به نام خدا

دانشگاه تهران

دانشکده مهندسی برق و کامپیوتر

درس شبکه‌های عصبی و یادگیری عمیق

تمرین اول- سوال ۳

نام و نام خانوادگی	بهیاد زرنقی – امیرحسین محمدزاده
شماره دانشجویی	۸۱۰۶۹۸۲۴۹ - ۸۱۰۶۹۸۳۰۲
تاریخ ارسال گزارش	۱۴۰۲.۰۱.۰۷

فهرست

شکل‌ها

پاسخ ۳ – Auto-Encoders for classification

۳-۱. پیش پردازش داده‌ها

دیتای مربوط به این بخش با استفاده ازکتابخانه‌ی Pytorch فراخوانی خواهد شد. برای انجام این مهم کد بلوک زیر ران شده است.

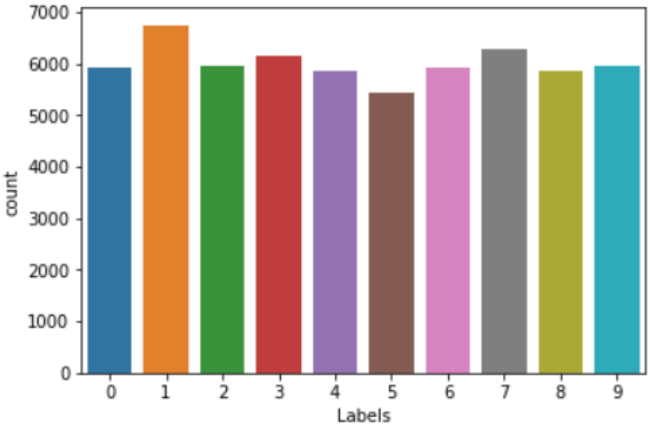
```
train_loader = datasets.MNIST(root="data", train=True, download=True, transform=transforms.Compose([transforms.ToTensor(),transforms.Normalize(0,1)]))
train_data = torch.utils.data.DataLoader(dataset=train_loader,
                                         batch_size=100,
                                         shuffle=True)


test_loader = datasets.MNIST(root="data", train=False, download=True, transform=transforms.Compose([transforms.ToTensor(),transforms.Normalize(0,1)]))
test_data = torch.utils.data.DataLoader(dataset=test_loader,
                                         batch_size=100,
                                         shuffle=True)
```

. **شکل ۱.** بارگیری دیتای مجموعه‌ی MNIST

.

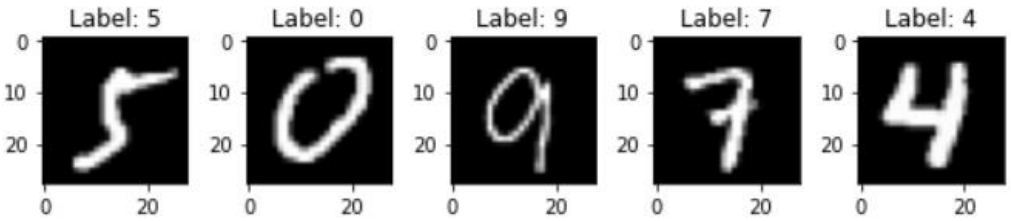
نکته‌ی حائز اهمیت در این بخش این است که فرآیند مربوط به Normalization (نرمالیزه کردن) در همین بخش بارگیری داده‌ها، به وسیله‌ی متد transforms.Normalize انجام گرفته است. همچنین هر دو داده‌ی train و test به صورت batch (دسته) های ۱۰۰ تایی نهایتاً لود خواهد شد تا به این طریق بتوان به صورت موازی فرآیند آموزش داده‌ها را انجام داد و آن را بهینه کرد و در نهایت حدالامکان از زمان آموزش شبکه کاست.

در ادامه نمودار توزیع تعداد هر گروه در داده‌های train به صورت زیر رسم شده است.



