
 <p>دانشگاه صنعتی خواجه نصیرالدین طوسی دانشکده مهندسی برق - گروه مهندسی کنترل</p>	<p>به نام خدا دانشگاه تهران - دانشکده صنعتی خواجه نصیرالدین طوسی تهران دانشکده مهندسی برق و کامپیوتر</p>	
<p><b>شبکه کانولوشنی کم عمق برای طبقه بندی تصاویر</b></p>		

<p>محمد جواد احمدی</p>	<p>نام و نام خانوادگی</p>
<p>۴۰۱۰۰۰۸۶</p>	<p>شماره دانشجویی</p>

## فهرست مطالب

۴	پاسخ پرسش اول	۱
۴	آماده‌سازی و پیش‌پردازش داده‌ها	۱.۱
۸	توضیح لایه‌های مختلف معماری شبکه	۲.۱
۱۰	پیاده‌سازی معماری	۳.۱
۲۱	نتایج پیاده‌سازی	۴.۱
۲۱	پاسخ قسمت الف	۱.۴.۱
۲۸	پاسخ قسمت ب	۲.۴.۱
۳۳	پاسخ قسمت ج	۳.۴.۱

## فهرست تصاویر

۱	نمودار دقت پیاده‌سازی (مجموعه‌داده MNIST)	۲۴
۲	ماتریس درهم‌ریختگی پیاده‌سازی (مجموعه‌داده MNIST)	۲۴
۳	نمودار دقت پیاده‌سازی (مجموعه‌داده Fashion-MNIST)	۲۵
۴	ماتریس درهم‌ریختگی پیاده‌سازی (مجموعه‌داده Fashion-MNIST)	۲۵
۵	نمودار دقت پیاده‌سازی (مجموعه‌داده CIFAR10 - آزمایش اول)	۲۶
۶	ماتریس درهم‌ریختگی پیاده‌سازی (مجموعه‌داده CIFAR10 - آزمایش اول)	۲۶
۷	نمودار دقت پیاده‌سازی (مجموعه‌داده CIFAR10 - آزمایش دوم)	۲۷
۸	ماتریس درهم‌ریختگی پیاده‌سازی (مجموعه‌داده CIFAR10)	۲۷
۹	نمودارهای دقت و اتلاف آموزش و اعتبارسنجی (پیاده‌سازی روی مجموعه‌داده MNIST)	۳۳
۱۰	نمودارهای دقت و اتلاف آموزش و اعتبارسنجی (پیاده‌سازی روی مجموعه‌داده Fashion-MNIST)	۳۳
۱۱	نمودارهای دقت و اتلاف آموزش و اعتبارسنجی (مجموعه‌داده CIFAR10 - آزمایش اول)	۳۴
۱۲	نمودارهای دقت و اتلاف آموزش و اعتبارسنجی (مجموعه‌داده CIFAR10 - آزمایش دوم)	۳۴

## فهرست جداول

۱	نتیجه ارزیابی مدل‌ها روی داده‌های آزمون	۲۸
---	---	----

## پرسش ۱. شبکه‌ی عصبی پیچشی کم عمق برای طبقه‌بندی تصاویر

### ۱ پاسخ پرسش اول

توضیح پوشه کدهای شبکه‌ی عصبی پیچشی کم عمق برای طبقه‌بندی تصاویر

کدهای مربوط به این قسمت، علاوه بر پوشه محلی کدها در این لینک گوگل کولب آورده شده است.

### ۱.۱ آماده‌سازی و پیش‌پردازش داده‌ها

در متن مقاله در رابطه با آماده‌سازی و پیش‌پردازش داده‌ها اینگونه نوشته شده است:

نقل قول از مقاله Shallow convolutional neural network for image classification

مجموعه داده MNIST یک مجموعه داده استاندارد برای طبقه بندی تصاویر است. این مجموعه داده شامل ۱۰ کلاس است، از جمله ۱۰ رقم از ۰ تا ۹. ما ۶۰۰۰۰ تصویر خاکستری برای آموزش و ۱۰۰۰۰ تصویر خاکستری برای آزمون انتخاب کرده ایم. اندازه تصاویر  $28 \times 28$  پیکسل است. مجموعه داده Fashion-MNIST یک مجموعه داده تصویری جدید است که شامل ۷۰۰۰۰ تصویر محصولات مد از ۱۰ دسته بندی مختلف است. همانند MNIST، مجموعه داده Fashion-MNIST شامل ۶۰۰۰۰ تصویر آموزش و ۱۰۰۰۰ تصویر آزمون است. تصاویر این مجموعه داده تصاویر خاکستری با اندازه  $28 \times 28$  پیکسل هستند. ما با احتمال ۰.۵ تصاویر آموزش و آزمون را به صورت تصادفی وارون می کنیم و آنها را به عنوان مجموعه داده آموزش و آزمون خود استفاده می کنیم. مجموعه داده CIFAR10 یک مجموعه داده پرکاربرد برای طبقه بندی تصاویر است، که شامل ۵۰۰۰۰ تصویر رنگی آموزشی و ۱۰۰۰۰ تصویر رنگی آزمون از ۱۰ دسته بندی مختلف است. اندازه این تصاویر  $32 \times 32$  پیکسل است.

برای این منظور برنامه‌ای می‌نویسیم که در آن یک سری تبدیلات برای داده‌های تصویری تعریف می‌شود و داده‌های سه مجموعه داده مورد خواست سوال و گفته شده در مقاله، در دسته‌های مختلف پیش‌پردازش و ذخیره می‌شوند. در ابتدا تمامی کتابخانه‌های مورد نیاز برای کار با داده‌ها و انجام پیش‌پردازش‌ها فراخوانی می‌شوند. سپس، با استفاده از random\_seed، یک نشان تصادفی برای ایجاد تکرارپذیری در اجرای کد تعیین می‌شود. سپس با استفاده از توابع تبدیلات، پیش‌پردازش‌های لازم برای دیتاست‌های مختلف تعریف شده‌اند. هم چنین از آنجا که نرمال سازی مجموعه داده معمولاً عملکرد شبکه های عصبی را بهبود می‌دهد، این فرآیند را هم در برنامه مربوط به پیش‌پردازش داده‌ها گنجانده ایم. بسته به شرایط مختلف و ماهیت داده‌ها، روش‌های مختلفی برای نرمال سازی داده‌ها وجود دارند. ستفاده از میانگین و انحراف معیار خود مجموعه داده به عنوان روش نرمال سازی برای ورودی شبکه عصبی می‌تواند مزایای زیر را داشته باشد:

- کاهش بیش‌برازش وابستگی نتایج به مقدار اولیه داده‌ها: با نرمال سازی مجموعه داده، احتمال بیش‌برازش کاهش می‌یابد. زیرا با کاهش واریانس داده‌ها، شبکه عصبی نسبت به تغییرات ناهمگون در داده‌های آموزشی حساسیت کمتری پیدا می‌کند. هم چنین هنگامی که داده‌هایی با مقیاس‌ها و واحدهای مختلف هستیم، نتایج ممکن است بر اساس این مقیاس‌ها تحت

تأثیر قرار گیرند و به نتایجی منجر شوند که درست نیستند. با انجام نرمال‌سازی، این مشکل رفع می‌شود و وابستگی نتایج به مقدار اولیه داده‌ها کاهش می‌یابد.

- سرعت آموزش بیشتر: با نرمال‌سازی مجموعه داده، همگرایی الگوریتم آموزشی سریعتر می‌شود و به همین دلیل زمان آموزش شبکه کوتاه‌تر می‌شود.

- بالاترین دقت در آزمایش‌ها: استفاده از میانگین و انحراف معیار خود مجموعه داده به عنوان معیار نرمال‌سازی می‌تواند منجر به دقت بالاتری در مدل‌های آموزش داده شده شود. در مقابل، استفاده از مقادیر ثابت به عنوان میانگین و انحراف معیار، می‌تواند باعث شود که مدل دقت کمتری داشته باشد و همچنین ممکن است احتمال بیش‌برازش را تقویت کند. درواقع با استفاده از میانگین و انحراف معیار، ما می‌توانیم تمام داده‌ها را در یک مقیاس یکسان قرار دهیم و در نتیجه دقت تحلیل‌ها را افزایش دهیم. این به دلیل این است که داده‌ها پراکنده‌تر و توزیع آن‌ها غیر یکنواخت نباشد.

بنابراین، روشی که از آن برای نرمال‌سازی استفاده کرده‌ایم استفاده از میانگین و انحراف معیار داده‌های آموزشی بوده است. استفاده از میانگین و انحراف معیار داده‌های آزمون به جای داده‌های آموزش می‌تواند به نوعی یک نوع تقلب در ارزیابی مدل باشد، چرا که با این کار، اطلاعات اضافی در مورد داده‌های آزمون به مدل داده شده است که در واقع برای یادگیری مدل در دوره آموزش در دسترس نبوده است. دلیل استفاده از محور (0, 1, 2) برای محاسبه میانگین و انحراف معیار در مجموعه داده CIFAR10 اما نه در مجموعه داده‌های MNIST و Fashion-MNIST این است که تصاویر CIFAR10 سه کانال رنگی (IrRGB) دارند در حالی که تصاویر MNIST و Fashion-MNIST تنها یک کانال (خاکستری) دارند. در ادامه یک مجموعه پیش‌پردازش مخصوص هر مجموعه داده تعریف می‌کنیم و در نهایت آن را روی آن مجموعه داده اعمال می‌کنیم. این پیش‌پردازش‌ها یک ترکیب از تبدیلات داده‌های تصویری را مورد استفاده قرار می‌دهند. داده‌ها به قالب تنسور تبدیل می‌شوند (ToTensor)، سپس با استفاده از میانگین و انحراف معیار داده‌های تصویری، تصویر نرمال شده و اندازه آن به (28, 28) تغییر می‌کند. هم‌چنین در یک حرکت اضافی تصاویر مجموعه داده‌های MNIST و Fashion-MNIST به سیاه و سفید تبدیل می‌شوند (Grayscale). هم‌چنین از آن‌جا که در توضیح لایه‌های مدل به صراحت ذکر شده که ورودی تک‌کاناله است، پیش‌پردازشی برای این موضوع هم به مجموعه پیش‌پردازشی مجموعه داده CIFAR10 اضافه کرده ایم. از طرف دیگر، برای مجموعه داده Fashion-MNIST عملیات پردازشی flip با احتمال گفته شده در مقاله 0.5 هم در نظر گرفته شده است. در نهایت مجموعه داده‌های پیش‌پردازش شده در دسته‌های مختلف آموزش، اعتبارسنجی و آزمون و در قالب pt جهت استفاده‌های بعدی ذخیره می‌شوند. برنامه مربوط به این قسمت در برنامه ۱ آورده شده است.

