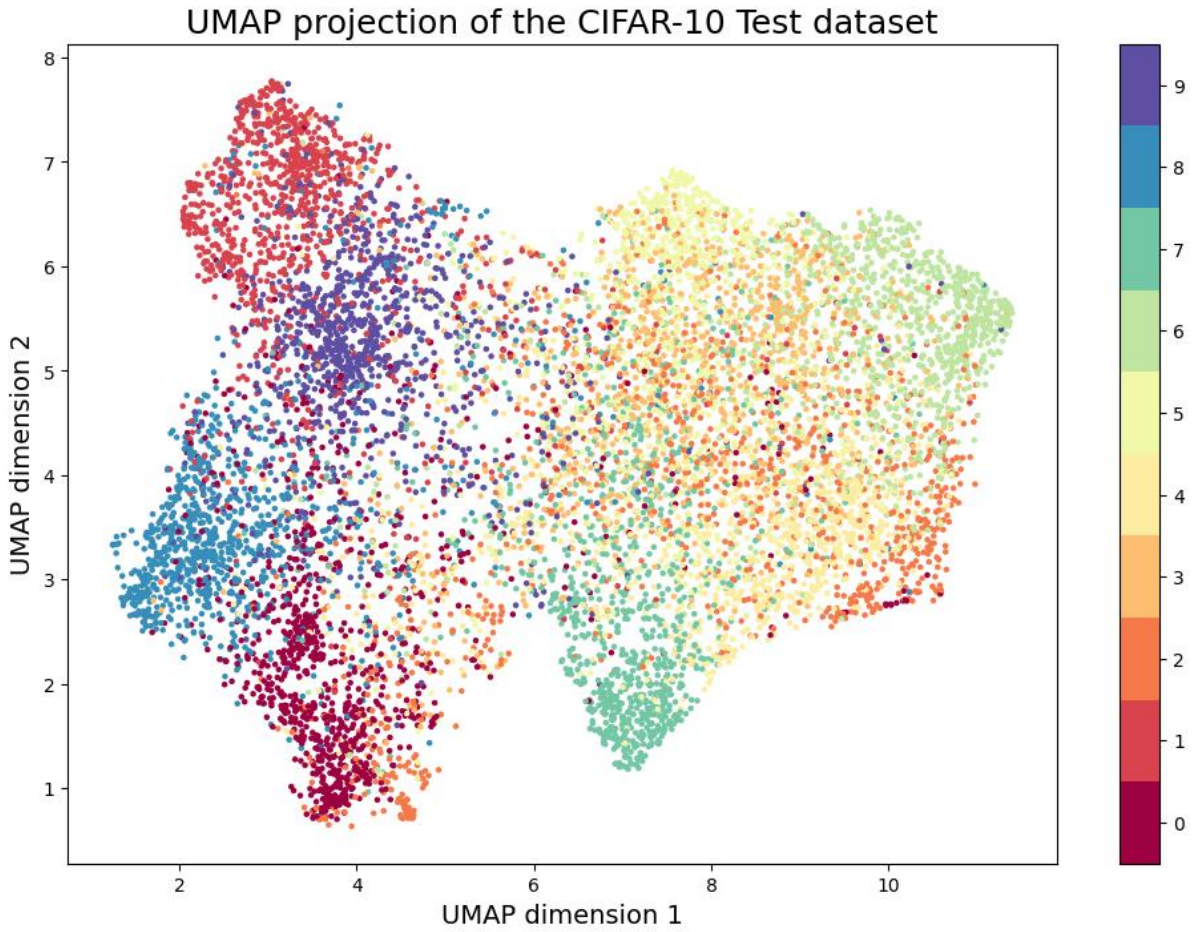
تحلیل نمودار بالا :

با توجه به نمودار کلاس ها به خوبی از هم تفکیک شده‌اند و در قسمت‌های مختلف نمودار قرار گرفته‌اند. این بدان معناست که ویژگی‌های استخراج شده توسط مدل می‌توانند برای تمایز بین این کلاس‌ها مفید باشند.

همچنین برخی از کلاس‌ها نزدیک به هم قرار گرفته‌اند، که ممکن است نشان‌دهنده تداخل بین کلاس‌ها و چالش در تمایز بین آن‌ها باشد. این می‌تواند به دلیل شباهت‌های بصری بین دسته‌ها یا محدودیت‌های ویژگی‌های استخراج شده از تصاویر باشد.

در ادامه مدل را بر روی داده های تست ارزیابی کردم که به دقت حدود 70 درصد رسید که دقت مناسبی به نظر میرسد

همچنین نمودار UMAP داده های تست به این شکل شد .