شکل ۲. نمودار تعداد بر حسب گروه در داده‌های train

همچنین ۵ تصویر به صورت رندوم از دیتاست را به همراه برچسب‌هایش با استفاده از plt.imshow() نشان می‌دهیم.



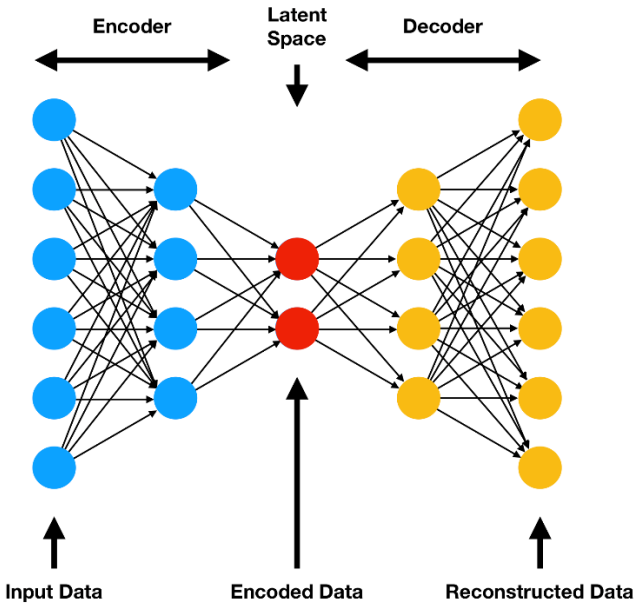
شکل ۳. پنج داده‌ی رندوم از مجموعه‌ی **MNIST**

۳-۲. طراحی و آموزش شبکه Auto-Encoder

۳-۲-۱. ساختار Auto Encoder

Auto Encoder (خودرمزگذار) شامل یک encoder (رمزگذار) و decoder (رمزگشا) به همراه لایه پنهان می‌باشد. ورودی به encoder داده شده و خروجی از decoder استخراج می‌شود. در این نوع شبکه به جای آموزش شبکه و پیش‌بینی مقدار تابع هدف در ازای ورودی، خودرمزگذار آموزش می‌بیند که ورودی خود را بازسازی کند؛ بنابراین بردار خروجی همان ابعاد بردار ورودی را خواهد داشت؛ یعنی تعداد نورون‌های موجود در لایه‌ی ورودی و خروجی با یکدیگر برابر است. لایه وسط این شبکه که bottle neck نام دارد نیز کمترین تعداد نورون را خواهد داشت و حاوی دیتای encoded (رمزگذاری شده) خواهد بود.

این نوع از شبکه‌ها در مواردی همچون فشرده‌سازی تصاویر و حذف نویز کاربرد گسترده‌ای دارند.



شکل ۴. ساختار شبکه‌ی **Auto – Encoder**

۳-۲-۲. طراحی و مدل Auto Encoder

همانطور که در بخش قبلی مطرح شد، Auto Encoder شامل دو بخش encoder و decoder می‌باشد. معماری Auto Encoder مفروض مسئله به صورت زیر می‌باشد.

معماری	
Input: 784 FC: 500 FC: 200 FC: 100 Output: 30	Encoder
Input: 30 FC: 100 FC: 200 FC: 500 Output: 784	Decoder