#### Program 1: Pre-processing Code

```
1 import torch
2 import torchvision
3 import torchvision.transforms as transforms
4 import random
5 import os
6
7 # set random seed
8 random_seed = 42
9 torch.manual_seed(random_seed)
10 random.seed(random_seed)
11
12 # download datasets
```

```

13 main_trainset_cifar10 = torchvision.datasets.CIFAR10(root='./data', train=True,
14                                     download=True, transform=None)
15 train_mean_cifar10 = main_trainset_cifar10.data.mean(axis=(0, 1, 2))/255
16 train_std_cifar10 = main_trainset_cifar10.data.std(axis=(0, 1, 2))/255
17
18 main_testset_cifar10 = torchvision.datasets.CIFAR10(root='./data', train=False,
19                                     download=True, transform=None)
20 test_mean_cifar10 = main_testset_cifar10.data.mean(axis=(0, 1, 2))/255
21 test_std_cifar10 = main_testset_cifar10.data.std(axis=(0, 1, 2))/255
22
23 main_trainset_mnist = torchvision.datasets.MNIST(root='./data', train=True,
24                                     download=True, transform=None)
25 train_mean_mnist = main_trainset_mnist.data.float().mean()/255
26 train_std_mnist = main_trainset_mnist.data.float().std()/255
27
28 main_testset_mnist = torchvision.datasets.MNIST(root='./data', train=False,
29                                     download=True, transform=None)
30 test_mean_mnist = main_testset_mnist.data.float().mean()/255
31 test_std_mnist = main_testset_mnist.data.float().std()/255
32
33 main_trainset_fashion_mnist = torchvision.datasets.FashionMNIST(root='./data', train=True,
34                                     download=True, transform=None)
35 train_mean_fashion_mnist = main_trainset_fashion_mnist.data.float().mean()/255
36 train_std_fashion_mnist = main_trainset_fashion_mnist.data.float().std()/255
37
38 main_testset_fashion_mnist = torchvision.datasets.FashionMNIST(root='./data', train=False,
39                                     download=True, transform=None)
40 test_mean_fashion_mnist = main_testset_fashion_mnist.data.float().mean()/255
41 test_std_fashion_mnist = main_testset_fashion_mnist.data.float().std()/255
42
43 # define data transformations
44 transform_cifar10 = transforms.Compose(
45     [transforms.ToTensor(),
46      transforms.Normalize(mean=train_mean_cifar10, std=train_std_cifar10),
47      transforms.Resize((28, 28))]
48 )
49
50 # define data transformations
51 transform_mnist = transforms.Compose(
52     [transforms.Grayscale(num_output_channels=1),
53      transforms.ToTensor(),
54      transforms.Normalize(mean=train_mean_mnist, std=train_std_mnist),
55      transforms.Resize((28, 28))]
56 )
57

```

```

58 # define flip transform for Fashion-MNIST dataset
59 transform_fashion_mnist = transforms.Compose(
60     [transforms.RandomHorizontalFlip(p=0.5),
61      transforms.Grayscale(num_output_channels=1),
62      transforms.ToTensor(),
63      transforms.Normalize(mean=train_mean_fashion_mnist, std=train_std_fashion_mnist),
64      transforms.Resize((28, 28))]
65 )
66
67 # download datasets
68 trainset_cifar10 = torchvision.datasets.CIFAR10(root='./data', train=True,
69                                                download=True, transform=transform_cifar10)
70 testset_cifar10 = torchvision.datasets.CIFAR10(root='./data', train=False,
71                                                download=True, transform=transform_cifar10)
72
73 trainset_mnist = torchvision.datasets.MNIST(root='./data', train=True,
74                                             download=True, transform=transform_mnist)
75 testset_mnist = torchvision.datasets.MNIST(root='./data', train=False,
76                                             download=True, transform=transform_mnist)
77
78 trainset_fashion_mnist = torchvision.datasets.FashionMNIST(root='./data', train=True,
79                                                            download=True, transform=transform_fashion_mnist)
80 testset_fashion_mnist = torchvision.datasets.FashionMNIST(root='./data', train=False,
81                                                            download=True, transform=transform_fashion_mnist)
82
83 # split trainset into train and validation sets
84 trainset_cifar10, valset_cifar10 = torch.utils.data.random_split(trainset_cifar10, [40000,
85                                                                                     10000])
86 trainset_mnist, valset_mnist = torch.utils.data.random_split(trainset_mnist, [50000, 10000])
87 trainset_fashion_mnist, valset_fashion_mnist = torch.utils.data.random_split(
88     trainset_fashion_mnist, [48000, 12000])
89
90 # save datasets
91 os.makedirs('./data/preprocessed/', exist_ok=True)
92 torch.save(trainset_cifar10, './data/preprocessed/trainset_cifar10.pt')
93 torch.save(valset_cifar10, './data/preprocessed/valset_cifar10.pt')
94 torch.save(testset_cifar10, './data/preprocessed/testset_cifar10.pt')
95
96 torch.save(trainset_mnist, './data/preprocessed/trainset_mnist.pt')
97 torch.save(valset_mnist, './data/preprocessed/valset_mnist.pt')
98 torch.save(testset_mnist, './data/preprocessed/testset_mnist.pt')
99
100 torch.save(trainset_fashion_mnist, './data/preprocessed/trainset_fashion_mnist.pt')
101 torch.save(valset_fashion_mnist, './data/preprocessed/valset_fashion_mnist.pt')
102 torch.save(testset_fashion_mnist, './data/preprocessed/testset_fashion_mnist.pt')

```



## ۲.۱ توضیح لایه‌های مختلف معماری شبکه

این مقاله به مزایای شبکه‌های عصبی کانولوشنی (CNN) برای برخی وظایف بینایی رایانه مانند طبقه‌بندی تصویر و تشخیص اشیا اشاره می‌کند. این مقاله برای رفع مشکلات مرتبط با تعداد لایه‌های زیاد و پیچیدگی در شبکه‌های عمیق که منجر به مصرف منابع محاسباتی و حافظه بالا در ساختاری پیچیده و زمان آموزش بالا می‌گردند، یک شبکه عصبی کانولوشنی کم عمق (SCNNB) با فقط چهار لایه معرفی می‌کند که به پیچیدگی فضایی-زمانی کمی هم نیاز دارد. لایه‌های این شبکه شامل دو لایه کانولوشنی، دو لایه پولینگ، یک لایه تماماً متصل و یک لایه سافت مکس است. از تکنیک هم در پیاده‌سازی مدل استفاده شده است. هم‌چنین از Batch Normalization برای شتاب‌دادن به همگرایی شبکه و بهبود دقت استفاده شده است. تمرکز اصلی این مقاله بر روی همین لایه است. لایه Batch Normalization با تغییر توزیع ورودی هر لایه از شبکه، از بروز مشکل Covariate Shift جلوگیری می‌کند و به تسریع همگرایی شبکه کمک می‌کند. به عبارت دیگر، با اعمال آن شبکه می‌تواند بدون وابستگی به محدودیت‌های تغییرات ورودی، با سرعت بیشتری به همگرایی برسد. در واقع، این تکنیک باعث کاهش تاثیر وابستگی بین وزن‌های لایه‌ها می‌شود و مانع از بروز بیش‌برازش در شبکه می‌شود. بنابراین، استفاده از این لایه در شبکه‌های کانولوشنی باعث بهبود سرعت و کیفیت آموزش شبکه، افزایش دقت تخمین و کاهش مشکلات ناشی از مقداردهی اولیه پارامترها می‌شود. این مقاله، با افزودن این لایه بعد از هر لایه کانولوشنی، سرعت همگرایی شبکه را افزایش داده و دقت آن را هم زیاد کرده است. لازم به ذکر است که لایه‌های کانولوشنی در این شبکه دارای اندازه کوچک  $3 \times 3$  هستند که باعث کاهش پیچیدگی فضایی-زمانی شبکه می‌شود. در مقایسه مدل پیشنهادی مقاله با مدلی دیگر، نشان داده شده که این مدل بدون نیاز به پیش آموزش، می‌تواند دقت بالاتری در دسته بندی تصاویر داشته باشد. تمرکز اصلی این مقاله بر تاثیر Batch Normalization در دسته‌بندی تصاویر است.

مقاله تصریح می‌کند که شبکه‌های عصبی عمیق کانولوشنی مانند Mobilenet تعداد زیادی لایه دارند و زمان آموزش آن‌ها تا چندین روز، هفته و یا حتی بیش‌تر طول می‌کشد. برای حل کردن این چالش، مقاله یک چهارچوب کم عمق با لایه‌های کم‌تر و اندازه‌هسته‌های کوچک‌تر کانولوشنی پیشنهاد کرده است. مدل پیشنهادی شبکه ورودی با ابعاد ۲۸ می‌پذیرد و در ابتدا، ویژگی‌های داده‌ای کم عمق را با کانولوشن‌های سه‌درسه با ۳۲ فیلتر استخراج می‌کند. به‌صورت کلی این مقاله از لایه‌های زیر برای پیاده‌سازی خود استفاده می‌کند:

- **Input layer:** این لایه که لایه ورودی نام دارد، ابعاد تصاویر ورودی را مشخص و اعمال می‌کند. در این مورد، ابعاد تصویر ورودی ۲۸ در ۲۸ پیکسل خواهد بود. تعیین ابعاد دقیق لایه ورودی بسیار مهم است و باید با دقت انجام شود تا داده‌های ورودی به درستی دریافت و پردازش شوند.
- **Convolutional layer:** لایه کانولوشنی یکی از مهم‌ترین لایه‌های در شبکه‌های کانولوشنی است که می‌تواند با استفاده از کرنل و ماتریس‌های کانولوشن وزن‌دار، رأساً ویژگی استخراج کند. هرچه هسته‌های کانولوشن بیش‌تر باشد، توانایی استخراج ویژگی‌ها قوی‌تر است. مدل پیشنهادی مقاله شامل دو لایه کانولوشن  $3 \times 3$  است که به ترتیب دارای ۳۲ و ۶۴ فیلتر هستند. تعداد فیلترها و اندازه‌های مختلف آن‌ها، به شبکه کمک می‌کند تا ویژگی‌های مختلف و پیچیده‌تری از داده‌ها را استخراج کند.
- **Max-pooling layer:** این لایه در شبکه‌های عصبی کانولوشنی به منظور کاهش ابعاد داده‌ها و پیچیدگی محاسباتی و هم‌چنین حفظ ویژگی‌های مفید استفاده می‌شود. پس از استخراج ویژگی‌های داده‌ها توسط لایه کانولوشنی، از لایه ادغام برای کاهش افزونگی از طریق فرونمونه‌برداری ویژگی‌های استخراج‌شده استفاده می‌شود. مدل پیشنهادی مقاله از دو لایه  $2 \times 2$  max-pooling برای کاهش ابعاد داده و پیچیدگی محاسباتی استفاده می‌کند و در عین حال ویژگی‌های مفید استخراج‌شده را تقریباً بدون تغییر نگه می‌دارد. لایه‌های ادغام نه تنها افزونگی ویژگی‌های داده‌ها و خطر بیش‌برازش را کاهش می‌دهند، بلکه سرعت هم‌گرایی را نیز بهبود می‌بخشند.

• Fully connected layer: لایه تماماً متصل، لایه‌ای از شبکه‌های عصبی عمیق است که همه نورون‌های آن به همه نورون‌های لایه‌ی قبلی متصل هستند و هیچ ارتباطی بین نورون‌های هم‌سطح لایه وجود ندارد. هدف از استفاده از لایه کاملاً متصل، استخراج ویژگی‌های پیچیده‌تری از ویژگی‌های استخراج شده از لایه‌های قبلی است. در مدل پیشنهادی مقاله، ویژگی‌های استخراج شده از لایه قبلی با استفاده از یک لایه کاملاً متصل با ۱۲۸۰ نورون با هم تلفیق می‌شوند. این لایه کاملاً متصل در واقع یک عمل کانولوشن  $1 \times 1 \times 3136$  است که اندازه فیلترهای آن با اندازه ویژگی‌های خروجی لایه قبلی  $(7 \times 7 \times 64)$  برابر است. به این صورت، ویژگی‌های استخراج شده از لایه‌های قبلی با هم ترکیب شده و اطلاعات مهم‌تری در اختیار شبکه‌ی عصبی قرار می‌گیرد.

