به نام خدا

گزارش فاز چهارم

پروژه هوش محاسباتی

پوریا ناظمی و طهورا سعیدی

بخش اول (طراحی معماری مدل شبکه عصبی)

به دنبال طراحی شبکه ای هستیم که با عبور تصاویر از لایه های پیچشی به یک بازنمایی مناسب از آنها برسیم و در کنار بردار ویژگی داده شده از آنها، اقدام به طبقه بندی اعداد کنیم. معماری شبکه پیچشی اول بایستی چند لایه را شامل شود تا به وسیله آنها، ویژگی های تصاویر استخراج شوند. دیتاست ما شاما دیتاست های معروف USPSو SYNthetic digits (SYN) ، SVHN، MNIST-M، MNIST می باشند با این تفاوت که همه تصاویر 32 در 32 و 3 چنله (RGB) شده اند.

با استفاده از تلاش های دیگری که از قبل روی این دیتاست ها شده، میتوان از معماری های استفاده شده، الگوبرداری کرد. معماری در نظر گرفته شده برای این دیتابیس را باید بر این اساس تنظیم کنیم که تعداد فیچر های تولیدی در خروجی لایه های پیچشی باید با تعداد داده ها و فیچر های آنها مناسب باشد. همچنین باید در نظر گرفت که بعد از لایه های پیچشی، بازنمایی ایجاد شده همراه با بردار ویژگی ها باید ادغام شوند. نکته ای که وجود دارد اگر مستقیما این دو دیتا را به لایه های خروجی وصل کنیم و لایه های دیگری در این مابین نباشد، مدل نمیتواند در ترکیب این داده ها عملکرد خوبی نشان دهد، پس یک لایه دیگر نیز قبل از لایه خروجی قرار میدهیم تا وزن ها در ترکیب این دیتاها دخیل شوند و نتیجه ترکیب بهتری بدست آید.

در ابتدا مدلی ساده با ساختار زیر را ایجاد کردیم و نتایج بدست آمده جای بیشتر شدن داشت

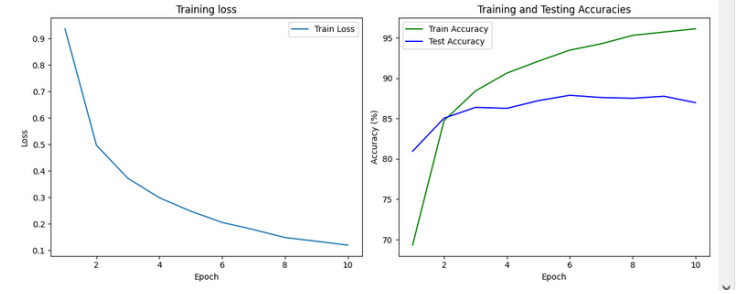
```
class DigitCNNModel(nn.Module):
    def __init__(self, num_of_features, num_classes=10):
        super(DigitCNNModel, self).__init__()

        self.conv_layers = nn.Sequential(
            nn.Conv2d(3, 8, kernel_size=3, stride=1, padding=1),
            nn.ReLU(inplace=True),
        )

        self.fc = nn.Linear(8 * 32 * 32 + num_of_features, 128)
        self.activation = nn.ReLU()
        self.fc2 = nn.Linear(128, 256)
        self.activation = nn.ReLU()
        self.output_layer = nn.Linear(256, num_classes)

    def forward(self, images, features):
```

```
Epoch 1/10 - Train Loss: 0.9363, Train Accuracy: 69.32%, Test Accuracy: 80.95%
Epoch 2/10 - Train Loss: 0.4968, Train Accuracy: 84.75%, Test Accuracy: 85.05%
Epoch 3/10 - Train Loss: 0.3720, Train Accuracy: 88.42%, Test Accuracy: 86.38%
Epoch 4/10 - Train Loss: 0.2990, Train Accuracy: 90.64%, Test Accuracy: 86.27%
Epoch 5/10 - Train Loss: 0.2480, Train Accuracy: 92.11%, Test Accuracy: 87.22%
Epoch 6/10 - Train Loss: 0.2056, Train Accuracy: 93.48%, Test Accuracy: 87.89%
Epoch 7/10 - Train Loss: 0.1786, Train Accuracy: 94.27%, Test Accuracy: 87.61%
Epoch 8/10 - Train Loss: 0.1484, Train Accuracy: 95.31%, Test Accuracy: 87.51%
Epoch 9/10 - Train Loss: 0.1346, Train Accuracy: 95.72%, Test Accuracy: 87.76%
Epoch 10/10 - Train Loss: 0.1201, Train Accuracy: 96.13%, Test Accuracy: 86.97%
```