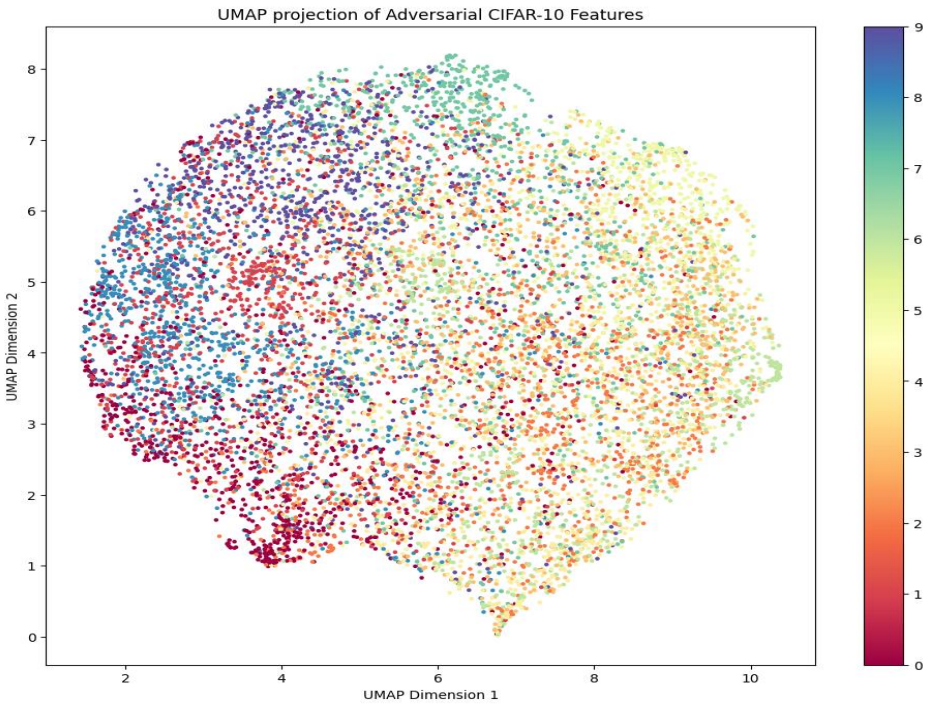
از این جای کد میخواهیم عملکرد مدل را روی داده های تست Adversarial مشاهده کنیم برای این کار ابتدا با روش FGSM داده های Adversarial ایجاد کرده ام و در ادامه مقدار نویز به آن اضافه کرده ام برای این کار از دو تابع زیر استفاده کرده ام .

```
def fgsm_attack(image, epsilon, data_grad):
    sign_data_grad = data_grad.sign()
    perturbed_image = image + epsilon * sign_data_grad
    perturbed_image = torch.clamp(perturbed_image, 0, 1)
    return perturbed_image
```

```python
def add_random_noise(image, noise_level=0.01):
    noise = torch.randn_like(image) * noise_level
    noisy_image = image + noise
    noisy_image = torch.clamp(noisy_image, 0, 1)
    return noisy_image
```

در این حالت مدل به دقت 48 درصد دست یافت

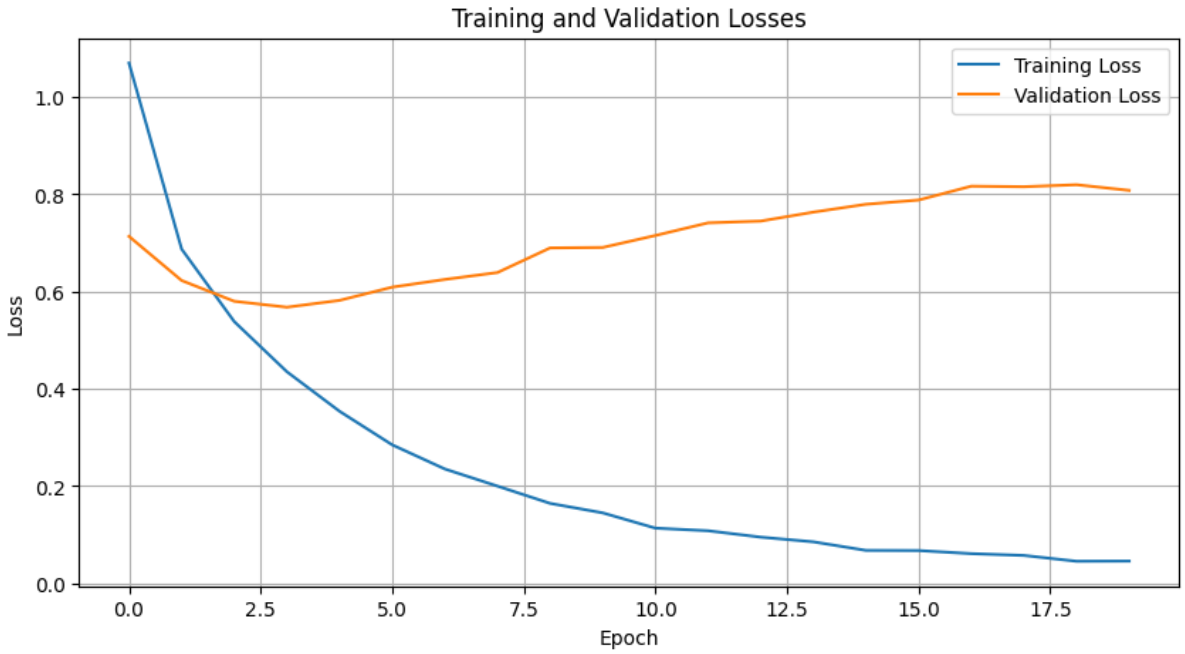
همچنین نمودار UMAP روی داده های Adversarial به این شکل شد .