• Softmax output layer: این لایه در شبکه‌های عصبی کانولوشنال با هدف طبقه‌بندی چندکلاسه استفاده می‌شود. این لایه، خروجی مدل را به یک توزیع احتمال تبدیل می‌کند. به عبارت دیگر، این لایه برای محاسبه احتمال هر کلاس از خروجی‌های مدل استفاده می‌شود. تابع softmax مقدار خروجی را به نحوی تغییر می‌دهد که مجموع مقادیر خروجی‌ها برابر یک شود و این خروجی‌ها را به عنوان نوعی از احتمالات پیش‌بینی کلاس در نظر می‌گیرد. با استفاده از تابع softmax، مقدار پیش‌بینی شده برای هر کلاس به عنوان یک احتمال تعریف می‌شود و مجموع تمام احتمالات برابر با یک می‌شود. بنابراین، اگر ما نیاز داشته باشیم تا کلاسی را که مدل برای آن احتمال بالاتری پیش‌بینی کرده است، تشخیص دهیم، می‌توانیم کلاسی که مقدار احتمال بالاتری دارد را به عنوان پاسخ مدل انتخاب کنیم. اگر  $y_i$  را خروجی گره hl، را تعداد گره‌های لایه خروجی در نظر بگیریم رابطه محاسباتی این لایه به صورت زیر تعیین می‌شود:

$$\text{softmax}(y)_i = \frac{e^{y_i}}{\sum_{j=1}^n e^{y_j}} \quad (۱)$$

• Relu: این تابع فعال‌ساز غیرخطی می‌تواند از داده‌ها نگاشت هر تابع پیچیده از ورودی تا خروجی را برای حل مسائل غیرخطی بیاموزد و مدل را قدرتمندتر کند. این تابع به عنوان یک تابع فعال‌سازی، با اعمال مقدار بیشینه بین ۰ و ورودی، به صورت زیر تعریف می‌شود:

$$f(x) = \max(0, x) \quad (۲)$$

با توجه به این که این تابع، تابعی غیرخطی است، قادر است به شبکه اجازه دهد تا از توانایی حل مسائل غیرخطی برخوردار شود. علاوه بر این، یکی از ویژگی‌های مهم این تابع این است که از نظر فعال‌سازی چندان مقادیر غیرصفری دریافت نمی‌کند و در نتیجه، محاسبات تابع هزینه و پیش‌بینی مقادیر خروجی بسیار سریع خواهد بود. این ویژگی که به sparse activation شناخته می‌شود، به شبکه کمک می‌کند تا در برابر بیش‌برازش مقاوم شود و به خوبی عمل کند. به طور کلی مزایای این تابع را می‌توان این گونه برشمرد: Relu به عنوان یک تابع فعال‌ساز ساده و سریع به شمار می‌رود و در نتیجه به صورت کلی سرعت یادگیری در شبکه را افزایش می‌دهد. Relu از نظر فعال‌سازی چندان مقادیر غیرصفر دریافت نمی‌کند و در نتیجه انجام محاسبات و پیش‌بینی مقادیر خروجی بسیار سریع خواهد بود. Relu باعث ایجاد activation Sparse می‌شود که به شبکه کمک می‌کند تا در برابر بیش‌برازش مقاوم شود. Relu در مقایسه با توابع فعال‌سازی دیگری که در گذشته استفاده می‌شدند، از جمله تابع sigmoid و tanh، مشکلات کم‌تری دارد و بهترین عملکرد را در شبکه‌های عصبی عمیق دارد. اما این تابع معایبی هم دارد؛ از جمله این که: اگر در برخی موارد، مقدار ورودی به تابع Relu کوچک باشد، این تابع مقدار صفر را برمی‌گرداند. هم‌چنین در برخی حالت‌ها، این تابع ممکن است باعث بروز مشکل Gradient Explosion شود که باعث کندی در فرآیند یادگیری می‌شود. برای مقابله با مشکل Vanishing Gradient در استفاده از توابع فعال‌سازی، توابعی مانند Leaky Relu، ELU و IrSELU توسعه داده شده‌اند که در برخی موارد می‌توانند بهتر

از Relu عمل کنند. به عنوان مثال، Relu Leaky با حفظ مشخصات Relu و افزودن یک مقدار منفی کوچک برای ورودی‌های منفی، مشکل Vanishing Gradient را برطرف می‌کند.

- BN (Batch Normalization): این لایه با نرمال‌سازی خروجی لایه‌ها منجر به افزایش سرعت آموزش مدل شود و به جلوگیری از مشکل اشباع و ناپایداری در آموزش شبکه کمک می‌کند. در مورد این لایه که مهم‌ترین ابتکار این مقاله است پیش‌تر به صورت مفصل توضیحاتی ارائه شده است.
- Dropout: این ابتکار یک تکنیک مناسب برای جلوگیری از بیش‌برازش در شبکه‌های عصبی است. این لایه در فرآیند آموزش شبکه عصبی، با احتمال مشخص، گره‌های مختلفی را خاموش می‌کند. خاموشی تصادفی این گره‌ها باعث می‌شود که شبکه عصبی به صورت اجباری با اطلاعات کمتری آموزش ببیند. در شبکه‌های عصبی با تعداد پارامترهای بسیار زیاد، بسیاری از این پارامترها ممکن است غیرضروری باشند. به طور مشابه، در برخی موارد، تعداد داده‌های آموزش نیز ممکن است کافی نباشد. این موارد می‌تواند منجر به ایجاد بیش‌برازش شود که به معنی آموزش خیلی خوب روی داده‌های آموزش ولی عملکرد ناکافی بر روی داده‌های آزمون است.

### ۳.۱ پیاده‌سازی معماری

برای پیاده‌سازی معماری یک مدل شبکه عصبی با نام SCNNB را با استفاده از کتابخانه پایتورچ تعریف می‌کنیم. کلاس SCNNB از کلاس nn.Module که یک کلاس پایه برای همه ماژول‌های شبکه عصبی در پایتورچ است به ارث می‌رسد. روش \_\_init\_\_ معماری پایه شبکه عصبی را با ایجاد و مقادردهی لایه‌های مختلف تعریف می‌کند. لایه‌های در نظر گرفته شده بر مبنای مقاله به شرح زیر هستند:

- conv1: یک لایه کانولوشنی دوبعدی با یک کانال ورودی، ۳۲ کانال خروجی و اندازه هسته ۳.
- bn1: یک لایه نرمال‌سازی دسته که خروجی conv1 را نرمال می‌کند.
- relu1: یک تابع فعال‌سازی که به خروجی bn1 خاصیت غیرخطی هم اضافه می‌کند.
- pool1: یک لایه ادغام بیشینه حداکثر دوبعدی که ابعاد فضایی خروجی relu1 را فرو نمونه‌برداری کرده و کاهش می‌دهد.
- conv2: یک لایه کانولوشنی دوبعدی با ۳۲ کانال ورودی (تعداد کانال‌های خروجی، conv1 ۶۴ کانال خروجی و اندازه هسته ۳).
- bn2: یک لایه نرمال‌سازی دسته که خروجی conv2 را نرمال می‌کند.
- relu2: یک تابع فعال‌سازی که به خروجی bn2 خاصیت غیرخطی هم اضافه می‌کند.
- pool2: یک لایه ادغام بیشینه حداکثر دوبعدی که ابعاد فضایی خروجی relu2 را فرو نمونه‌برداری کرده و کاهش می‌دهد.
- fc1: یک لایه تماماً متصل که خروجی مسطح pool2 را به عنوان ورودی می‌گیرد و دارای ۱۲۸۰ واحد خروجی است.
- relu3: یک تابع فعال‌سازی که به خروجی fc1 خاصیت غیرخطی هم اضافه می‌کند.
- dropout: یک روش تنظیم است که به طور تصادفی کسری از خروجی relu3 را در حین آموزش خاموش می‌کند تا از بیش‌برازش جلوگیری شود.

• fc2: یک لایهٔ تماماًمتصل که خروجی را به عنوان ورودی می‌گیرد و دارای ۱۰ واحد خروجی است (مطابق با ۱۰ کلاس در مجموعه داده).

• softmax: یک تابع فعال سازی softmax که خروجی fc2 را به توزیع احتمالاتی در ۱۰ کلاس تبدیل می‌کند.

در ادامه روش Forward گذار رو به جلوی شبکه را تعریف می‌کند که یک تانسور ورودی با نام x را گرفته و از هر لایه به ترتیبی که تعریف شد عبور می‌دهد. گذر رو به جلو شامل عملیات‌های زیر است:

• conv1, bn1, relu1 و pool1 روی ورودی x اعمال می‌شوند.

• خروجی pool2 مسطح می‌شود و از fc1 و relu3 عبور می‌کند.

• تنظیم dropout به به خروجی relu3 اعمال می‌شود.

• خروجی تنظیم از fc2 و softmax عبور داده می‌شود.

• خروجی نهایی بازگردانده می‌شود.

همان طور که در مقاله اشاره شده به منظور اثبات اینکه bn می‌تواند آموزش شبکه را تسریع کند و دقت را بهبود بخشد، دو نوع SCNNB به شرح زیر معرفی شده است که در هریک تغییراتی که در کد کلاس رخ می‌دهد را قید کرده‌ایم:

• SCNNB-a: تنها لایهٔ bn را بعد از اولین لایهٔ کانولوشنی حذف می‌کنیم (یعنی لایهٔ bn1).

• SCNNB-b: تمام لایه‌های bn را بعد از اولین لایهٔ کانولوشنی حذف می‌کنیم (یعنی لایه‌های bn1 و bn2).

با ارائهٔ این توضیحات کد کلاس مربوط به شبکه‌ها به شرح برنامهٔ ۲ است.

## Program 2: SCNNB Implementation

```
1 # Define the model
2 class SCNNB(nn.Module):
3     def __init__(self):
4         super(SCNNB, self).__init__()
5         self.conv1 = nn.Conv2d(in_channels=1, out_channels=32, kernel_size=3)
6         self.bn1 = nn.BatchNorm2d(32)
7         self.relu1 = nn.ReLU(inplace=True)
8         self.pool1 = nn.MaxPool2d(kernel_size=2, stride=2)
9         self.conv2 = nn.Conv2d(in_channels=32, out_channels=64, kernel_size=3)
10        self.bn2 = nn.BatchNorm2d(64)
11        self.relu2 = nn.ReLU(inplace=True)
12        self.pool2 = nn.MaxPool2d(kernel_size=2, stride=2)
13        self.fc1 = nn.Linear(in_features=64*5*5, out_features=1280)
14        self.relu3 = nn.ReLU(inplace=True)
15        self.dropout = nn.Dropout(p=0.5)
16        self.fc2 = nn.Linear(in_features=1280, out_features=10)
17        self.softmax = nn.Softmax(dim=1)
18
19    def forward(self, x):
```

```

20     x = self.conv1(x)
21     x = self.bn1(x)
22     x = self.relu1(x)
23     x = self.pool1(x)
24     x = self.conv2(x)
25     x = self.bn2(x)
26     x = self.relu2(x)
27     x = self.pool2(x)
28     x = x.view(-1, 64*5*5)
29     x = self.fc1(x)
30     x = self.relu3(x)
31     x = self.dropout(x)
32     x = self.fc2(x)
33     x = self.softmax(x)
34     return x
35
36 # SCNNB-a
37 class SCNNB_a(nn.Module):
38     def __init__(self):
39         super(SCNNB_a, self).__init__()
40         self.conv1 = nn.Conv2d(in_channels=1, out_channels=32, kernel_size=3)
41         self.relu1 = nn.ReLU(inplace=True)
42         self.pool1 = nn.MaxPool2d(kernel_size=2, stride=2)
43         self.conv2 = nn.Conv2d(in_channels=32, out_channels=64, kernel_size=3)
44         self.bn2 = nn.BatchNorm2d(64)
45         self.relu2 = nn.ReLU(inplace=True)
46         self.pool2 = nn.MaxPool2d(kernel_size=2, stride=2)
47         self.fc1 = nn.Linear(in_features=64*5*5, out_features=1280)
48         self.relu3 = nn.ReLU(inplace=True)
49         self.dropout = nn.Dropout(p=0.5)
50         self.fc2 = nn.Linear(in_features=1280, out_features=10)
51         self.softmax = nn.Softmax(dim=1)
52
53     def forward(self, x):
54         x = self.conv1(x)
55         x = self.relu1(x)
56         x = self.pool1(x)
57         x = self.conv2(x)
58         x = self.bn2(x)
59         x = self.relu2(x)
60         x = self.pool2(x)
61         x = x.view(-1, 64*5*5)
62         x = self.fc1(x)
63         x = self.relu3(x)
64         x = self.dropout(x)