شکل ۵. معماری شبکه

برای پیاده‌سازی این ساختار در پایتون، یک class تعریف می‌کنیم و داخل آن با استفاده از ماژول Pytorch تعداد نورون‌های هرلایه‌ی شبکه را به تعداد مشخص شده تعیین می‌کنیم. این کار را یک بار برای بخش encoder و بار دیگر برای بخش decoder طراحی می‌کنیم. مشخصاً روند توالی لایه‌ها در encoder و decoder قرینه‌ی یکدیگر می‌باشد. برای جلوگیری از پدید آمدن vanishing gradient problem ، بین لایه ها از تابع فعالساز Relu استفاده شده است. چراکه مشتق این تابع به ازای مقادیر مثبت برابر با یک است و در تعداد لایه‌های بالا از رخداد مشکل یاد شده جلوگیری خواهد کرد. نکته‌ی حائز اهمیت دیگر در این بخش، لزوم استفاده از تابع فعالسازی Sigmoid در لایه‌ی آخر decoder است. چراکه مقادیر پیکسل های تصویرهای خروجی بایستی همچون تصاویر ورودی بین ۰ تا ۱ باشند.

```python
class Autoencoder(nn.Module):
    def __init__(self):
        super().__init__()
        self.encoder = nn.Sequential(
            nn.Linear(28 * 28, 500),
            nn.ReLU(),
            nn.Linear(500, 200),
            nn.ReLU(),
            nn.Linear(200, 100),
            nn.ReLU(),
            nn.Linear(100, 30)
        )

        self.decoder = nn.Sequential(
            nn.Linear(30, 100),
            nn.ReLU(),
            nn.Linear(100, 200),
            nn.ReLU(),
            nn.Linear(200, 500),
            nn.ReLU(),
            nn.Linear(500, 28 * 28),
            nn.Sigmoid()
        )


    def forward(self, x):
        encoded = self.encoder(x)
        decoded = self.decoder(encoded)
        return decoded
```

شکل ۶. معماری شبکه‌ی **Auto – Encoder** در پایتون

سپس با تعیین مؤلفه‌های مربوط به Loss و Optimizaion، مدل را تعریف می‌کنیم. برای پارامتر loss از روش MSE و برای بهینه سازی نیز از روش Adam استفاده شده است. در این بخش مقدار learning rate که نشان دهنده سرعت آموزش مدل با استفاده از روش است، برابر با 10^{-3} است. گرچه با توجه به استفاده‌ی ما از Adam ، مقدار این مولفه در طول آموزش ثابت نخواهد ماند. همچنین مقدار مولفه‌ی weight_decay که به عنوان پارامتری برای Regularization می‌باشد، برابر با 10^{-5} قرار داده شده است.

```python
model = Autoencoder()

criterion = nn.MSELoss()
optimizer = torch.optim.Adam(model.parameters(),
                             lr=1e-3,
                             weight_decay=1e-5)
```

شکل ۷. تعریف مولفه‌های مربوط به loss و optimization مدل

۳-۲-۲. آموزش شبکه

بعد اتمام تعریف مدل، اکنون با استفاده از داده‌های train در ۱۰ epoch (مرحله) مدل خود را آموزش خواهیم داد. در وهله‌ی اول باید به این نکته توجه نماییم که دیتای خام مربوط به تصاویر، به صورت ماتریس‌های ۲۸*۲۸ می‌باشند. بنابرین لازم است که با یک تابع reshape دیتای مربوط به تصاویر را به

بردار یک بعدی به اندازه‌ی تعداد پیکسل‌ها در هر تصویر یعنی ۷۸۴ تبدیل کرد. بعد دادن دیتای مذکور به مدل، خروجی را ذخیره می‌کنیم و loss بین ورودی و خروجی را بدست می‌آوریم و با استفاده از متدهای optimizer.zero_grad، loss.backward و optimizer.step مدل را آموزش می‌دهیم.

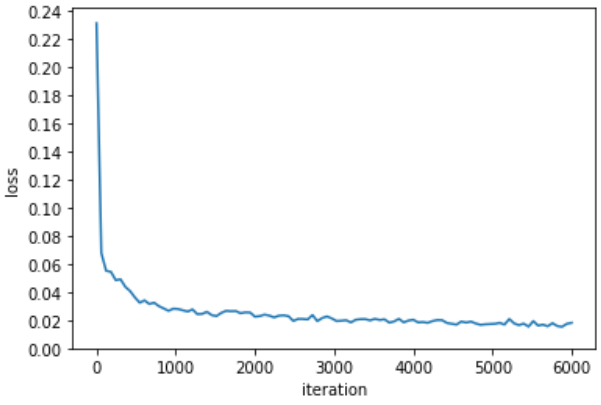
وضعیت loss در هر epoch به صورت زیر خواهد بود.

```
Epoch:1, Loss:0.0352
Epoch:2, Loss:0.0276
Epoch:3, Loss:0.0233
Epoch:4, Loss:0.0222
Epoch:5, Loss:0.0210
Epoch:6, Loss:0.0199
Epoch:7, Loss:0.0191
Epoch:8, Loss:0.0173
Epoch:9, Loss:0.0169
Epoch:10, Loss:0.0165
```