بخش ب

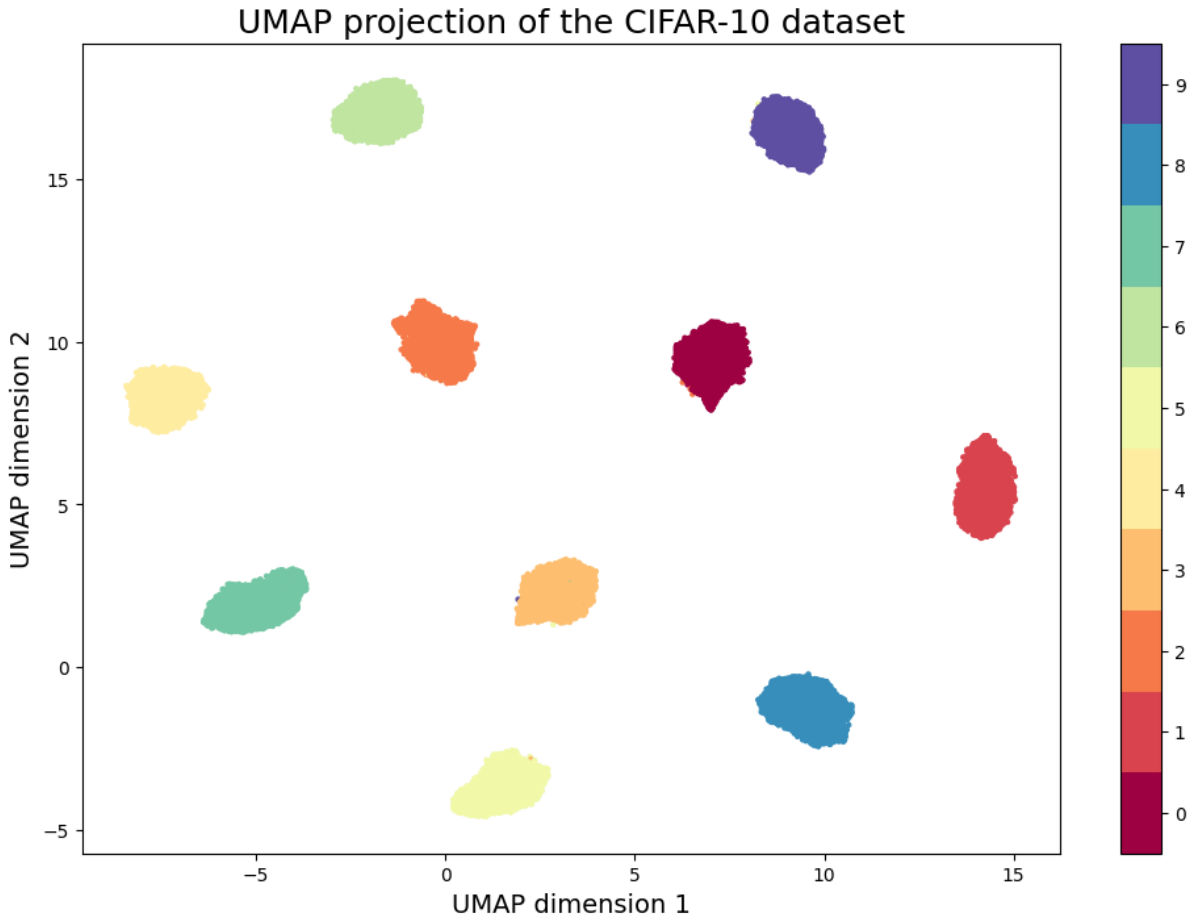
در این بخش از دیتاست augment شده برای آموزش مدل استفاده کرده ام به این صورت که در روند آموزش از روش FGSM استفاده کرده ا م و در هر batch به احتمال 50 درصد داده ها را Adversarial کرده ام و در روند آموزش استفاده کرده ام

مدل را آموزش داده ام که تغییرات خطا روی داده های آموزشی و ارزیابی ، نمودار زیر شد



نمودار تغییرات خطا روی داده های آموزش و ارزیابی قسمت ب

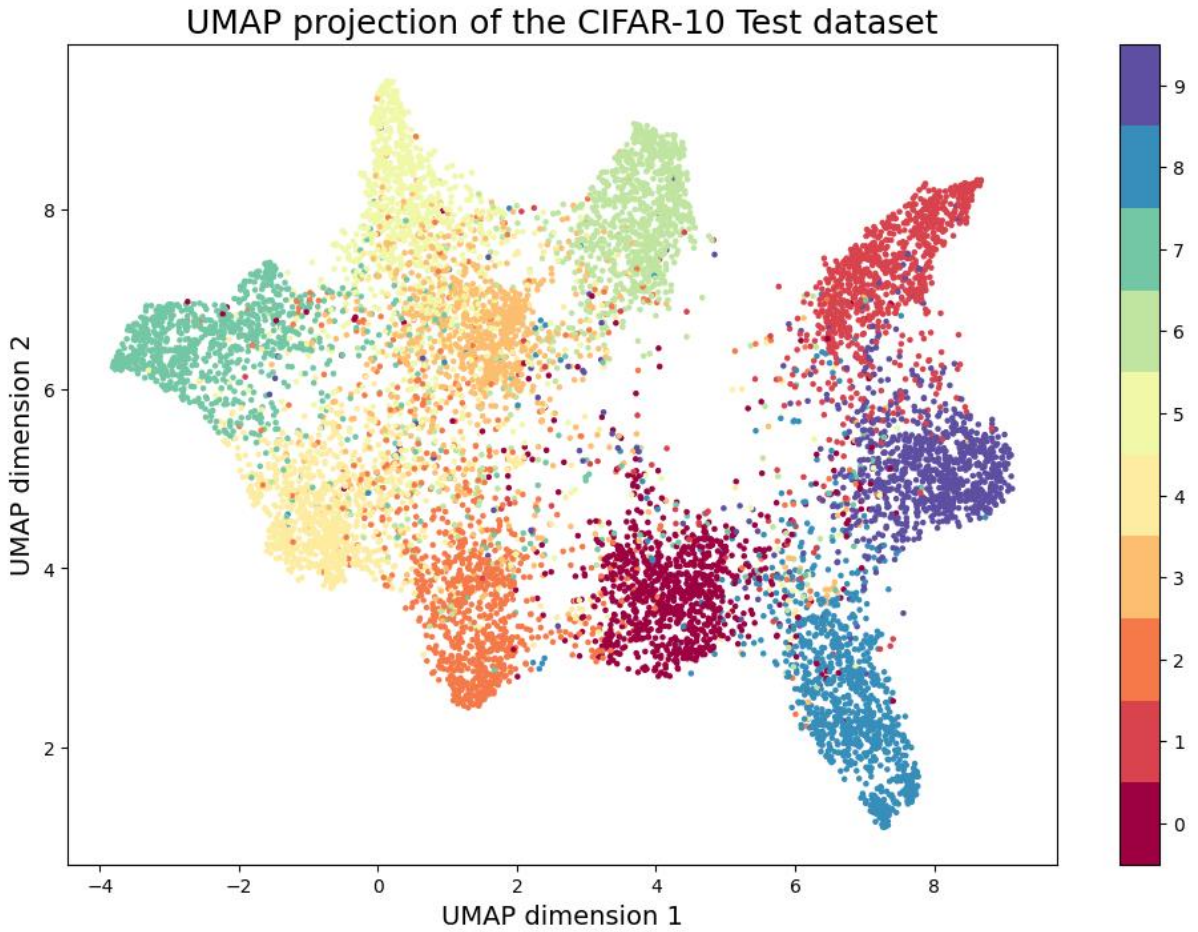
همچنین نمودار UMAP مربوط به داده های آموزشی اینگونه شد