```

```

65     x = self.fc2(x)
66     x = self.softmax(x)
67     return x
68
69 # SCNNB-b
70 class SCNNB_b(nn.Module):
71     def __init__(self):
72         super(SCNNB_b, self).__init__()
73         self.conv1 = nn.Conv2d(in_channels=1, out_channels=32, kernel_size=3)
74         self.relu1 = nn.ReLU(inplace=True)
75         self.pool1 = nn.MaxPool2d(kernel_size=2, stride=2)
76         self.conv2 = nn.Conv2d(in_channels=32, out_channels=64, kernel_size=3)
77         self.relu2 = nn.ReLU(inplace=True)
78         self.pool2 = nn.MaxPool2d(kernel_size=2, stride=2)
79         self.fc1 = nn.Linear(in_features=64*5*5, out_features=1280)
80         self.relu3 = nn.ReLU(inplace=True)
81         self.dropout = nn.Dropout(p=0.5)
82         self.fc2 = nn.Linear(in_features=1280, out_features=10)
83         self.softmax = nn.Softmax(dim=1)
84
85     def forward(self, x):
86         x = self.conv1(x)
87         x = self.relu1(x)
88         x = self.pool1(x)
89         x = self.conv2(x)
90         x = self.relu2(x)
91         x = self.pool2(x)
92         x = x.view(-1, 64*5*5)
93         x = self.fc1(x)
94         x = self.relu3(x)
95         x = self.dropout(x)
96         x = self.fc2(x)
97         x = self.softmax(x)
98         return x

```

در ادامه به معرفی کامل تر کد پیاده‌سازی می‌پردازیم. ساختار کلی کد یک اسکریپت پایتون است که در ابتدا چندین ماژول را از کتابخانه‌های PyTorch و scikit-learn وارد می‌کند، مجموعه داده‌های از پیش پردازش و ذخیره شده در **بخش ۱.۱** را بارگیری می‌کند. این کد کتابخانه‌های مختلف پایتون را وارد می‌کند: PyTorch برای ایجاد و آموزش شبکه‌های عصبی، scikit-learn برای ارزیابی عملکرد مدل، Matplotlib و Seaborn برای تجسم داده‌ها و NumPy برای محاسبات عددی. در ادامه، دستوراتی نوشته‌ایم که مجموعه داده‌های از پیش پردازش و ذخیره شده در **بخش ۱.۱** را از مسیر فایل‌های مشخص شده بارگیری کند. سپس، فرآینادهای موردنیاز را بر اساس مقاله تعریف کردیم. اندازه دسته روی ۱۲۸ تنظیم شده است، به این معنی که مدل بر روی دسته‌هایی از ۱۲۸ تصویر در یک زمان آموزش داده می‌شود. نرخ یادگیری روی 0.02 تنظیم شده است که تعیین می‌کند مدل چقدر وزن‌های خود را در پاسخ به خطای بین پیش‌بینی‌های خود و برچسب‌های واقعی تنظیم می‌کند. مومنتوم روی 0.9 تنظیم شده است که به مدل کمک می‌کند تا با سرعت بیشتری در جهت شیب‌های تندتر حرکت کند. کاهش وزن روی 0.000005

تنظیم شده است، که با افزودن یک عبارت جریمه به تابع ضرر، به جلوگیری از بیش‌برازش کمک می‌کند. تعداد دوره‌ها روی ۱۵۰ تنظیم شده است که نشان‌دهنده تعداد دفعاتی است که از کل مجموعه آموزشی برای به‌روزرسانی مدل استفاده می‌شود. در ادامه، بارگذارهای داده با استفاده از کلاس  `DataLoader PyTorch`  ایجاد می‌شوند که مجموعه‌داده‌های از پیش پردازش شده و چندین آرگومان از جمله اندازه دسته و گزینه  `shuffle`  را می‌گیرد. ادامه دستورات را به صورتی می‌نویسیم که سه مدل تعریف شده را با استفاده از سه مجموعه‌داده آموزش دهد. مدل‌ها با استفاده از فرود‌گرایان تصادفی با تابع اتلاف آنتروپی متقابل آموزش داده می‌شوند. حلقه آموزشی برای تعداد معینی از دوره‌ها ( `num_epochs` ) اجرا می‌شود و برای هر دوره، مدل‌ها روی مجموعه آموزشی آموزش داده می‌شوند و بر روی مجموعه‌های اعتبارسنجی و تست ارزیابی می‌شوند. حلقه آموزش لیست‌هایی را برای ردیابی تلفات آموزشی و اعتبارسنجی و دقت برای هر مدل مقداردهی می‌کند. سپس برای هر مدل، بهینه‌ساز را با فرآپارامترهای مشخص شده تعریف می‌کند. سپس حلقه در هر دوره تکرار می‌شود و مراحل زیر انجام می‌پذیرد: متغیرها را برای ردیابی اتلاف و دقت مدل در طول آموزش و اعتبارسنجی راه‌اندازی می‌کند. مدل روی حالت آموزش تنظیم می‌شود و ضمن تکرار آموزش روی مجموعه آموزشی، اتلاف و دقت را محاسبه می‌کند و وزن‌های مدل را به‌روز می‌کند. در ادامه مدل روی حالت ارزیابی قرار می‌گیرد و روی مجموعه اعتبارسنجی، اتلاف و دقت را محاسبه می‌کند. در نهایت، دستوراتی برای نمایش و رسم عمل‌کرد مدل‌ها تعریف کرده‌ایم. با ارائه این توضیحات یک نمونه از مجموعه دستورات در **برنامه ۳** آورده شده است و در نمونه‌های دیگر صرفاً مسیرهای مجموعه‌داده جایگزین می‌شوند.

### Program 3: SCNNB Train/Test Implementation

```

1 import torch
2 import torch.nn as nn
3 import torch.optim as optim
4 from torch.utils.data import DataLoader
5 from sklearn.metrics import classification_report, confusion_matrix
6 import matplotlib.pyplot as plt
7 import seaborn as sns
8 import numpy as np
9
10 # Load the preprocessed datasets
11 trainset_mnist = torch.load('./data/preprocessed/trainset_mnist.pt')
12 valset_mnist = torch.load('./data/preprocessed/valset_mnist.pt')
13 testset_mnist = torch.load('./data/preprocessed/testset_mnist.pt')
14
15 # Define hyperparameters
16 batch_size = 128
17 learning_rate = 0.02
18 momentum = 0.9
19 weight_decay = 0.000005
20 num_epochs = 150
21
22 # Create data loaders
23 trainloader_mnist = DataLoader(trainset_mnist, batch_size=batch_size, shuffle=True)
24 valloader_mnist = DataLoader(valset_mnist, batch_size=batch_size, shuffle=False)
25 testloader_mnist = DataLoader(testset_mnist, batch_size=batch_size, shuffle=False)
26

```

```

27 # Define the model
28 class SCNNB(nn.Module):
29     def __init__(self):
30         super(SCNNB, self).__init__()
31         self.conv1 = nn.Conv2d(in_channels=1, out_channels=32, kernel_size=3)
32         self.bn1 = nn.BatchNorm2d(32)
33         self.relu1 = nn.ReLU(inplace=True)
34         self.pool1 = nn.MaxPool2d(kernel_size=2, stride=2)
35         self.conv2 = nn.Conv2d(in_channels=32, out_channels=64, kernel_size=3)
36         self.bn2 = nn.BatchNorm2d(64)
37         self.relu2 = nn.ReLU(inplace=True)
38         self.pool2 = nn.MaxPool2d(kernel_size=2, stride=2)
39         self.fc1 = nn.Linear(in_features=64*5*5, out_features=1280)
40         self.relu3 = nn.ReLU(inplace=True)
41         self.dropout = nn.Dropout(p=0.5)
42         self.fc2 = nn.Linear(in_features=1280, out_features=10)
43         self.softmax = nn.Softmax(dim=1)
44
45     def forward(self, x):
46         x = self.conv1(x)
47         x = self.bn1(x)
48         x = self.relu1(x)
49         x = self.pool1(x)
50         x = self.conv2(x)
51         x = self.bn2(x)
52         x = self.relu2(x)
53         x = self.pool2(x)
54         x = x.view(-1, 64*5*5)
55         x = self.fc1(x)
56         x = self.relu3(x)
57         x = self.dropout(x)
58         x = self.fc2(x)
59         x = self.softmax(x)
60         return x
61
62 # SCNNB-a
63 class SCNNB_a(nn.Module):
64     def __init__(self):
65         super(SCNNB_a, self).__init__()
66         self.conv1 = nn.Conv2d(in_channels=1, out_channels=32, kernel_size=3)
67         self.relu1 = nn.ReLU(inplace=True)
68         self.pool1 = nn.MaxPool2d(kernel_size=2, stride=2)
69         self.conv2 = nn.Conv2d(in_channels=32, out_channels=64, kernel_size=3)
70         self.bn2 = nn.BatchNorm2d(64)
71         self.relu2 = nn.ReLU(inplace=True)

```



```

72     self.pool2 = nn.MaxPool2d(kernel_size=2, stride=2)
73     self.fc1 = nn.Linear(in_features=64*5*5, out_features=1280)
74     self.relu3 = nn.ReLU(inplace=True)
75     self.dropout = nn.Dropout(p=0.5)
76     self.fc2 = nn.Linear(in_features=1280, out_features=10)
77     self.softmax = nn.Softmax(dim=1)
78
79     def forward(self, x):
80         x = self.conv1(x)
81         x = self.relu1(x)
82         x = self.pool1(x)
83         x = self.conv2(x)
84         x = self.bn2(x)
85         x = self.relu2(x)
86         x = self.pool2(x)
87         x = x.view(-1, 64*5*5)
88         x = self.fc1(x)
89         x = self.relu3(x)
90         x = self.dropout(x)
91         x = self.fc2(x)
92         x = self.softmax(x)
93         return x
94
95 # SCNNB-b
96 class SCNNB_b(nn.Module):
97     def __init__(self):
98         super(SCNNB_b, self).__init__()
99         self.conv1 = nn.Conv2d(in_channels=1, out_channels=32, kernel_size=3)
100        self.relu1 = nn.ReLU(inplace=True)
101        self.pool1 = nn.MaxPool2d(kernel_size=2, stride=2)
102        self.conv2 = nn.Conv2d(in_channels=32, out_channels=64, kernel_size=3)
103        self.relu2 = nn.ReLU(inplace=True)
104        self.pool2 = nn.MaxPool2d(kernel_size=2, stride=2)
105        self.fc1 = nn.Linear(in_features=64*5*5, out_features=1280)
106        self.relu3 = nn.ReLU(inplace=True)
107        self.dropout = nn.Dropout(p=0.5)
108        self.fc2 = nn.Linear(in_features=1280, out_features=10)
109        self.softmax = nn.Softmax(dim=1)
110
111        def forward(self, x):
112            x = self.conv1(x)
113            x = self.relu1(x)
114            x = self.pool1(x)
115            x = self.conv2(x)
116            x = self.relu2(x)