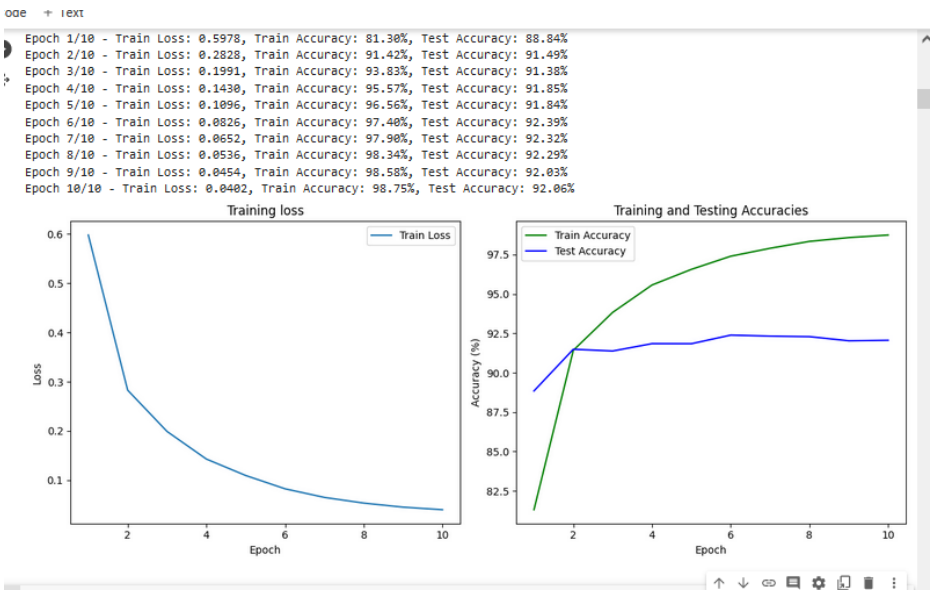


مدل با ساختار زیر را امتحان کردیم:

```python
class DigitCNNModel(nn.Module):
    def __init__(self, num_of_features, num_classes=10):
        super(DigitCNNModel, self).__init__()

        self.conv_layers = nn.Sequential(
            nn.Conv2d(3, 32, kernel_size=3, stride=1, padding=1),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2, stride=2),
            nn.Conv2d(32, 64, kernel_size=3, stride=1, padding=1),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2, stride=2),
        )

        self.fc = nn.Linear(64 * 8 * 8 + num_of_features, 128)
        self.activation = nn.ReLU()
        self.output_layer = nn.Linear(128, num_classes)
```



نتیجه به این شکل شد که خوب است اما 3 لایه پیچشی را نیز تست کردیم.

معماری مدل پایه ای که در حالت خام عملکرد نسبتا خوبی داشت، به شرح زیر است:

```
self.conv_layers = nn.Sequential(
    nn.Conv2d(3, 16, kernel_size=3, stride=1, padding=1),
    nn.ReLU(inplace=True),
    nn.MaxPool2d(kernel_size=2, stride=2),
    nn.Conv2d(16, 32, kernel_size=3, stride=1, padding=1),
    nn.ReLU(inplace=True),
    nn.MaxPool2d(kernel_size=2, stride=2),
    nn.Conv2d(32, 64, kernel_size=3, stride=1, padding=1),
    nn.ReLU(inplace=True),
    nn.MaxPool2d(kernel_size=2, stride=2)
)

self.fc = nn.Linear(64 * 4 * 4 + num_of_features, 256)
self.activation = nn.ReLU()
self.output_layer = nn.Linear(256, num_classes)
```

سه لایه پیچشی همراه با مکس پولینگ و دو لایه فولی کانکتد

لایه پیچشی اول فیچر های اولیه سطح صفر را از تصاویر را دریافت میکند، این لایه از 16 فیلتر 3*3 استفاده میکند و فیچر های ساده را استخراج میکند.

لایه پیچشی دوم 32 فیلتر 3*3 دارد و فیچر های پیچیده تر را استخراج میکند و لایه پیچشی سوم 64 فیلتر 3*3 دارد مه فیچر های سطح بالا را استخراج میکند.

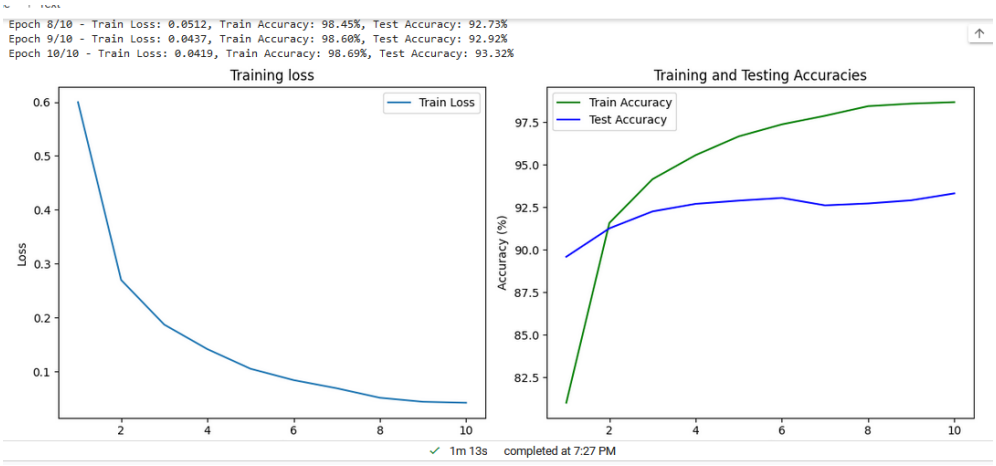
لایه های max pooling کمک میکند ابعاد خروجی کاهش یابد اما در عین حال فیچر های مهم حفظ شوند، بعد از هر لایه های پیچشی یک max pool با اندازه 2*2 داریم که در واقع ابعاد را نصف میکند.