نمودار UMAP مربوط به داده های آموزشی

از نمودار به طور واضح مشخص است که ما از روش قبلی بهتر عمل کردیم و داده های به خوبی و با دقت مناسب طبقه بندی شده اند .

دقت مدل روی داده های تست 82 درصد شد که دقت بسیار مناسبی است و از روش قبلی بهتر شد
همچنین نمودار UMAP مربوط به داده های تست اینگونه شد

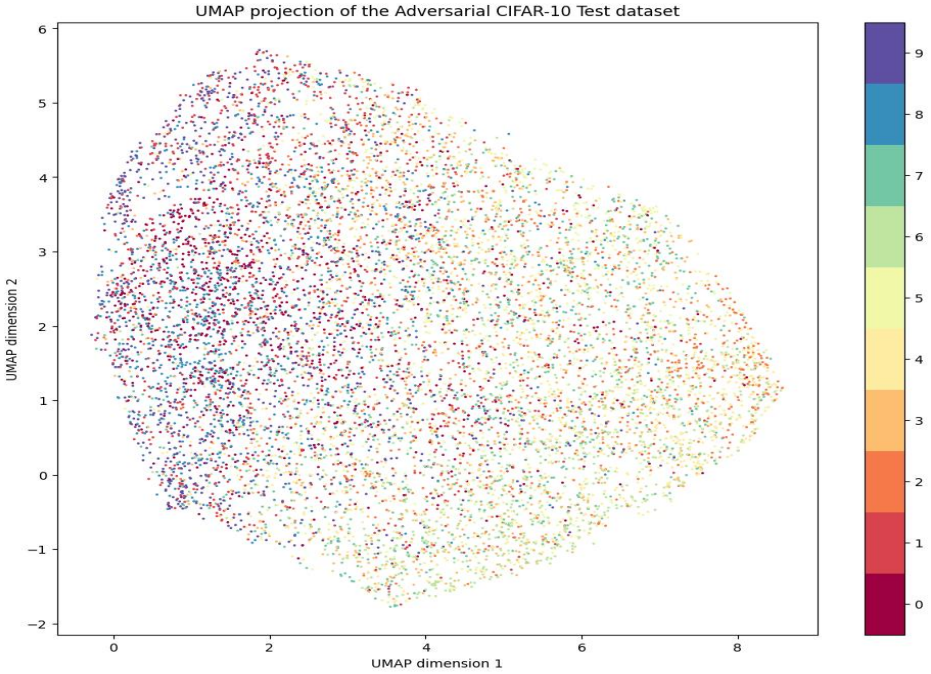


نمودار UMAP داده های تست برای قسمت ب

طبق نمودار داده‌ها در فضای دوبعدی به صورت یک توزیع پیوسته نشان داده شده‌اند که نقاط مختلف به وضوح از هم تفکیک شده‌اند، این نشان می‌دهد که فضای ویژگی‌ها ممکن است اطلاعات کافی برای تمایز بین کلاس‌ها را داشته باشد.

در ادامه داده های تست را با روش FGSM به نمونه های Adversarial تبدیل کرده ام و همچنین نویز به آن اضافه کردم و مدل را با این داده ها ارزیابی کردم و به دقت 53 درصد رسیدم که نسبت به روش قبلی دقت بهتری است

همچنین نمودار UMAP مربوط به این قسمت اینگونه شد



نمودار UMAP مربوط به داده های تست Adversarial

نتیجه گیری :

در این قسمت با توجه به روشی که اتخاذ کردیم به وضوح به نتایج بهتری نسبت به روش قبلی رسیدیم