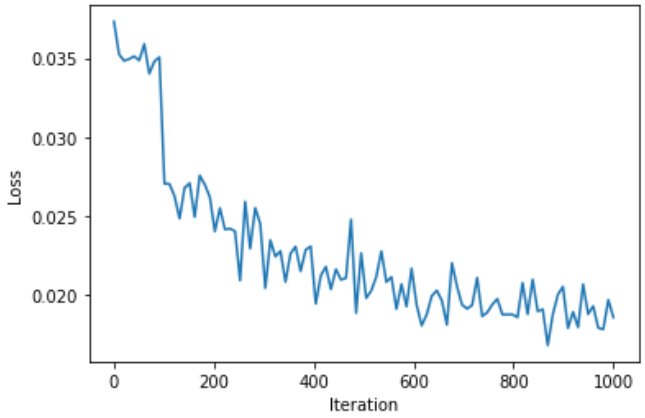
شکل ۸. Loss در هر epoch

همانطور که انتظار می‌رفت میزان loss در هر مرحله با توجه به آپدیت شدن وزن‌ها کاهش می‌یابد و به مقدار قابل قبولی می‌رسد. بنابرین نیازی به آموزش مدل افزون بر ده مرحله‌ی انجام پذیرفته نیست.

با رسم نمودار مربوط به loss در هر iteration آموزش، می‌توان روند آن را با دقت بیشتری مشاهده کرد.



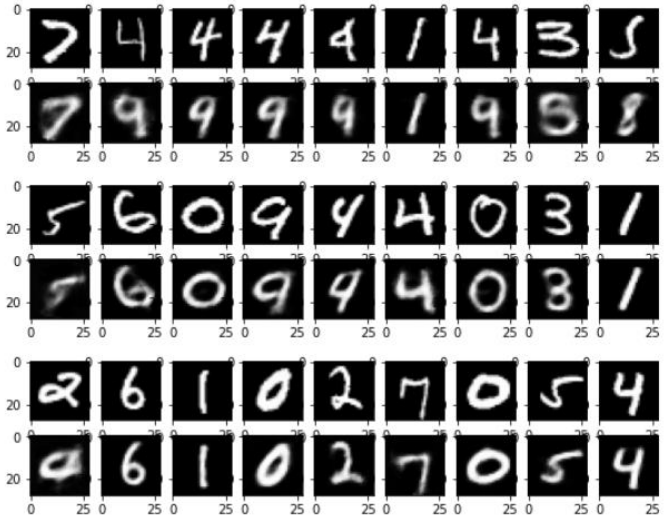
شکل ۹. نمودار Loss در طول آموزش مدل



شکل ۱۰. نمودار Validation Loss در طول آموزش مدل

ملاحظه می‌شود که همانطور که انتظار می‌رفت میزان loss حالت نزولی به خود گرفته است و در نهایت بعد طی ۱۰ مرحله به حد مناسبی از خطا می‌رسد. همچنین در نمودار Validation Loss نیز خطا تا حد دو درصد کاهش می‌یابد. در نتیجه مدل دچار over fit نشده است و وضعیت قابل قبولی دارد.

همچنین جهت بررسی کیفی مدل بعد آموزش، در شکل زیر ورودی ها و خروجی های رندومی از epoch اول، پنجم و نهم آورده شده است.



شکل ۱۱. تصاویر ورودی و خروجی مدل حین آموزش

مشاهده می‌شود که کیفیت عکس‌های خروجی در هر مرحله از لحاظ کیفی بهبود می‌یابند و از میزان blur کاسته می‌شود.