اکتیویشن فانکشن استفاده شده relu می باشد که مرسوم ترین است، اما در ادامه اکتیویشن فانکشن های دیگر را نیز تست میکنیم.

لایه فولی کانکتد اول ورودی حاصله از لایه های پیچشی را دریافت میکند و همراه با بردار های ویژگی، بردار حاصل برای طبقه بندی را سعی میکند تشکیل دهد. خروجی این لایه را به لایه فولی کانکتد output که 10 نورون دارد میدهیم تا عمل تشخیص 10 دیجیت صورت گیرد.

نکاتی که در طراحی شیکه وجود دارد این است که افزایش لایه ها و فیلتر ها سبب افزایش performance مدل حداقل بر رئی داده های ترین می شود. اما از جهتی ممکن است overfit رخ دهد یا حتی اگر دقت روی داده های تست نیز خوب باشد، هزینه محاسباتی زیادی را به دنبال دارد که باعث یک trade off می شود.

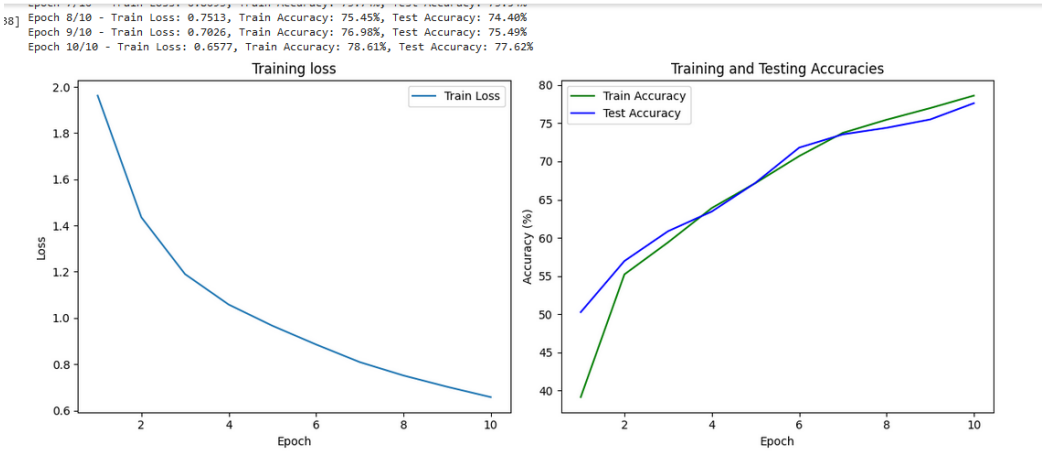
این مدل خام را بدون هیچ تعیین پارامتر خاصی بر روی داده ها اجرا می کنیم:



مدل اولیه دقت خوبی دارد، اما حال به سراغ تعیین پارامتر و هایپر تیون کردن آن می رویم.

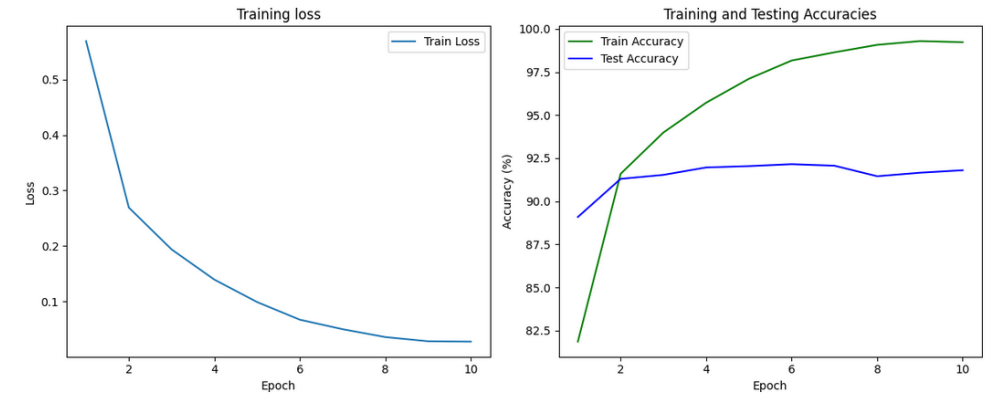
در ابتدا سعی میکنیم اکتیویشن فانکشن مناسب را بیابیم. به این منظور میدانیم که اکتیویشن فانکشن sigmoid به دلیل بار محاسباتی برای شیکه های cnn مناسب نیستند، به سراغ soft_max وtanh میرویم.