```

```

117     x = self.pool2(x)
118     x = x.view(-1, 64*5*5)
119     x = self.fc1(x)
120     x = self.relu3(x)
121     x = self.dropout(x)
122     x = self.fc2(x)
123     x = self.softmax(x)
124     return x
125
126 # define the models
127 models = [SCNNB(), SCNNB_a(), SCNNB_b()]
128 colors = ['blue', 'green', 'red']
129
130 # define the training loop
131 train_losses = [[] for _ in range(len(models))]
132 train_accs = [[] for _ in range(len(models))]
133 val_losses = [[] for _ in range(len(models))]
134 val_accs = [[] for _ in range(len(models))]
135 test_accs = [[] for _ in range(len(models))]
136
137 for i, model in enumerate(models):
138     print(f"Training model {i+1}")
139     for epoch in range(num_epochs):
140
141         criterion = nn.CrossEntropyLoss()
142         optimizer = optim.SGD(model.parameters(), lr=0.02, momentum=0.9, weight_decay=0.000005)
143
144         train_loss = 0.0
145         train_total = 0
146         train_correct = 0
147         val_loss = 0.0
148         val_total = 0
149         val_correct = 0
150         train_acc = 0.0
151         val_acc = 0.0
152
153         # train the model on training set
154         model.train()
155         for j, data in enumerate(trainloader_mnist, 0):
156             inputs, labels = data
157             optimizer.zero_grad()
158             outputs = model(inputs)
159             loss = criterion(outputs, labels)
160             loss.backward()
161             optimizer.step()

```

```

162         train_loss += loss.item()
163     _, predicted = torch.max(outputs.data, 1)
164     train_acc += (predicted == labels).sum().item()
165
166
167     # evaluate the model on validation set
168     model.eval()
169     with torch.no_grad():
170         for j, data in enumerate(valloader_mnist, 0):
171             inputs, labels = data
172             outputs = model(inputs)
173             loss = criterion(outputs, labels)
174
175             val_loss += loss.item()
176             _, predicted = torch.max(outputs.data, 1)
177             val_acc += (predicted == labels).sum().item()
178
179     # calculate average loss and accuracy for training and validation sets
180     train_loss /= len(trainloader_mnist)
181     train_acc /= len(trainset_mnist)
182     val_loss /= len(valloader_mnist)
183     val_acc /= len(valset_mnist)
184     train_losses[i].append(train_loss)
185     train_accs[i].append(train_acc)
186     val_losses[i].append(val_loss)
187     val_accs[i].append(val_acc)
188
189     # evaluate the model on test set
190     model.eval()
191     with torch.no_grad():
192         test_acc = 0.0
193         for j, data in enumerate(testloader_mnist, 0):
194             inputs, labels = data
195             outputs = model(inputs)
196             _, predicted = torch.max(outputs.data, 1)
197             test_acc += (predicted == labels).sum().item()
198
199         test_acc /= len(testset_mnist)
200         test_accs[i].append(test_acc)
201
202     # print the loss and accuracy for each epoch
203     print(f'Model {i+1} | Epoch {epoch+1:3d} | Train Loss: {train_loss:.4f} | Train Acc: {train_acc:.4f} | Val Loss: {val_loss:.4f} | Val Acc: {val_acc:.4f} | Test Acc: {test_acc:.4f}')
204

```

```

205
206 # Plot the results for all models in the same graph
207 plt.figure(figsize=(8, 6))
208 for i in range(len(models)):
209     plt.plot(train_losses[i], color=colors[i], label=f'Train Loss ({models[i].__class__.__name__
210         })')
211     plt.plot(val_losses[i], '--', color=colors[i], label=f'Val Loss ({models[i].__class__
212         .__name__})')
213
214 plt.xlabel('Epoch')
215 plt.ylabel('Loss')
216 plt.legend()
217 plt.title('Training and Validation Loss over Epochs')
218 plt.savefig("Q21figure1.pdf")
219 plt.show()
220
221 # Plot the results for all models in the same graph
222 plt.figure(figsize=(8, 6))
223 for i in range(len(models)):
224     plt.plot(train_accs[i], color=colors[i], label=f'Train Acc ({models[i].__class__.__name__})')
225     plt.plot(val_accs[i], '--', color=colors[i], label=f'Val Acc ({models[i].__class__.__name__
226         })')
227
228 plt.xlabel('Epoch')
229 plt.ylabel('Accuracy')
230 plt.legend()
231 plt.title('Training and Validation Accuracy over Epochs')
232 plt.savefig("Q21figure2.pdf")
233 plt.show()
234
235 # Plot the results for all models in the same graph
236 plt.figure(figsize=(8, 6))
237 for i in range(len(models)):
238     plt.plot(test_accs[i], color=colors[i], label=f'Test Acc ({models[i].__class__.__name__})')
239     # plt.plot(val_accs[i], '--', color=colors[i], label=f'Val Loss ({models[i].__class__
240         .__name__})')
241
242 plt.xlabel('Epoch')
243 plt.ylabel('Loss')
244 plt.legend()
245 plt.title('Test Accuracy over Epochs')
246 plt.savefig("Q21figure3.pdf")
247 plt.show()

```

```

246 # evaluate the model on test set
247 for i, model in enumerate(models):
248     print(f"Test model {i+1}")
249     model.eval()
250     with torch.no_grad():
251         test_acc = 0.0
252         all_targets = []
253         all_predictions = []
254         for i, data in enumerate(testloader_mnist, 0):
255             inputs, labels = data
256             outputs = model(inputs)
257             _, predicted = torch.max(outputs.data, 1)
258             test_acc += (predicted == labels).sum().item()
259             all_targets.extend(labels.cpu().numpy())
260             all_predictions.extend(predicted.cpu().numpy())
261
262 # calculate accuracy on test set
263 test_acc /= len(testset_mnist)
264 test_accs.append(test_acc)
265
266 # print classification report
267 print('Classification report:')
268 print(classification_report(all_targets, all_predictions))
269
270 # plot confusion matrix
271 cm = confusion_matrix(all_targets, all_predictions)
272 ax = sns.heatmap(cm, annot=True, fmt='g', cmap='Blues', cbar=False)
273 ax.set_xlabel('Predicted labels')
274 ax.set_ylabel('True labels')
275 ax.xaxis.set_ticklabels(['0', '1', '2', '3', '4', '5', '6', '7', '8', '9'])
276 ax.yaxis.set_ticklabels(['0', '1', '2', '3', '4', '5', '6', '7', '8', '9'])
277 plt.title(f'Confusion Matrix ({model.__class__.__name__})')
278 plt.savefig(f"confusion_matrix_{model.__class__.__name__}.pdf")
279 plt.show()

```

## ۴.۱ نتایج پیاده‌سازی

## ۱.۴.۱ پاسخ قسمت الف

با توجه به برنامه ۳ و توضیحاتی که درخصوص آن در بخش ۱.۱ ارائه شد، نتایج مربوط به ارزیابی (آزمون دادگان ارزیابی) هر مجموعه داده به صورت زیر است.

- مجموعه داده MNIST: نتایج نمودار دقت مربوط به این مجموعه داده در شکل ۱ و نتایج مربوط به ماتریس درهم‌ریختگی در شکل ۲ نشان داده شده است. نتایج کلی طبقه‌بندی هم به شرح زیر است:

1	Test model 1
2	Classification report:
3	precision recall f1-score support
4	
5	0 0.99 1.00 0.99 980
6	1 1.00 1.00 1.00 1135
7	2 0.99 1.00 0.99 1032
8	3 0.99 1.00 0.99 1010
9	4 0.99 1.00 1.00 982
10	5 1.00 0.99 0.99 892
11	6 1.00 0.99 0.99 958
12	7 0.99 0.99 0.99 1028
13	8 0.99 0.99 0.99 974
14	9 0.99 0.99 0.99 1009
15	
16	accuracy 0.99 10000
17	macro avg 0.99 0.99 0.99 10000
18	weighted avg 0.99 0.99 0.99 10000
19	
20	Test model 2
21	Classification report:
22	precision recall f1-score support
23	
24	0 0.99 1.00 0.99 980
25	1 0.99 1.00 1.00 1135
26	2 0.99 0.99 0.99 1032
27	3 0.99 0.99 0.99 1010
28	4 0.99 1.00 0.99 982
29	5 0.99 0.99 0.99 892
30	6 0.99 0.99 0.99 958
31	7 0.99 0.99 0.99 1028
32	8 0.99 0.99 0.99 974
33	9 0.99 0.98 0.99 1009
34	
35	accuracy 0.99 10000
36	macro avg 0.99 0.99 0.99 10000
37	weighted avg 0.99 0.99 0.99 10000
38	
39	Test model 3
40	Classification report:
41	precision recall f1-score support
42	
43	0 0.99 1.00 0.99 980
44	1 1.00 1.00 1.00 1135
45	2 0.99 1.00 1.00 1032
46	3 0.99 1.00 0.99 1010
47	4 0.99 0.99 0.99 982
48	5 0.99 0.99 0.99 892
49	6 0.99 0.99 0.99 958
50	7 0.99 0.99 0.99 1028
51	8 1.00 0.99 0.99 974
52	9 0.99 0.99 0.99 1009
53	
54	accuracy 0.99 10000
55	macro avg 0.99 0.99 0.99 10000
56	weighted avg 0.99 0.99 0.99 10000

- مجموعه داده Fashion-MNIST: نتایج نمودار دقت مربوط به این مجموعه داده در شکل ۳ و نتایج مربوط به ماتریس درهم‌ریختگی در شکل ۴ نشان داده شده است. نتایج کلی طبقه‌بندی هم به شرح زیر است:

```

1 Test model 1
2 Classification report:
3           precision    recall  f1-score   support
4
5      0           0.86      0.88      0.87      1000
6      1           0.99      0.99      0.99      1000
7      2           0.88      0.88      0.88      1000
8      3           0.92      0.92      0.92      1000
9      4           0.86      0.87      0.87      1000
10     5           0.98      0.97      0.98      1000
11     6           0.78      0.75      0.76      1000
12     7           0.95      0.98      0.96      1000
13     8           0.98      0.98      0.98      1000
14     9           0.97      0.96      0.97      1000
15
16 accuracy              0.92      10000
17 macro avg              0.92      10000
18 weighted avg           0.92      10000
19
20 Test model 2
21 Classification report:
22           precision    recall  f1-score   support
23
24     0           0.84      0.89      0.86      1000
25     1           0.99      0.98      0.99      1000
26     2           0.83      0.92      0.87      1000
27     3           0.91      0.93      0.92      1000
28     4           0.88      0.85      0.87      1000
29     5           0.98      0.98      0.98      1000
30     6           0.82      0.70      0.75      1000
31     7           0.96      0.96      0.96      1000
32     8           0.97      0.99      0.98      1000
33     9           0.97      0.97      0.97      1000
34
35 accuracy              0.92      10000
36 macro avg              0.92      10000
37 weighted avg           0.92      10000
38
39 Test model 3
40 Classification report:
41           precision    recall  f1-score   support
42
43     0           0.88      0.86      0.87      1000
44     1           0.99      0.98      0.99      1000
45     2           0.87      0.89      0.88      1000
46     3           0.92      0.92      0.92      1000
47     4           0.86      0.88      0.87      1000
48     5           0.98      0.98      0.98      1000
49     6           0.78      0.77      0.77      1000
50     7           0.95      0.97      0.96      1000
51     8           0.99      0.99      0.99      1000
52     9           0.98      0.96      0.97      1000
53
54 accuracy              0.92      10000
55 macro avg              0.92      10000
56 weighted avg           0.92      10000