۳-۳. طبقه‌بندی

اکنون در این بخش یک شبکه‌ی Classification (طبقه‌بند) را به صورت دو لایه مخفی با استفاده از داده‌های بدست آمده از بخش encoding شبکه‌ی Auto Encoder، طراحی خواهیم کرد. طبیعتاً با توجه به آنکه نتایج بدست آمده از بخش encoding بردار های ۳۰ بعدی بوده اند، بایستی در لایه‌ی اول ۳۰ نورون قرار بدهیم. همچنین با توجه به اینکه مسئله‌ی ما ۱۰ کلاس دارد، در لایه آخر شبکه نیز بایستی ۱۰ نورون قرار بدهیم تا با استفاده از score های این نورون‌ها و اعمال تابع softmax روی آن، احتمال هر کلاس را برای ورودی داده شده بدست بیاوریم و در نهایت برچسب پیشنهادی را با استفاده از تابع argmax() بعد اتمام آموزش مدل تعیین نماییم.

```python
class Classification(nn.Module):
    def __init__(self):
        super().__init__()
        self.Neural_layers = nn.Sequential(
            nn.Linear(30, 20),
            nn.ReLU(),
            nn.Linear(20, 15),
            nn.ReLU(),
            nn.Linear(15, 10),
            nn.Softmax()
        )


    def forward(self, x):
        Class = self.Neural_layers(x)
        return Class
```

شکل ۱۲. معماری شبکه‌ی طبق‌بند در پایتون

قبل شروع آموزش مدل مقادیر مربوط به بعضی از hyper parameter ها را همچون شبکه‌ی قبلی تعیین می‌کنیم. تفاوتی که در این بخش خواهیم داشت مربوط به loss function است. با توجه به اینکه این شبکه جهت طبقه‌بندی قرار است طراحی گردد، بایستی از cross entropy استفاده نماییم.

```
model_2 = Classification()

criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model_2.parameters(),
                             lr=1e-3,
                             weight_decay=1e-5)
```

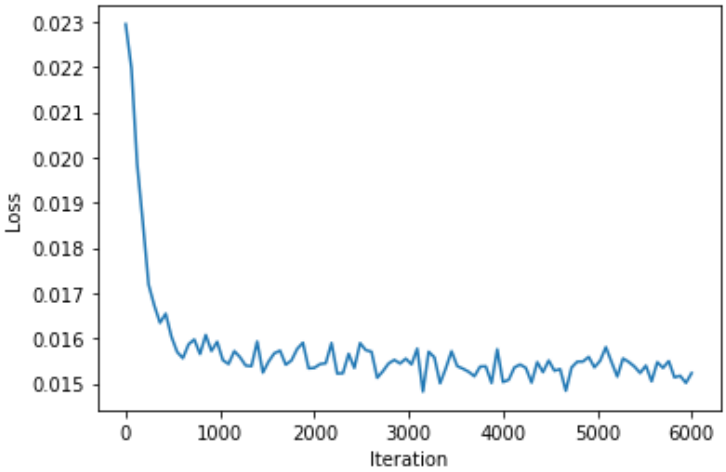
شکل ۱۳. تعریف مولفه‌های مربوط به loss و optimization مدل

در فرآیند آموزش این شبکه، همانطور که قبلا مطرح شد از دیتای encode شده شبکه‌ی قبل استفاده خواهد شد. با دادن این مورد به عنوان ورودی شبکه نتایج بدست آمده از شبکه‌ی طبقه‌بند را به همراه برچسب متناظر ورودی به تابع loss می‌دهیم تا با استفاده از خطا، مدل آموزش یابد و خطایش کمتر شود. همچنین در هر مرحله مقدار دقت مدل را نیز محاسبه می‌کنیم. نتیجه‌ی حاصل بعد طی شدن ۱۰ epoch به صورت زیر خواهد بود.