نتایج soft_max به این شکل شد:



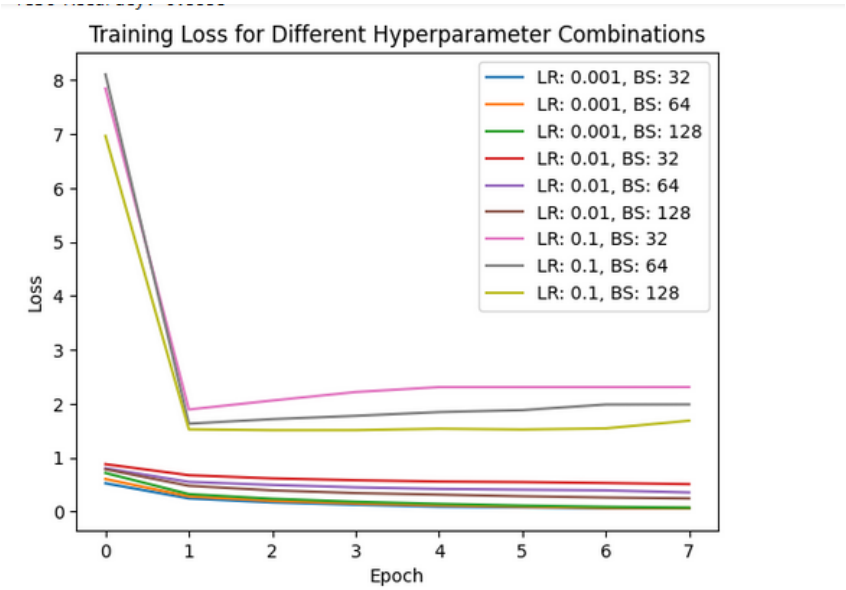
دقت به مقدار زیادی کاهش یافت پس مناسب نیست. به سراع اکتیویشن فانکشن tanh میرویم:

```
Epoch 8/10 - Train Loss: 0.0356, Train Accuracy: 99.08%, Test Accuracy: 91.45%
Epoch 9/10 - Train Loss: 0.0281, Train Accuracy: 99.30%, Test Accuracy: 91.66%
Epoch 10/10 - Train Loss: 0.0274, Train Accuracy: 99.23%, Test Accuracy: 91.80%
```



نتیجه مناسب است اما باز هم به نسبت relu ضعیف تر است. پس همان relu را برمیگزینیم.

حال به دنبال تیون کردن batch و lr میرویم.

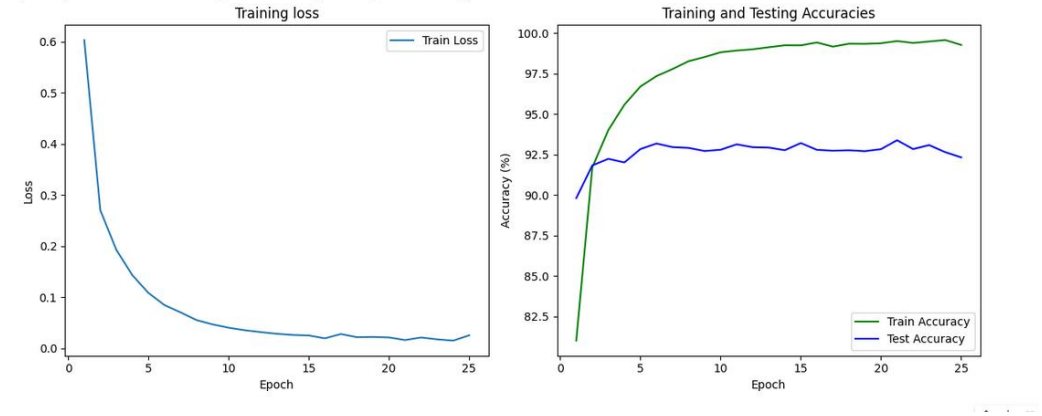



```
Best Test Accuracy: 0.9318
Best Learning Rate: 0.001, Best Batch Size: 32
```

تعداد بچ ها 32 و اندازه مناسب learning rate 0.001 می باشد. این مقادیر را در مدلسازی های بعدی لحاظ میکنیم.