بخش ج

Circle Loss دیدگاه تازه‌ای در بهینه سازی یادگیری ویژگی عمیق با تاکید بر بهینه‌سازی شباهت زوج ارائه می‌دهد. ایده اصلی این است که شباهت درون کلاسی را به حداکثر رسانده و شباهت بین کلاسی را به حداقل برساند، هدفی که برای بسیاری از توابع خطا مانند خطای تریپلت (triplet) و خطای softmax cross-entropy softmax مشترک است. با این حال، Circle Loss با رفع محدودیت‌های این روش‌های سنتی، رویکردی انعطاف‌پذیرتر و هدفمندتر در بهینه‌سازی ارائه می‌کند

در روش‌های قبلی مانند triplet loss، انتخاب نمونه‌های مناسب (anchor, positive, negative) می‌تواند چالش‌برانگیز باشد و تعادل بین نمونه‌های سخت و آسان برای حفظ است. Circle Loss با تنظیم خودکار وزن نمونه‌ها، این مشکل را برطرف می‌کند و به طور هوشمندانه روی نمونه‌هایی که نیاز به بهینه‌سازی بیشتری دارند، تمرکز می‌کند.

همچنین روش‌های مبتنی بر triplet یا pair به انتخاب دقیق نمونه‌ها نیاز دارند که می‌تواند پیچیدگی محاسباتی آموزش را افزایش دهد. Circle Loss با کاهش نیاز به انتخاب دستی نمونه‌ها و فراهم آوردن یک رویکرد سیستماتیک، این پیچیدگی را کاهش می‌دهد

در حالی که توابع هزینه سنتی مانند cross-entropy برای بسیاری از کاربردها مناسب هستند، آن‌ها ممکن است در مواجهه با چالش‌های خاص یادگیری عمیق (مانند یادگیری با نظارت کم) انعطاف‌پذیری کافی نداشته باشند. Circle Loss با ارائه یک رویکرد قابل تنظیم برای تمام نمونه‌ها، به محققان امکان می‌دهد تا بهینه‌سازی شباهت زوج را با دقت بیشتری کنترل کنند.

بخش د

کد های این قسمت را در یک فایل جوپیتر نوت بوک به نام Q2_part3 نوشته ام .

برای حل این سوال تابع circle loss موجود در مقاله را بصورت دستی به شکل زیر پیاده سازی کرده ام .

```
def convert_label_to_similarity(normed_feature: Tensor, label: Tensor) -> Tuple[Tensor, Tensor]:
    similarity_matrix = normed_feature @ normed_feature.transpose(1, 0)
    label_matrix = label.unsqueeze(1) == label.unsqueeze(0)

    positive_matrix = label_matrix.triu(diagonal=1)
    negative_matrix = label_matrix.logical_not().triu(diagonal=1)

    similarity_matrix = similarity_matrix.view(-1)
    positive_matrix = positive_matrix.view(-1)
    negative_matrix = negative_matrix.view(-1)
    return similarity_matrix[positive_matrix], similarity_matrix[negative_matrix]
```

این تابع رابرای محاسبه شباهت بین جفت‌های نمونه‌ها در یک مجموعه داده بر اساس ویژگی‌های نرمال‌شده و برچسب‌های آن‌ها نوشته ام .

هدف اصلی این تابع، تولید دو تنسور است: یکی برای شباهت‌های مثبت (نمونه‌هایی با برچسب یکسان) و دیگری برای شباهت‌های منفی (نمونه‌هایی با برچسب متفاوت).

```python
class CircleLoss(nn.Module):
    def __init__(self, m: float, gamma: float) -> None:
        super(CircleLoss, self).__init__()
        self.m = m
        self.gamma = gamma
        self.soft_plus = nn.Softplus()

    def forward(self, sp: Tensor, sn: Tensor) -> Tensor:
        ap = torch.clamp_min(- sp.detach() + 1 + self.m, min=0.)
        an = torch.clamp_min(sn.detach() + self.m, min=0.)

        delta_p = 1 - self.m
        delta_n = self.m

        logit_p = - ap * (sp - delta_p) * self.gamma
        logit_n = an * (sn - delta_n) * self.gamma

        loss = self.soft_plus(torch.logsumexp(logit_n, dim=0) + torch.logsumexp(logit_p, dim=0))

        return loss
```

کدی که در شکل بالا مشخص است کد اصلی تابع خطای Circle را تعریف کرده ام . هدف این است که فاصله بین نمونه‌های مشابه یا هم‌کلاس را کم کنیم (یعنی آنها را به هم نزدیک‌تر کنیم) و فاصله بین نمونه‌های نامشابه یا دیگر کلاس‌ها را افزایش دهیم (یعنی آنها را از هم دور کنیم)

در این کد sp نماینگر ویژگی‌های مثبت است که نمونه‌هایی با برچسب یکسان را نشان میدهد و sn نماینگر ویژگی‌های منفی است که نمونه‌هایی با برچسب‌های مختلف را نشان میدهد.

کد را با تابع خطای بالا آموزش دادم برای پیدا کردن پارامتر های این تابع خطا حدس و خطا های مختلف را زدم که در نهایت مقدار m را برابر 0.5 و مقدار gamma را برابر 64 قرار دادم که با توجه به نتایج ، پارامترهای مناسبی به نظر میرسد.