```

- مجموعه داده CIFAR10 - آزمایش اول: نتایج نمودار دقت مربوط به این مجموعه داده در شکل ۵ و نتایج مربوط به ماتریس درهم‌ریختگی در شکل ۶ نشان داده شده است. نتایج کلی طبقه‌بندی هم به شرح زیر است:

```

1 Test model 1
2 Classification report:
3           precision    recall  f1-score   support
4
5      0           0.78      0.73      0.76      1000

```

```

6      1      0.90      0.82      0.86      1000
7      2      0.59      0.61      0.60      1000
8      3      0.55      0.52      0.53      1000
9      4      0.68      0.69      0.69      1000
10     5      0.64      0.66      0.65      1000
11     6      0.76      0.82      0.79      1000
12     7      0.79      0.77      0.78      1000
13     8      0.82      0.84      0.83      1000
14     9      0.80      0.83      0.81      1000
15
16     accuracy                0.73      10000
17     macro avg              0.73      0.73      0.73      10000
18     weighted avg           0.73      0.73      0.73      10000
19
20 Test model 2
21 Classification report:
22           precision    recall  f1-score   support
23
24      0       0.76      0.77      0.76      1000
25      1       0.85      0.83      0.84      1000
26      2       0.62      0.61      0.61      1000
27      3       0.55      0.54      0.54      1000
28      4       0.67      0.70      0.69      1000
29      5       0.64      0.64      0.64      1000
30      6       0.77      0.79      0.78      1000
31      7       0.81      0.75      0.78      1000
32      8       0.80      0.86      0.83      1000
33      9       0.81      0.79      0.80      1000
34
35     accuracy                0.73      10000
36     macro avg              0.73      0.73      0.73      10000
37     weighted avg           0.73      0.73      0.73      10000
38
39 Test model 3
40 Classification report:
41           precision    recall  f1-score   support
42
43      0       0.75      0.72      0.73      1000
44      1       0.79      0.83      0.81      1000
45      2       0.56      0.60      0.58      1000
46      3       0.55      0.47      0.50      1000
47      4       0.63      0.64      0.64      1000
48      5       0.61      0.58      0.60      1000
49      6       0.75      0.79      0.77      1000
50      7       0.77      0.72      0.75      1000
51      8       0.75      0.83      0.79      1000
52      9       0.79      0.76      0.77      1000
53
54     accuracy                0.69      10000
55     macro avg              0.69      0.69      0.69      10000
56     weighted avg           0.69      0.69      0.69      10000

```

- مجموعه داده CIFAR10 - آزمایش دوم: نتایج نمودار دقت مربوط به این مجموعه داده در شکل ۷ و نتایج مربوط به ماتریس درهم‌ریختگی در شکل ۸ نشان داده شده است. نتایج کلی طبقه‌بندی هم به شرح زیر است:

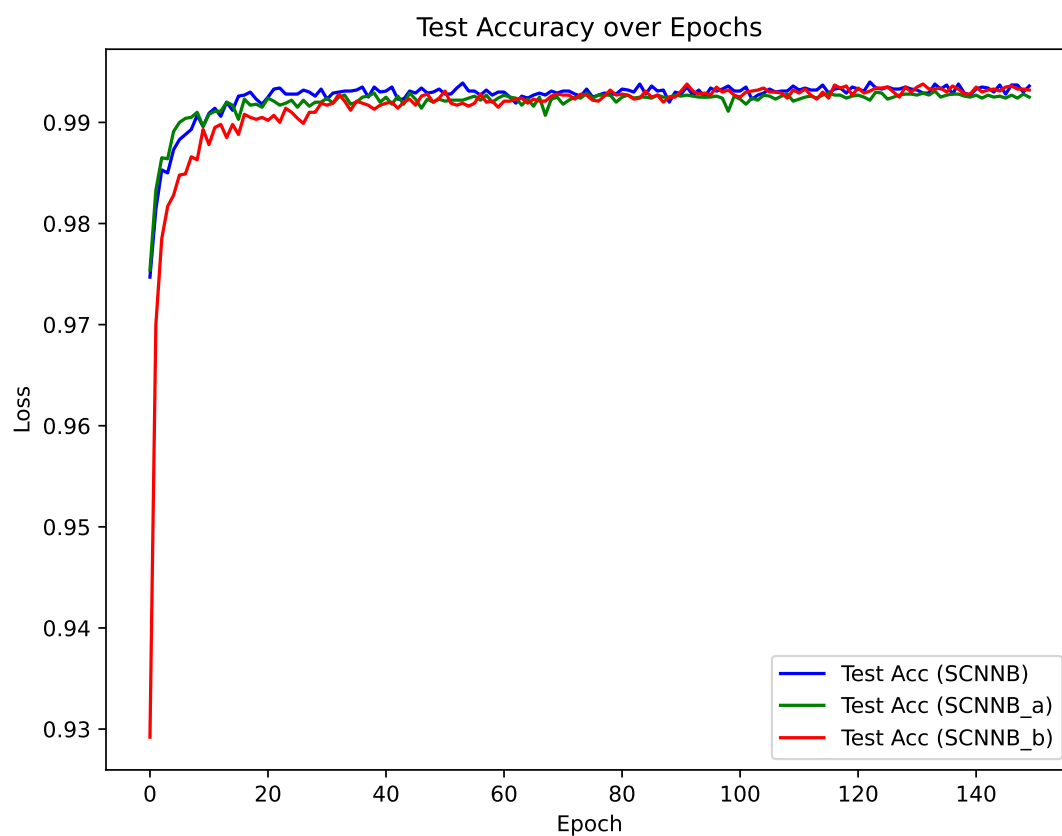
```

1 Classification report:
2           precision    recall  f1-score   support
3
4      0       0.80      0.71      0.75      1000
5      1       0.87      0.84      0.86      1000
6      2       0.58      0.65      0.61      1000
7      3       0.55      0.54      0.55      1000
8      4       0.70      0.68      0.69      1000
9      5       0.65      0.64      0.65      1000
10     6       0.78      0.80      0.79      1000
11     7       0.82      0.78      0.80      1000
12     8       0.77      0.88      0.82      1000
13     9       0.82      0.79      0.80      1000
14

```



15	accuracy			0.73	10000
16	macro avg	0.73	0.73	0.73	10000
17	weighted avg	0.73	0.73	0.73	10000



شکل ۱: نمودار دقت پیاده‌سازی (مجموعه داده MNIST).

0	978	0	0	0	0	0	1	1	0	0
1	0	1131	0	1	0	1	1	1	0	0
2	0	1	1028	0	0	0	0	3	0	0
3	0	0	0	1007	0	3	0	0	0	0
4	0	0	1	0	973	0	2	0	1	5
5	2	0	1	5	0	883	1	0	0	0
6	5	2	0	0	1	2	947	0	1	0
7	0	1	2	1	0	0	0	1022	1	1
8	1	0	2	2	0	0	0	0	967	2
9	0	1	0	0	6	4	0	1	1	996
	0	1	2	3	4	5	6	7	8	9

SCNNB-b (ج)

0	976	1	1	0	0	0	1	1	0	0
1	0	1133	1	0	0	0	1	0	0	0
2	1	1	1023	0	1	0	0	5	1	0
3	0	1	2	1003	0	2	0	1	1	0
4	0	0	0	0	978	0	2	0	0	2
5	1	0	0	5	0	885	1	0	0	0
6	3	1	0	0	1	4	948	0	1	0
7	0	2	3	0	0	0	0	1022	0	1
8	2	0	2	1	0	0	0	1	966	2
9	0	1	0	1	7	2	0	4	3	991
	0	1	2	3	4	5	6	7	8	9

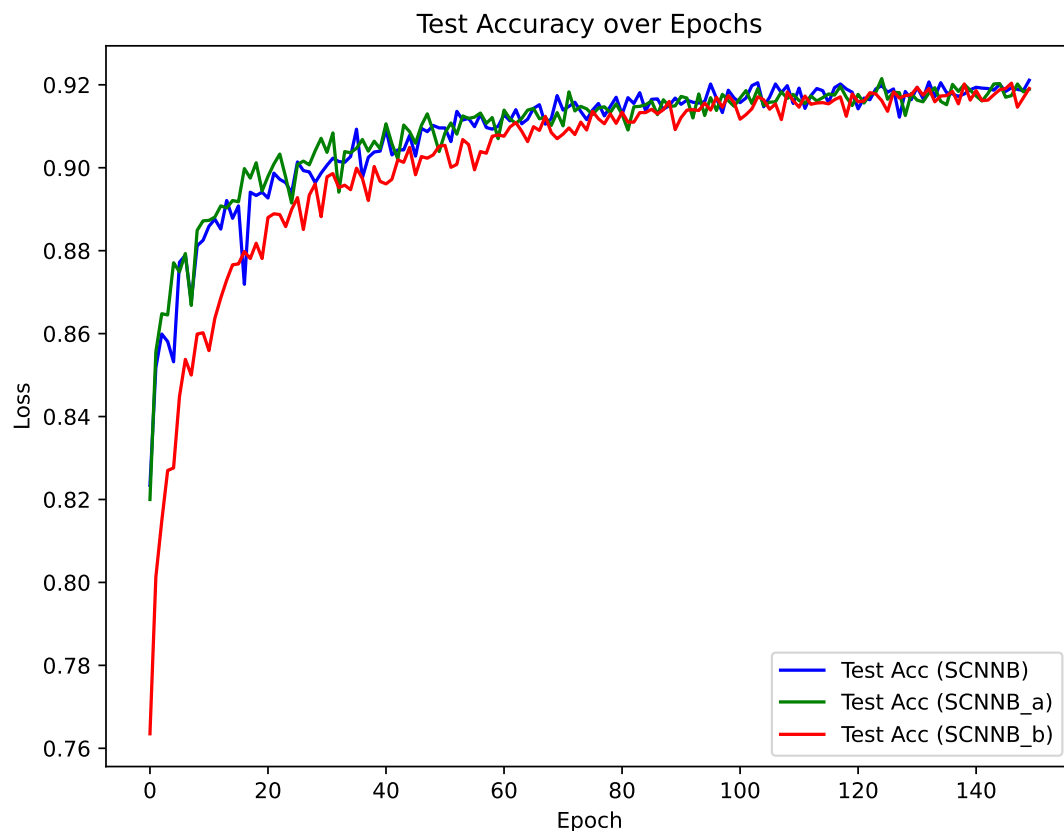
SCNNB-a (ب)

0	978	0	0	0	0	0	1	1	0	0
1	0	1132	1	1	0	0	1	0	0	0
2	1	0	1027	1	0	0	0	3	0	0
3	0	0	1	1008	0	0	0	0	1	0
4	0	0	0	0	978	0	1	0	0	3
5	1	0	1	6	0	883	1	0	0	0
6	4	1	0	0	1	3	947	0	2	0
7	0	2	5	1	0	0	0	1018	0	2
8	1	0	1	0	0	0	0	2	968	2
9	1	0	0	1	4	1	0	3	2	997
	0	1	2	3	4	5	6	7	8	9

SCNNB (ا)

شکل ۲: ماتریس درهم‌ریختگی پیاده‌سازی (مجموعه داده MNIST).