حال به دنبال تعدادepoch مناسب میگردیم. مدل را با epoch 25 اجرا میکنیم. نتیجه accuracy به این شکل است:

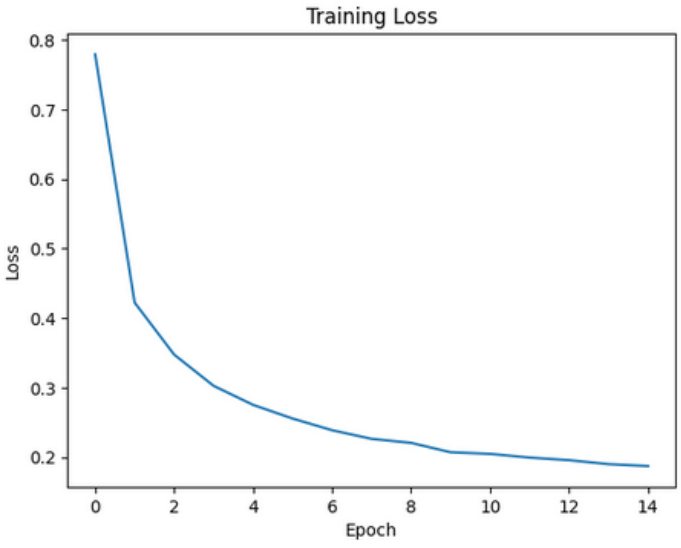
```
Epoch 23/25 - Train Loss: 0.0176, Train Accuracy: 99.48%, Test Accuracy: 93.08%
Epoch 24/25 - Train Loss: 0.0151, Train Accuracy: 99.57%, Test Accuracy: 92.65%
Epoch 25/25 - Train Loss: 0.0255, Train Accuracy: 99.27%, Test Accuracy: 92.32%
```



در epoch،15 بهترین دقت را داشتیم پس از این پس تعداد epoch را 15 در نظر میگیریم.

در ادامه سراغ بهتر کردن عملکرد مدل رفته و dropout و regularization قرار میدهیم. برای l2 دو مقدار 0.01 و 0.001 در نظر گرفتیم و با مقدار 0.001 نتیجه بهتری حاصل شد و در کل دقت مدل افزایش یافت:

```
Epoch 7/15 - Loss: 0.2387 - Train Accuracy: 0.9450 - Test Accuracy: 0.9288
Epoch 8/15 - Loss: 0.2264 - Train Accuracy: 0.9494 - Test Accuracy: 0.9315
Epoch 9/15 - Loss: 0.2206 - Train Accuracy: 0.9536 - Test Accuracy: 0.9345
Epoch 10/15 - Loss: 0.2071 - Train Accuracy: 0.9582 - Test Accuracy: 0.9344
Epoch 11/15 - Loss: 0.2048 - Train Accuracy: 0.9600 - Test Accuracy: 0.9383
Epoch 12/15 - Loss: 0.1994 - Train Accuracy: 0.9626 - Test Accuracy: 0.9386
Epoch 13/15 - Loss: 0.1957 - Train Accuracy: 0.9639 - Test Accuracy: 0.9387
Epoch 14/15 - Loss: 0.1900 - Train Accuracy: 0.9667 - Test Accuracy: 0.9390
Epoch 15/15 - Loss: 0.1872 - Train Accuracy: 0.9687 - Test Accuracy: 0.9411
```



با دقت 94 درصد این بخش را به پایان میرسانیم.

بخش دوم (پیاده سازی تابع خطا triplet_loss)

تنها لازم است تابع forward این کلاس را پیاده سازی کنیم. پیاده سازی آن به شکل زیر است:

```
class TripletLoss(nn.Module):
    def __init__(self, margin: float = 0.05):
        device = torch.device("cuda:0" if torch.cuda.is_available() and use_gpu else "cpu")
        super().__init__()
        self.margin = torch.tensor(margin)
        self.device = device

    def forward(self, embeddings, labels):
        dp, dn = self.batch_hard_triplet_loss(embeddings, labels)

        triplet_loss = torch.mean(torch.clamp(dp - dn + self.margin, min=0))
        return triplet_loss
```

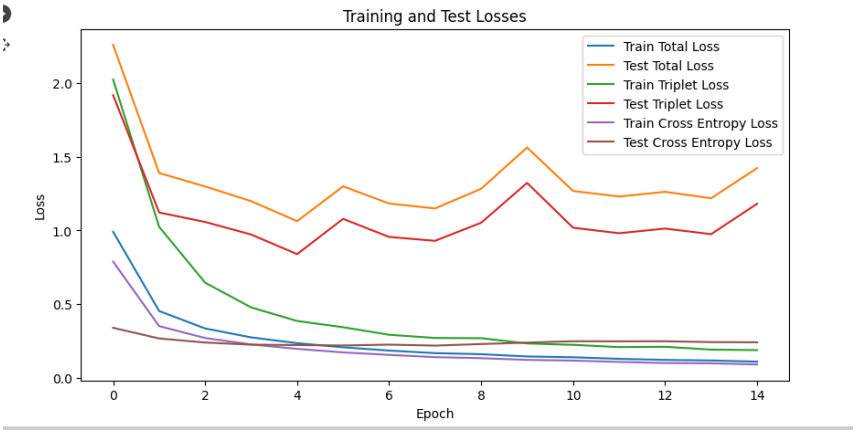
در واقع بین صفر و تفاوت دورترین دیتاهای با یک لیبل و نزدیکترین دیتاهای با دو لیبل متفاوت، به علاوه margin ماکسیمم میگیریم.