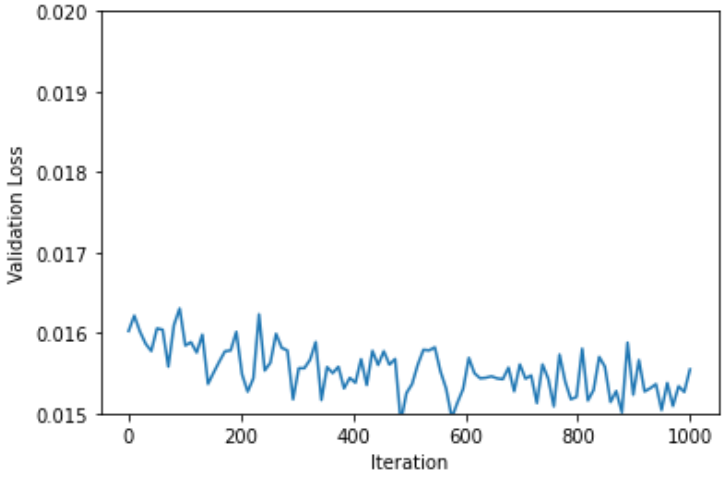
```
Epoch:1, Loss:0.0161, Accuracy:89.0
Epoch:2, Loss:0.0158, Accuracy:89.0
Epoch:3, Loss:0.0155, Accuracy:88.0
Epoch:4, Loss:0.0153, Accuracy:91.0
Epoch:5, Loss:0.0156, Accuracy:89.0
Epoch:6, Loss:0.0158, Accuracy:91.0
Epoch:7, Loss:0.0154, Accuracy:90.0
Epoch:8, Loss:0.0153, Accuracy:91.0
Epoch:9, Loss:0.0151, Accuracy:94.0
Epoch:10, Loss:0.0149, Accuracy:95.0
```

شکل ۱۴. دقت و loss مدل در هر epoch

و نمودار مربوط به Loss و Validation Loss نیز به صورت زیر خواهد بود.

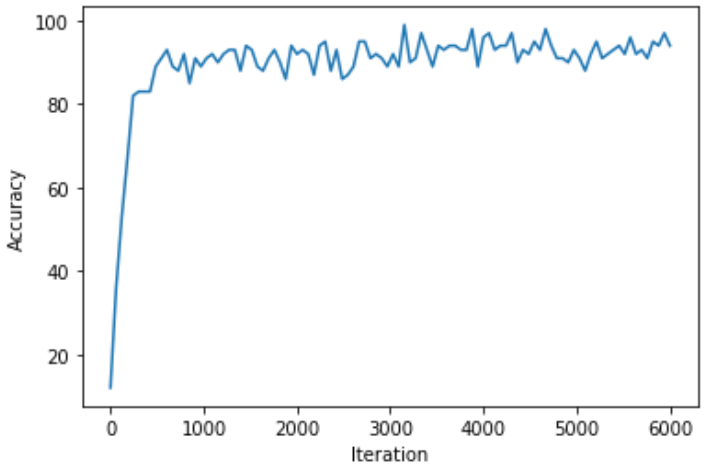


شکل ۱۵. نمودار Loss در طول آموزش مدل

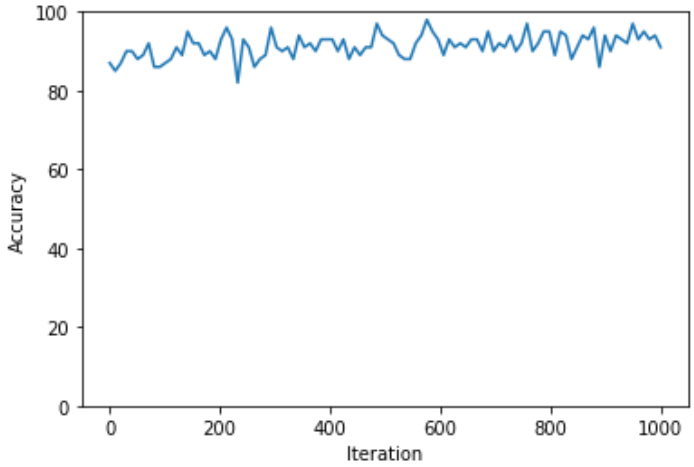


شکل ۱۶. نمودار Validation Loss در طول آموزش مدل

نمودار مربوط به دقت در دو حالت train و validation به ترتیب به صورت زیر است.