برای طبقه بندی داده های این سوال ، از ایده طبقه بند KNN استفاده کرده ام و مقدار k را برابر 5 قرار داده ام

بری این کار ، ابتدا مدل را روی داده‌های آموزشی CIFAR-10 آموزش داده ام. پس از آموزش، این مدل قادر به استخراج ویژگی‌های قدرتمند از تصاویر است. از آنجا که لایه‌های آخر مدل معمولاً مختص طبقه‌بندی برای مجموعه داده‌ای خاص هستند، لایه‌های طبقه‌بندی را حذف کرده تا مدل فقط بر

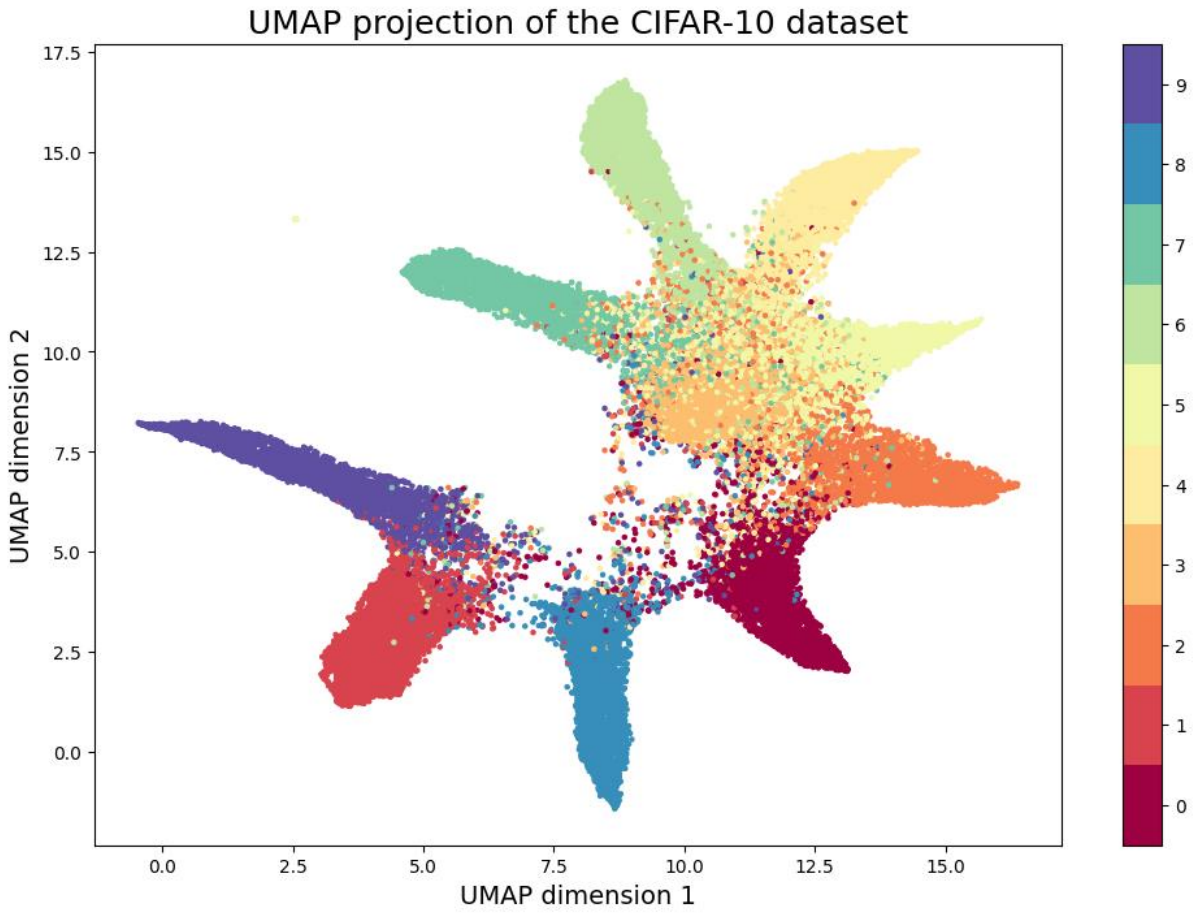
روی بخش استخراج ویژگی تمرکز کند. سپس با استفاده از ویژگی‌های استخراج شده از مجموعه داده آموزشی، یک طبقه بند KNN آموزش داده ام. این اجازه می‌دهد تا بتوانیم الگوهای موجود در فضای ویژگی را تشخیص دهیم و بر اساس آنها تصمیم‌گیری کنیم.

همچنین از تابع بهینه ساز آدام با learning rate برابر 0.001 استفاده کرده ام و مدل را در 25 ایپاک آموزش داده ام که نمودار خطا برای داده های آموزشی و ارزیابی بدین گونه شد .

نمودار تغییرات خطا روی داده های آموزشی و ارزیابی

طبق نمودار هر دو خطای آموزشی و ارزیابی با گذشت زمان کاهش می‌یابند که نشان دهنده‌ی این است که مدل در حال یادگیری و بهبود عملکرد خود بر روی داده‌های آموزشی و ارزیابی است.

نمودار UMAP مربوط به داده های آموزشی مطابق شکل زیر شد :