margin مربوط به خود تریپلت لاس است. اگر صفر باشد،در این صورت، برای هر داده ماکسیمم بین فاصله مثبت منهای فاصله منفی و همچنین صفر را محاسبه کنیم. پس اگر فاصله مثبت حتی یک اپسیلون از فاصله منفی کمتر باشد، لاس صفر شده و هیچ آپدیتی صورت نمیگیرد.
حال فرض کنید مقدار alpha را برابر 1 دهم قرار داده ایم، این یعنی: فاصله مثبت باید از فاصله منفی بیشتر از یک دهم کمتر باشد تا آپدیتی صورت بپیرد! و شبکه این term را انتخاب و سعی در کم کردن آن خواهد داشت. پس مقدار مارجین نباید نه خیلی کم باشد مثل صفر (که حاشیه مثبت و منفی تقریبا هیچی میشود) و نه باید خیلی بیشتر شود.

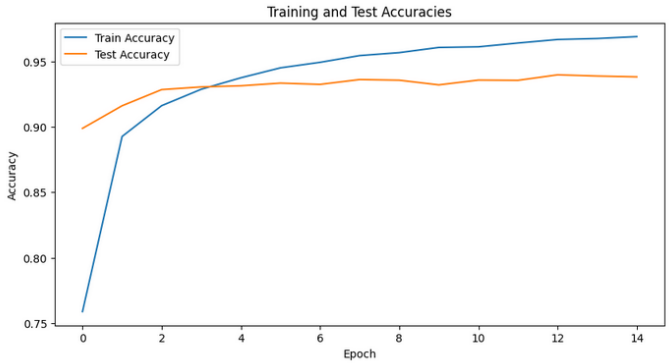
متغیر دیگر لاندا است که میزان تاثیر triplet_lost بر روی loss نهایی را تعیین میکند. نکته ای که وجود دارد این است که triplet_loss با توجه به لیبل دامین است و روی لایه آخر اجرا نمیشود.

مدل بیسی را با اضافه کردن این تابع خطا اجرا میکنیم و مقادیر margin وlanda را رندوم قرار میدهیم.

خروجی این مدل و بررسی روند loss ها به شکل زیر است:



دقت مدل با ترکیب دو تابع خطا:



به راحتی به دقت 94 -93 درصد رسیده ایم بدون regularization

تابع خطا که بر روی داده های ترین optimize می شود به خوبی روند کاهشی خود را دنبال میکند و پیاده سازی تاثیر آن درست است.

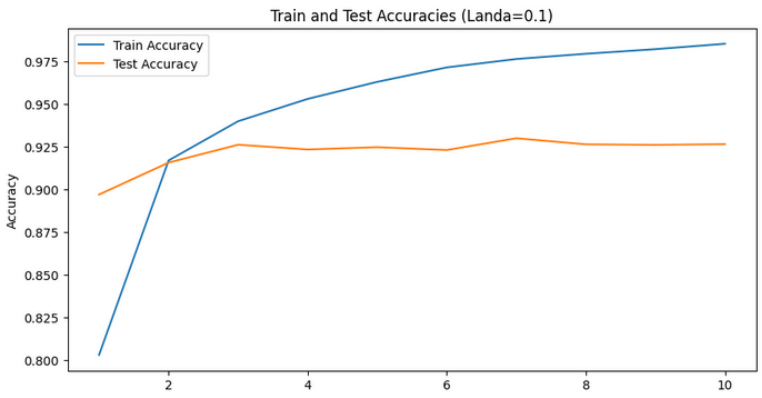
حال باید دو پارامتر landa و margin را tune کنیم، نکته ای که وجود دارد این اسن که دو عامل تاثیر متقابل ندارند و مستقل از هم اند و میتوان هر یک را جداگانه tune کرد و نتیجه را ادغام کرد.