در نهایت نتیجه ارزیابی مدل‌ها روی داده‌های آزمون را در جدولی مانند جدول یک مقاله؛ یعنی در جدول ۱ نشان می‌دهیم.



شکل ۳: نمودار دقت پیاده‌سازی (مجموعه داده Fashion-MNIST).

	0	1	2	3	4	5	6	7	8	9
0	864	0	22	20	3	2	86	0	3	0
1	2	982	1	8	2	0	3	0	2	0
2	16	0	890	10	35	0	47	0	2	0
3	11	2	10	921	29	1	25	0	1	0
4	7	1	1	50	22	876	0	49	0	1
5	0	0	0	1	0	977	0	13	0	9
6	90	1	53	20	68	0	766	0	2	0
7	0	0	0	0	0	0	15	0	972	0
8	1	1	1	4	1	2	2	2	986	0
9	0	0	0	0	0	0	3	1	38	0
	0	1	2	3	4	5	6	7	8	9

SCNNB-b (ج)

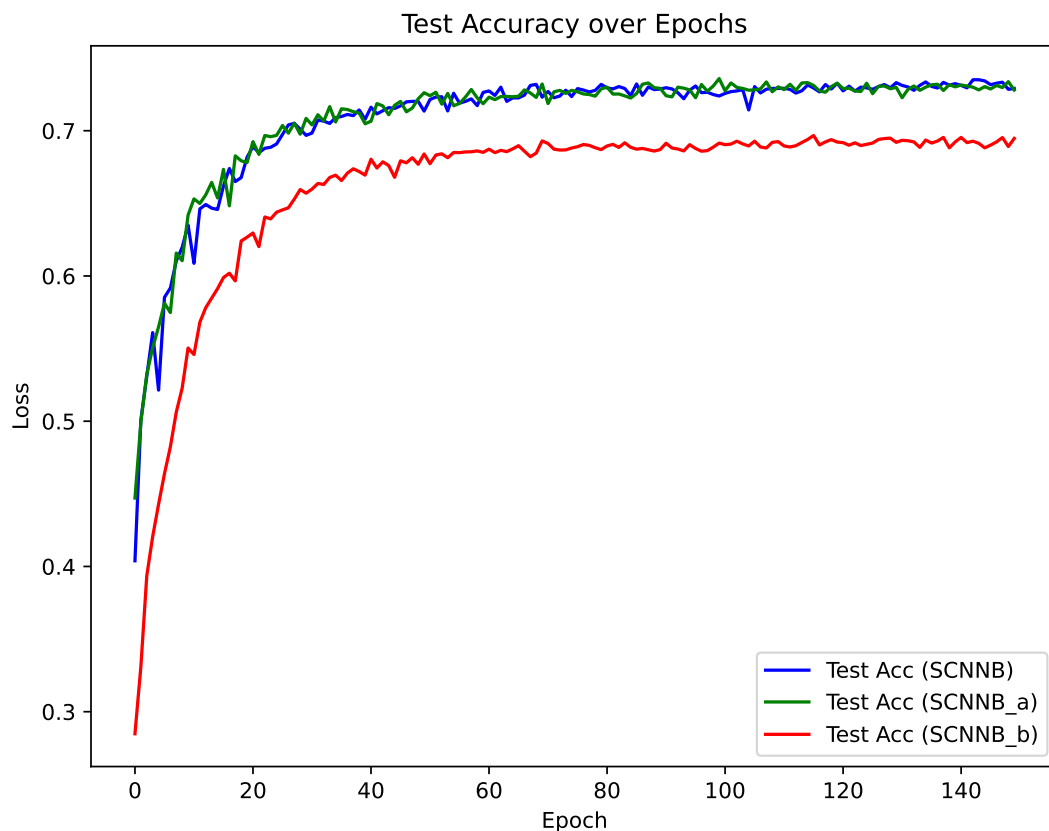
	0	1	2	3	4	5	6	7	8	9
0	887	2	20	14	0	3	63	0	11	0
1	2	981	1	11	2	0	1	0	2	0
2	19	0	919	11	30	0	19	0	2	0
3	14	2	12	929	20	0	20	0	3	0
4	0	0	1	73	26	948	0	50	0	2
5	1	0	0	1	0	0	980	0	10	1
6	131	1	78	23	59	0	699	0	9	0
7	0	0	0	0	0	12	0	964	0	24
8	0	1	0	4	1	3	1	2	980	0
9	0	0	0	0	0	7	1	23	0	969
True labels	0	1	2	3	4	5	6	7	8	9
Predicted labels	0	1	2	3	4	5	6	7	8	9

SCNNB-a (ب)

	0	1	2	3	4	5	6	7	8	9
0	884	1	21	15	4	3	66	0	6	0
1	1	985	2	8	0	0	3	0	1	0
2	14	0	870	11	47	0	56	0	2	0
3	8	5	10	925	26	0	24	0	2	0
4	1	1	34	20	887	1	54	0	2	0
5	0	0	0	0	0	981	0	11	0	8
6	106	2	50	24	53	0	755	1	9	0
7	0	0	0	0	0	7	0	979	0	14
8	1	1	0	3	1	4	1	5	983	1
9	0	0	0	0	0	6	0	36	0	958
	0	1	2	3	4	5	6	7	8	9

SCNNB (ا)

شکل ۴: ماتریس درهم‌ریختگی پیاده‌سازی (مجموعه داده Fashion-MNIST).



شکل ۵: نمودار دقت پیاده‌سازی (مجموعه داده CIFAR10 - آزمایش اول).

Confusion Matrix (SCNNB\_b)

0	721	17	75	10	30	9	13	14	83	28
1	834	12	13	7	5	18	1	39	51	
2	61	15	500	54	101	56	49	28	28	8
3	31	25	93	470	82	150	68	34	20	27
4	31	19	86	53	642	37	47	57	18	10
5	15	12	85	152	43	585	38	44	13	13
6	6	17	55	47	25	28	786	7	14	15
7	20	5	34	35	60	64	11	720	17	34
8	38	37	21	12	10	21	7	6	831	17
9	23	79	10	16	17	11	13	21	52	758
	0	1	2	3	4	5	6	7	8	9

SCNNB-b (ج)

Confusion Matrix (SCNNB\_a)

0	770	15	60	10	23	14	11	6	64	27
1	834	7	12	6	5	16	2	31	70	
2	63	6	609	73	90	63	53	18	18	7
3	29	14	77	536	73	136	59	28	20	28
4	23	7	57	61	703	42	43	50	9	5
5	13	5	68	144	49	641	23	39	11	7
6	9	7	51	53	33	28	792	4	15	8
7	12	9	35	52	54	54	9	747	10	18
8	48	25	14	12	6	7	6	9	856	17
9	31	62	9	18	13	4	14	20	37	792
	0	1	2	3	4	5	6	7	8	9

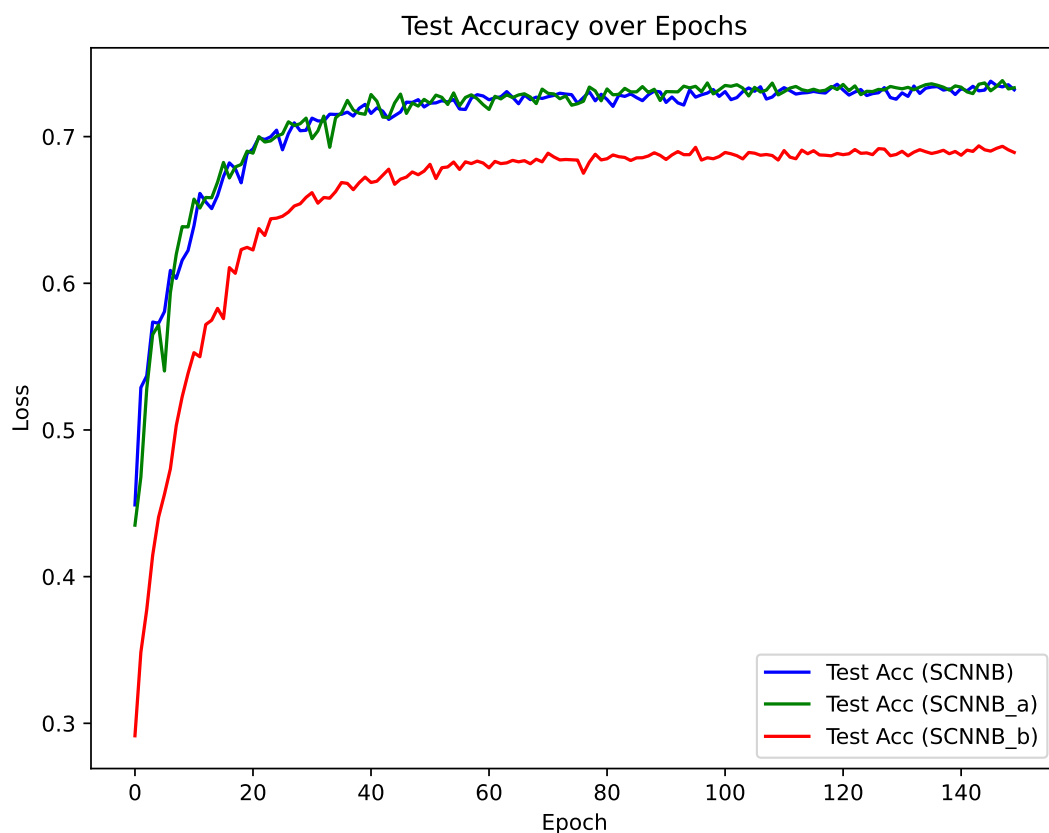
SCNNB-a (ب)

Confusion Matrix (SCNNB)

0	732	12	81	21	30	10	12	11	55	36
1	820	12	7	6	8	18	5	34	73	
2	50	6	610	71	89	70	50	29	18	7
3	25	4	86	516	66	151	69	37	17	29
4	19	3	67	62	695	41	49	52	6	6
5	12	1	65	130	44	660	32	44	5	7
6	11	4	40	50	25	21	816	7	16	10
7	3	3	33	45	41	63	12	770	8	22
8	51	15	24	11	12	6	8	7	842	24
9	13	48	12	17	13	8	12	16	31	830
	0	1	2	3	4	5	6	7	8	9

SCNNB (ا)

شکل ۶: ماتریس درهم‌ریختگی پیاده‌سازی (مجموعه داده CIFAR10 - آزمایش اول).



شکل ۷: نمودار دقت پیاده‌سازی (مجموعه داده CIFAR10 - آزمایش دوم).

Confusion Matrix (SCNNB\_b)

0	745	15	66	26	23	6	19	11	54	35
1	21	784	10	17	11	9	19	2	33	94
2	75	11	578	67	93	58	57	28	18	15
3	35	15	83	514	73	131	65	34	17	33
4	39	6	74	87	616	44	54	56	17	7
5	16	10	81	166	35	599	30	42	11	10
6	14	11	38	61	31	26	787	3	16	13
7	20	7	37	56	61	58	9	713	6	33
8	77	41	11	20	8	9	8	6	781	39
9	33	63	9	21	12	16	11	24	36	775
	0	1	2	3	4	5	6	7	8	9

True labels

Predicted labels

شکل ۸: ماتریس درهم‌ریختگی پیاده‌سازی (مجموعه داده CIFAR10 - آزمایش دوم).