شکل ۱۷. نمودار Accuracy در طول آموزش مدل



شکل ۱۸. نمودار Validation Accuracy در طول آموزش مدل

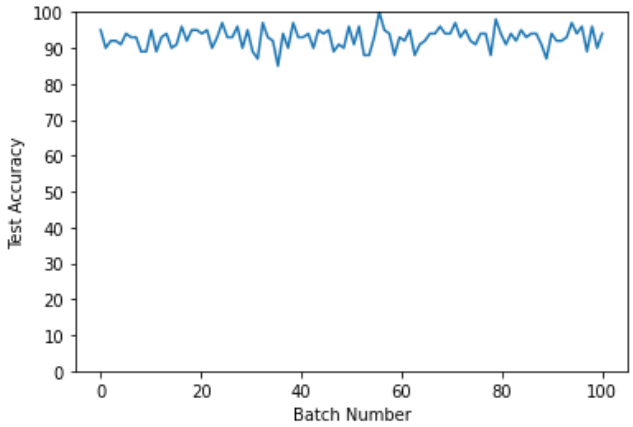
با در نظر گرفتن این نتایج می‌توان عملکرد شبکه را تایید کنیم و مطمئن شویم که مدل دچار overfit یا underfit نشده است.

در نهایت با اتمام آموزش شبکه، از داده‌های تست استفاده می‌کنیم و دقت را بررسی می‌کنیم. دقت پیش‌بینی این مدل برای دیتای test به طور میانگین برابر است با ۹۳ درصد که نشان از قابل قبول بودن مدلمان را می‌دهد.

```
Average Accuracy For Test Data: 93 %
```

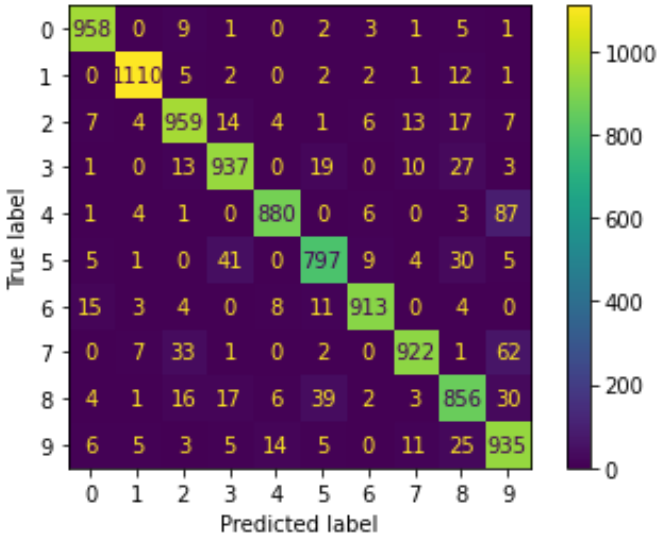
شکل ۱۹. مقدار متوسط دقت برای داده‌های تست

نمودار دقت در هر batch (بسته) برای داده‌های تست به صورت زیر است.



شکل ۲۰. نمودار Accuracy برای داده‌ی تست

برای درک بهتر نتایج مدل در ادامه از Confusion Matrix استفاده خواهیم کرد.



شکل ۲۱. Confusion Matrix مدل برای داده‌های تست .

با توجه به نمودار درهم ریختگی مشاهده می‌شود که همانطور که در نمودار دقت ملاحظه کردیم مدل عملکردی مناسبی در پیش‌بینی برای دیتا های جدید دارد. برای مثال از داده‌هایی که دارای برچسب ۲ بوده اند، ۹۵۹ مورد به درستی پیش‌بینی شده اند. همچنین مدل در پیش‌بینی تصاویر با برچسب ۱ نسبت به بقیه‌ی گروه‌ها از همه بهتر عمل کرده است.