ابتدا به سراغ tune کردن landa میرویم. مجموعه مقادیر زیر را تعریف میکنیم و بهترین را میبابیم.

```
margin = 0.01
landa_values = [0.1, 0.01, 0.001, 0.5, 0.05, 0.9, 0.009]
results = []
```

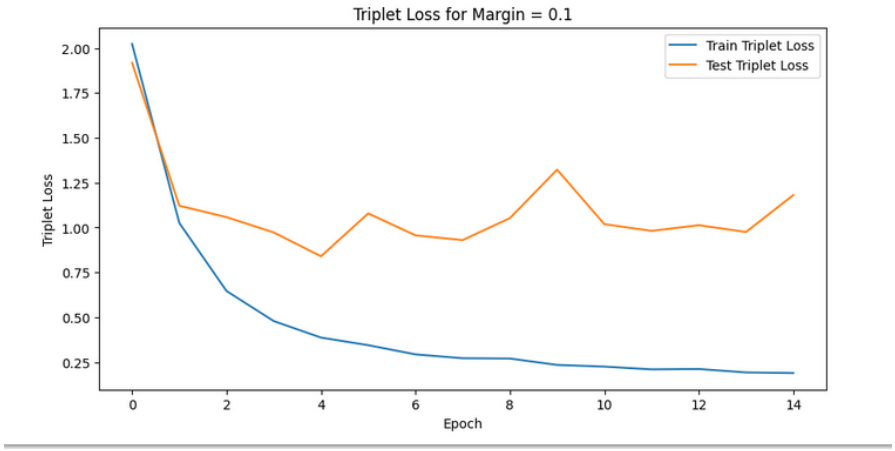
مقدار بهینه بدست آمده نهایی، 0.1 می باشد:

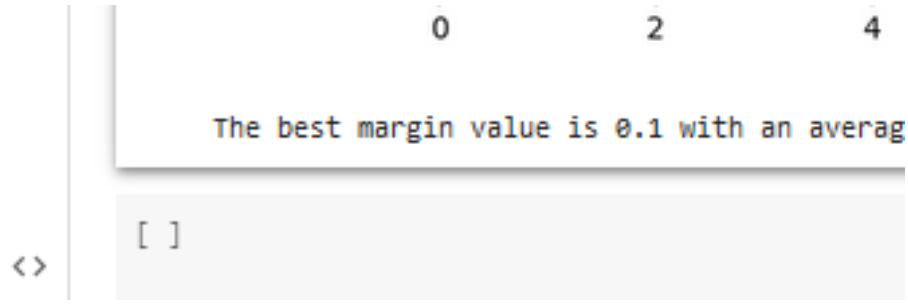
```
Best Landa: 0.1
Best Test Accuracy: 0.9264
```



حال به سراغ tune کردن margin میرویم. مجموعه مقادیر زیر را در نظر میگیریم و بهترین را میبابیم.

```
landa = 0.1
margin_values = [0.1, 0.01, 0.001, 0.0001, 0.005, 0.05]
results = []
```

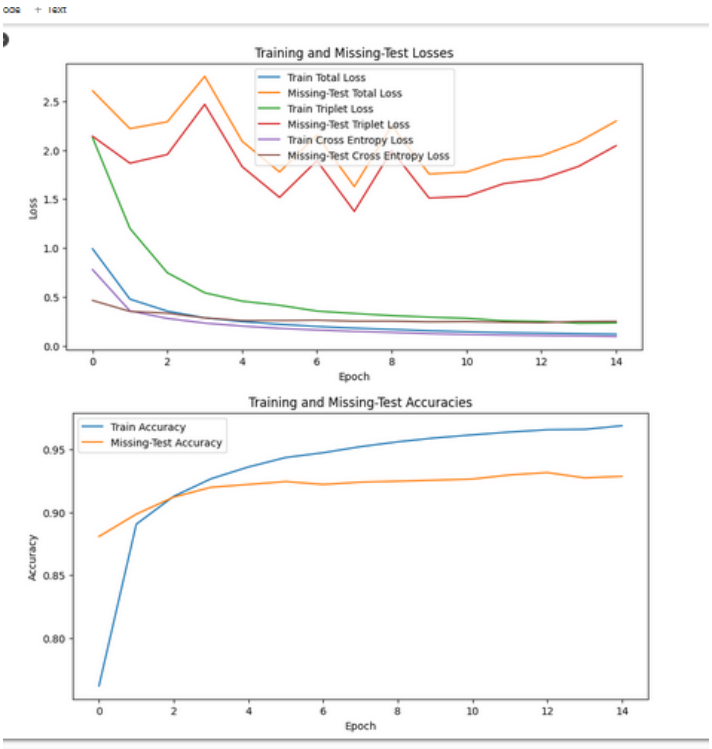




مقدار بهینه با توجه به میانگین میزان loss بدست می آید.