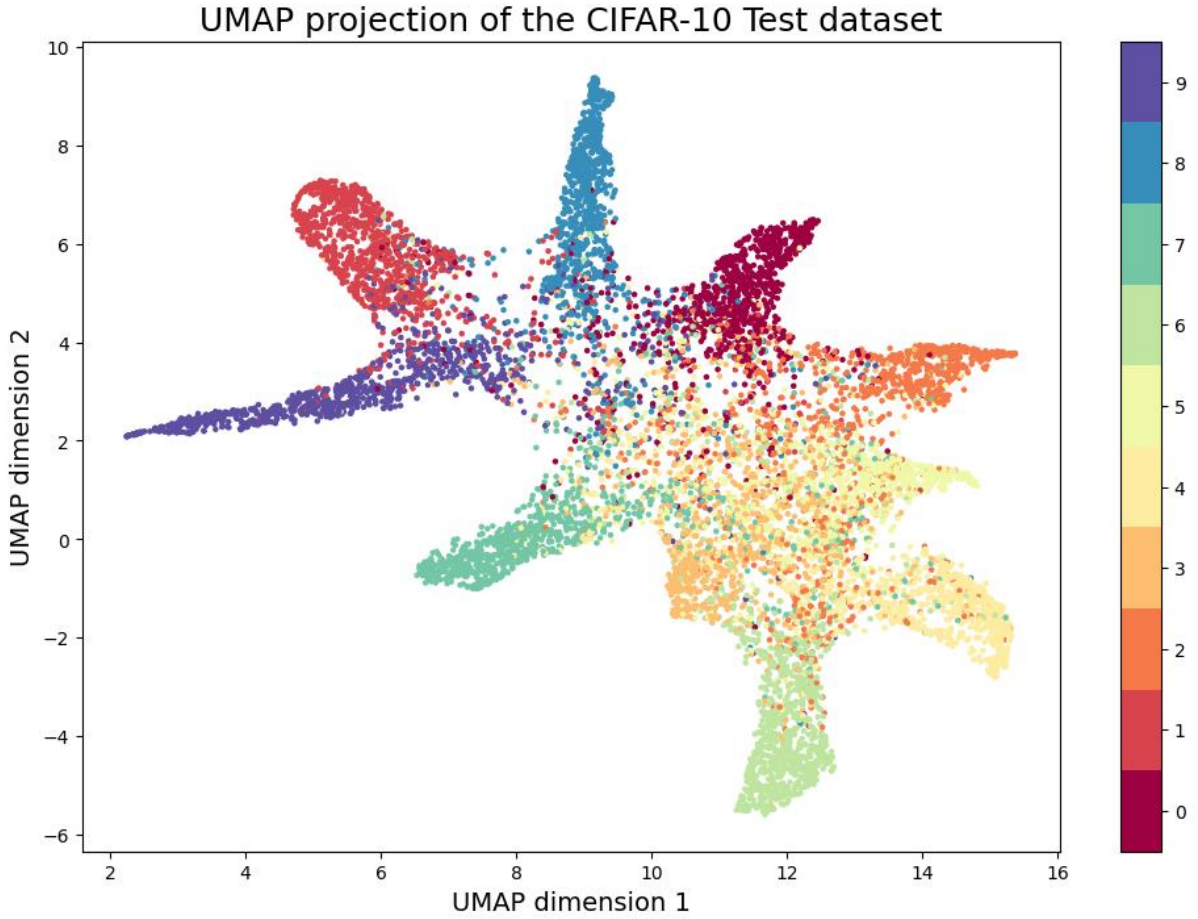


نمودار UMAP داده های آموزش

تفسیر نمودار :

این نمودار یک نمایش دو بعدی از توزیع داده‌های CIFAR-10 است که به وسیله کاهش بعد از فضای ویژگی‌های بالاتر به فضایی با دو بعد انجام شده است. رنگ‌ها نشان‌دهنده کلاس‌های مختلف هستند که بر اساس برچسب‌های اصلی داده‌ها اختصاص یافته‌اند. به نظر می‌رسد برخی از کلاس‌ها به وضوح از یکدیگر جدا شده‌اند که این نشان‌دهنده خوبی از توانایی مدل در جداسازی ویژگی‌های کلاس‌های متفاوت است. مناطقی که رنگ‌ها با یکدیگر تداخل دارند، نشان‌دهنده همپوشانی بین کلاس‌ها هستند. این می‌تواند نشان دهد که برخی نمونه‌ها ویژگی‌های مشابهی دارند که می‌تواند باعث اشتباه در طبقه‌بندی شود.

همچنین نمودار UMAP روی داده های تست به شکل زیر شد.



نمودار UMAP روی داده های تست

تفسیر نمودار بالا :

برخی کلاس‌ها دارای نقاطی هستند که در سراسر فضای UMAP پراکنده شده‌اند، که می‌تواند نشانگر تنوع بیشتر درون کلاسی یا تفاوت‌های جزئی بین نمونه‌ها باشد

در کل، این نمودار UMAP نشان می‌دهد که ویژگی‌های استخراج شده از مدل دارای ساختاری هستند که قادر به نمایش تفکیک‌پذیری کلاس‌های مختلف است، اما همچنین نشان دهنده‌ی این است که در برخی مناطق همچنان تداخل و همپوشانی وجود دارد

مدل را روی داده های تست ارزیابی کرده ام که دقت روی داده های تست 77 درصد شد

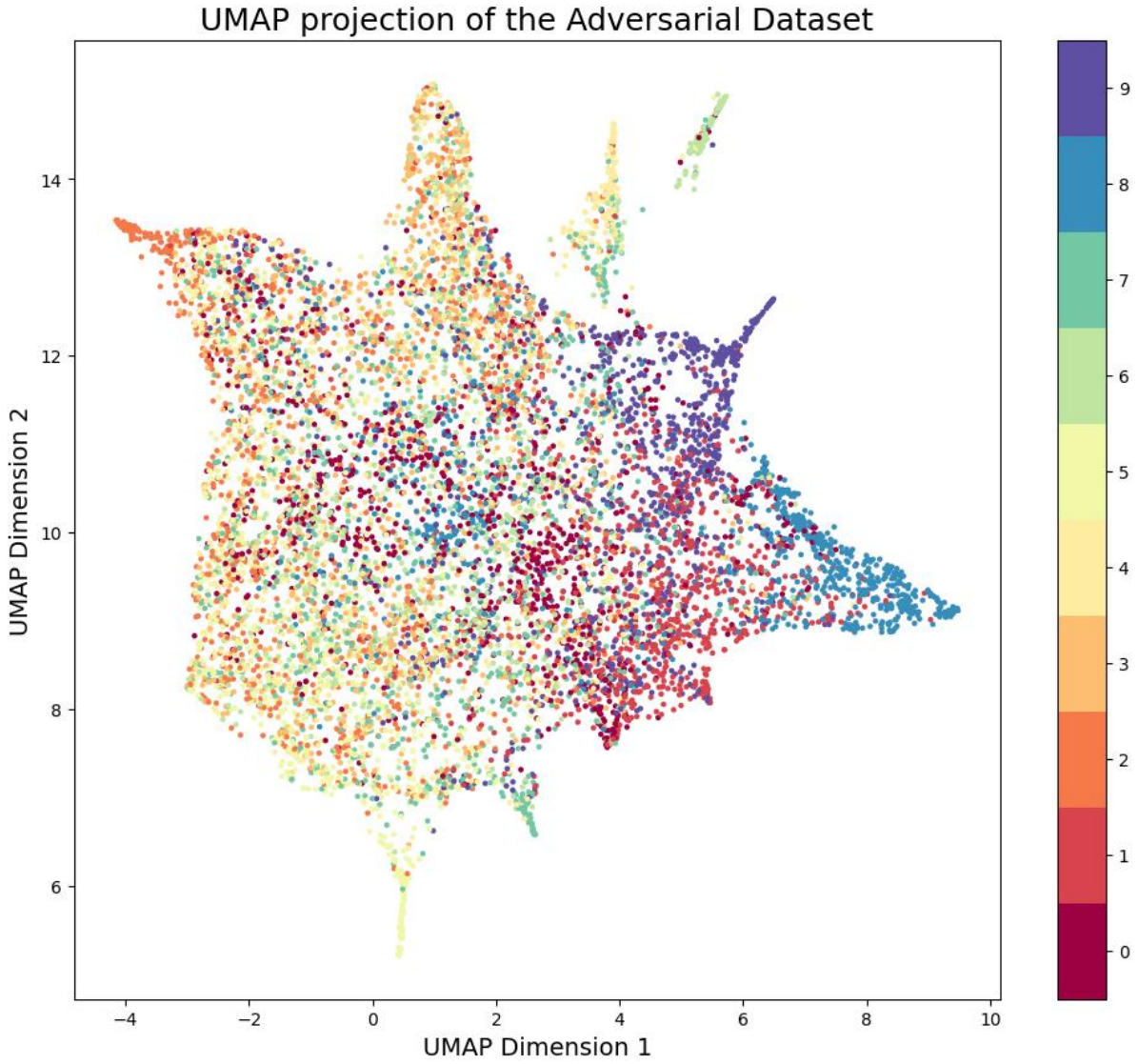
Accuracy of KNN classifier on test set: 77.64%

در ادامه داده های تست CIFAR 10 را با روش FGSM و با پارامتر (اپسیلون) 0.1 به نمونه های Adversarial تبدیل کرده ام همچنین به تصاویر مقداری نویز اضافه کرده ام و سپس مدل را با این داده های جدید ارزیابی کرده که به دقت 52 درصد رسیدم

Accuracy of KNN on adversarial examples: 52.24%

توجه : به دلیل اینکه FGSM از روش گرادیان برای تولید داده های Adversarial استفاده میکند و نیاز به یک تابع خطای مشتق پذیر دارد و Circle loss مشتق پذیر نیست برای تولید نمونه های Adversarial از تابع خطای Cross Entropy استفاده کرده ام ولی برای آموزش مدل از تابع خطای Circle Loss استفاده کرده ام .

نمودار UMAP مربوط به داده های تست Adversarial مطابق زیر شد :



نمودار UMAP داده های تست Adversarial

نقاط در نمودار UMAP بالا به نظر می‌رسد بیشتر پراکنده هستند نسبت به نمودار داده‌های آموزشی یا تست اصلی. این ممکن است نشان دهنده افزایش تنوع یا اغتشاش ایجاد شده توسط حملات متخاصمانه باشد.

برخی از کلاس‌ها همچنان دارای ساختار قابل تشخیص هستند، اما تفکیک بین کلاس‌ها به نظر کمتر مشخص است. این نشان دهنده تأثیر حملات متخاصمانه در بر هم زدن الگوهای ویژگی است که ممکن است موجب کاهش دقت طبقه‌بند شود

مقایسه و تحلیل بین نتایج

خلاصه نتایج هر سه بخش را در جدول زیر آورده ام

	دقت روی داده های تست	دقت روی داده های تست Adversarial همراه با نویز
قسمت الف	69.91	48.41
قسمت ب	82.36	52.23
قسمت ج	77.64	52.24

طبق نتایجی که به دست آمد در حالت ب که ما داده های augmented شده را به مدل دادیم و مدل در حین آموزش داده های Adversarial را مشاهده میکند دقت مناسبی را کسب کردیم همچنین در حالت ج که نوع تابع خطا را عوض کردیم و از circle loss استفاده کردیم در این حالت هم این تابع خطا اثر خود را گذاشته و به دقت مناسبی رسیدیم ولی در حالت الف که از روش خاصی استفاده نکردیم نسبت به دو روش قبلی به دقت کمتری رسیدیم .