بخش نمره اضافه (داده های تست بدون بردار ویژگی)

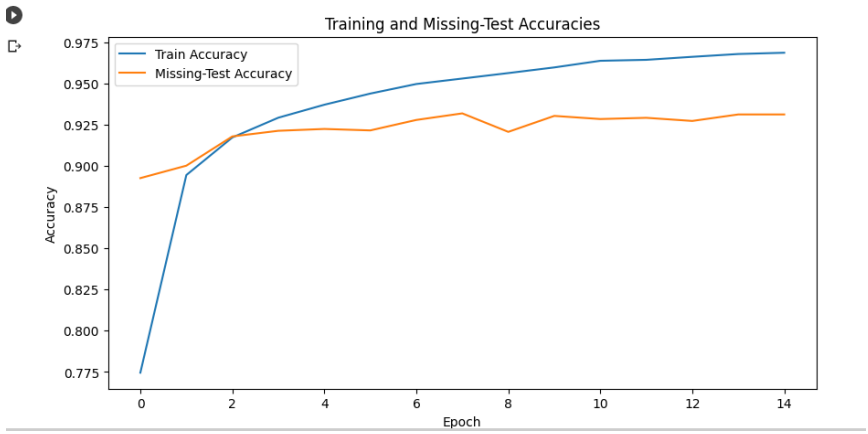
در ابتدا مدل مرحله قبل رو بر روی داده ها اجرا میکنیم تا عملکرد اولیه را بسنجیم.



نتایج حاصله دقت 92 درصدی برای test_missing_dataرا نشان میدهد که کاهش عملکرد به دلیل صفر شدن بردار ویژگی ها است. حال سعی میکنیم مدل را بهبود ببخشیم. اولین راهکاری که سراغ آن می رویم، data agumentation است که به نحوی با افزایش فیچر ها تاثیر نبود بردار ویژگی را کم کنیم. داده ها را با ابزار های زیر transform میکنیم:

```
# Data augmentation transforms
transform = transforms.Compose([
    transforms.RandomHorizontalFlip(p=1),
    transforms.RandomRotation(10, fill=(0, 0, 0)),
    transforms.ToTensor(),
    transforms.Normalize(mean=(0.5, 0.5, 0.5), std=(0.5, 0.5, 0.5))
])
```

نتیجه حاصله دقت مدل را تا حدودی افزایش داد و به عملکرد بهتری رسیدیم:



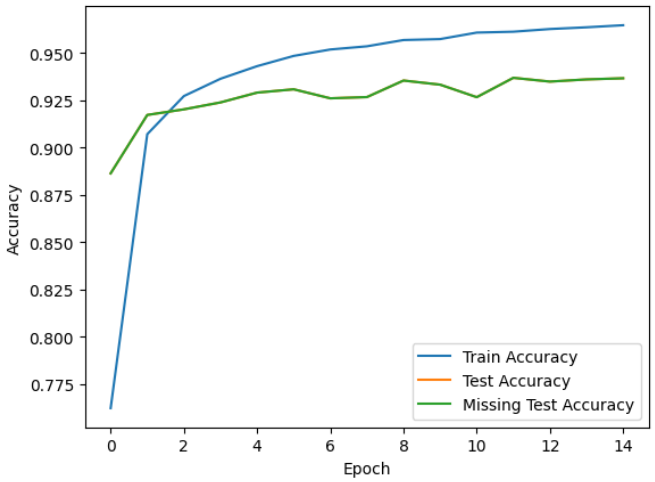

```
Epoch 15/15
Train Loss: 0.1141 - Missing-Test Loss: 3.7917
Train Accuracy: 0.9686 - Missing-Test Accuracy: 0.9312
Train Triplet Loss: 0.2133 - Missing-Test Triplet Loss: 3.5531
Train Cross Entropy Loss: 0.0928 - Missing-Test Cross Entropy Loss: 0.2386
Train Total Loss: 0.1141 - Missing-Test Total Loss: 3.7917
```

تغییر معماری شبکه را نیز امتحان میکنیم. شبکه ای با ساختار زیر میسازیم:

```
self.conv_layers = nn.Sequential(
            nn.Conv2d(3, 32, kernel_size=3, stride=1, padding=1),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2, stride=2),
            nn.Conv2d(32, 64, kernel_size=3, stride=1, padding=1),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2, stride=2),
            nn.Conv2d(64, 128, kernel_size=3, stride=1, padding=1),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2, stride=2)
        )

self.fc = nn.Linear(128 * 4 * 4, 256)
self.dropout = nn.Dropout(0.5)
self.output_layer = nn.Linear(256, num_classes)
```

نتیجه حاصله از این شبکه به شکل زیر است:




```
Missing Test Total Loss: 0.732208
----------------------------------------------------------------
Epoch: 15/15
Train Loss: 0.140439 | Train Accuracy: 0.9647
Train Triplet Loss: 0.229455 | Train Cross-Entropy Loss: 0.117493
Test Accuracy: 0.9367
Test Triplet Loss: 0.562093 | Test Cross-Entropy Loss: 0.203598
Test Total Loss: 0.765691
Missing Test Accuracy: 0.9367
Missing Test Triplet Loss: 0.566851 | Missing Test Cross-Entropy Loss: 0.203592
Missing Test Total Loss: 0.770443
----------------------------------------------------------------
```

با توجه به وجود 256 فیچر برابر با صفر، به نتیجه خوبی در هر دو روش رسیدیم.