جدول ۱: نتیجه ارزیابی مدل‌ها روی داده‌های آزمون

Our methods	MNIST classification test accuracy (%)	Fashion-MNIST classification test accuracy (%)	CIFAR10 classification test accuracy (%)
SCNNB-a	99.31	92.15	73.82
SCNNB-b	99.31	92.04	69.37
SCNNB	99.40	92.15	73.78

## ۲.۴.۱ پاسخ قسمت ب

با توجه به برنامه ۳ و توضیحاتی که درخصوص آن در بخش ۱.۱ ارائه شد، نتایج دقت و تابع اتلاف مربوط به آموزش روی داده‌های آموزشی و اعتبارسنجی هر مجموعه داده به صورت زیر است.

- مجموعه داده MNIST: نتایج نمودار دقت و اتلاف مربوط به این مجموعه داده در شکل ۹ نشان داده شده است و بخشی از نتایج نمایش داده شده برای هر اپیک به صورت زیر است.

1	Training model 1						
2	Model 1	Epoch 1	Train Loss: 1.5953	Train Acc: 0.8813	Val Loss: 1.4945	Val Acc: 0.9715	Test Acc: 0.9747
3	Model 1	Epoch 2	Train Loss: 1.4918	Train Acc: 0.9737	Val Loss: 1.4857	Val Acc: 0.9786	Test Acc: 0.9814
4	Model 1	Epoch 3	Train Loss: 1.4834	Train Acc: 0.9809	Val Loss: 1.4801	Val Acc: 0.9827	Test Acc: 0.9853
5	Model 1	Epoch 4	Train Loss: 1.4793	Train Acc: 0.9845	Val Loss: 1.4801	Val Acc: 0.9835	Test Acc: 0.9850
6	Model 1	Epoch 5	Train Loss: 1.4767	Train Acc: 0.9866	Val Loss: 1.4797	Val Acc: 0.9824	Test Acc: 0.9873
7	Model 1	Epoch 6	Train Loss: 1.4743	Train Acc: 0.9886	Val Loss: 1.4754	Val Acc: 0.9874	Test Acc: 0.9883
8	Model 1	Epoch 7	Train Loss: 1.4736	Train Acc: 0.9895	Val Loss: 1.4751	Val Acc: 0.9874	Test Acc: 0.9888
9	Model 1	Epoch 8	Train Loss: 1.4717	Train Acc: 0.9908	Val Loss: 1.4747	Val Acc: 0.9876	Test Acc: 0.9893
10	Model 1	Epoch 9	Train Loss: 1.4708	Train Acc: 0.9917	Val Loss: 1.4734	Val Acc: 0.9894	Test Acc: 0.9907
11	Model 1	Epoch 10	Train Loss: 1.4701	Train Acc: 0.9925	Val Loss: 1.4733	Val Acc: 0.9884	Test Acc: 0.9896
12	Model 1	Epoch 11	Train Loss: 1.4699	Train Acc: 0.9926	Val Loss: 1.4730	Val Acc: 0.9892	Test Acc: 0.9909
13							
14							
15							
16	Model 1	Epoch 140	Train Loss: 1.4621	Train Acc: 0.9992	Val Loss: 1.4689	Val Acc: 0.9927	Test Acc: 0.9927
17	Model 1	Epoch 141	Train Loss: 1.4621	Train Acc: 0.9991	Val Loss: 1.4696	Val Acc: 0.9917	Test Acc: 0.9933
18	Model 1	Epoch 142	Train Loss: 1.4621	Train Acc: 0.9991	Val Loss: 1.4689	Val Acc: 0.9928	Test Acc: 0.9935
19	Model 1	Epoch 143	Train Loss: 1.4621	Train Acc: 0.9991	Val Loss: 1.4690	Val Acc: 0.9923	Test Acc: 0.9934
20	Model 1	Epoch 144	Train Loss: 1.4621	Train Acc: 0.9991	Val Loss: 1.4689	Val Acc: 0.9928	Test Acc: 0.9930
21	Model 1	Epoch 145	Train Loss: 1.4621	Train Acc: 0.9991	Val Loss: 1.4688	Val Acc: 0.9928	Test Acc: 0.9937
22	Model 1	Epoch 146	Train Loss: 1.4621	Train Acc: 0.9991	Val Loss: 1.4689	Val Acc: 0.9924	Test Acc: 0.9928
23	Model 1	Epoch 147	Train Loss: 1.4621	Train Acc: 0.9991	Val Loss: 1.4690	Val Acc: 0.9920	Test Acc: 0.9937
24	Model 1	Epoch 148	Train Loss: 1.4621	Train Acc: 0.9991	Val Loss: 1.4690	Val Acc: 0.9923	Test Acc: 0.9937
25	Model 1	Epoch 149	Train Loss: 1.4620	Train Acc: 0.9992	Val Loss: 1.4690	Val Acc: 0.9924	Test Acc: 0.9930
26	Model 1	Epoch 150	Train Loss: 1.4620	Train Acc: 0.9992	Val Loss: 1.4690	Val Acc: 0.9923	Test Acc: 0.9936
27							
28	Training model 2						
29	Model 2	Epoch 1	Train Loss: 1.6008	Train Acc: 0.8747	Val Loss: 1.4940	Val Acc: 0.9724	Test Acc: 0.9754
30	Model 2	Epoch 2	Train Loss: 1.4902	Train Acc: 0.9750	Val Loss: 1.4839	Val Acc: 0.9808	Test Acc: 0.9833
31	Model 2	Epoch 3	Train Loss: 1.4824	Train Acc: 0.9818	Val Loss: 1.4803	Val Acc: 0.9833	Test Acc: 0.9865
32	Model 2	Epoch 4	Train Loss: 1.4786	Train Acc: 0.9847	Val Loss: 1.4799	Val Acc: 0.9827	Test Acc: 0.9864
33	Model 2	Epoch 5	Train Loss: 1.4761	Train Acc: 0.9872	Val Loss: 1.4771	Val Acc: 0.9859	Test Acc: 0.9891
34	Model 2	Epoch 6	Train Loss: 1.4745	Train Acc: 0.9888	Val Loss: 1.4763	Val Acc: 0.9863	Test Acc: 0.9900
35	Model 2	Epoch 7	Train Loss: 1.4730	Train Acc: 0.9898	Val Loss: 1.4757	Val Acc: 0.9868	Test Acc: 0.9904
36	Model 2	Epoch 8	Train Loss: 1.4714	Train Acc: 0.9914	Val Loss: 1.4751	Val Acc: 0.9874	Test Acc: 0.9905
37	Model 2	Epoch 9	Train Loss: 1.4705	Train Acc: 0.9921	Val Loss: 1.4741	Val Acc: 0.9887	Test Acc: 0.9910
38	Model 2	Epoch 10	Train Loss: 1.4698	Train Acc: 0.9928	Val Loss: 1.4741	Val Acc: 0.9886	Test Acc: 0.9896
39							
40							
41							
42	Model 2	Epoch 140	Train Loss: 1.4620	Train Acc: 0.9992	Val Loss: 1.4699	Val Acc: 0.9913	Test Acc: 0.9925
43	Model 2	Epoch 141	Train Loss: 1.4620	Train Acc: 0.9992	Val Loss: 1.4696	Val Acc: 0.9914	Test Acc: 0.9927
44	Model 2	Epoch 142	Train Loss: 1.4620	Train Acc: 0.9992	Val Loss: 1.4697	Val Acc: 0.9912	Test Acc: 0.9924
45	Model 2	Epoch 143	Train Loss: 1.4620	Train Acc: 0.9992	Val Loss: 1.4698	Val Acc: 0.9910	Test Acc: 0.9927
46	Model 2	Epoch 144	Train Loss: 1.4621	Train Acc: 0.9992	Val Loss: 1.4700	Val Acc: 0.9910	Test Acc: 0.9925

47	Model 2	Epoch 145	Train Loss: 1.4620	Train Acc: 0.9992	Val Loss: 1.4697	Val Acc: 0.9915	Test Acc: 0.9926
48	Model 2	Epoch 146	Train Loss: 1.4621	Train Acc: 0.9992	Val Loss: 1.4698	Val Acc: 0.9913	Test Acc: 0.9924
49	Model 2	Epoch 147	Train Loss: 1.4620	Train Acc: 0.9992	Val Loss: 1.4699	Val Acc: 0.9913	Test Acc: 0.9927
50	Model 2	Epoch 148	Train Loss: 1.4620	Train Acc: 0.9992	Val Loss: 1.4700	Val Acc: 0.9908	Test Acc: 0.9924
51	Model 2	Epoch 149	Train Loss: 1.4620	Train Acc: 0.9992	Val Loss: 1.4700	Val Acc: 0.9909	Test Acc: 0.9928
52	Model 2	Epoch 150	Train Loss: 1.4620	Train Acc: 0.9992	Val Loss: 1.4699	Val Acc: 0.9910	Test Acc: 0.9925
53							
54	Training model 3						
55	Model 3	Epoch 1	Train Loss: 1.7761	Train Acc: 0.6972	Val Loss: 1.5492	Val Acc: 0.9213	Test Acc: 0.9292
56	Model 3	Epoch 2	Train Loss: 1.5205	Train Acc: 0.9464	Val Loss: 1.4979	Val Acc: 0.9655	Test Acc: 0.9700
57	Model 3	Epoch 3	Train Loss: 1.4983	Train Acc: 0.9654	Val Loss: 1.4895	Val Acc: 0.9735	Test Acc: 0.9785
58	Model 3	Epoch 4	Train Loss: 1.4894	Train Acc: 0.9737	Val Loss: 1.4869	Val Acc: 0.9763	Test Acc: 0.9817
59	Model 3	Epoch 5	Train Loss: 1.4856	Train Acc: 0.9774	Val Loss: 1.4844	Val Acc: 0.9784	Test Acc: 0.9828
60	Model 3	Epoch 6	Train Loss: 1.4814	Train Acc: 0.9813	Val Loss: 1.4823	Val Acc: 0.9800	Test Acc: 0.9848
61	Model 3	Epoch 7	Train Loss: 1.4794	Train Acc: 0.9832	Val Loss: 1.4796	Val Acc: 0.9827	Test Acc: 0.9849
62	Model 3	Epoch 8	Train Loss: 1.4778	Train Acc: 0.9845	Val Loss: 1.4774	Val Acc: 0.9848	Test Acc: 0.9866
63	Model 3	Epoch 9	Train Loss: 1.4761	Train Acc: 0.9862	Val Loss: 1.4763	Val Acc: 0.9852	Test Acc: 0.9863
64	Model 3	Epoch 10	Train Loss: 1.4750	Train Acc: 0.9875	Val Loss: 1.4771	Val Acc: 0.9853	Test Acc: 0.9893
65							
66							
67							
68	Model 3	Epoch 140	Train Loss: 1.4623	Train Acc: 0.9989	Val Loss: 1.4696	Val Acc: 0.9922	Test Acc: 0.9928
69	Model 3	Epoch 141	Train Loss: 1.4622	Train Acc: 0.9990	Val Loss: 1.4701	Val Acc: 0.9914	Test Acc: 0.9935
70	Model 3	Epoch 142	Train Loss: 1.4622	Train Acc: 0.9990	Val Loss: 1.4704	Val Acc: 0.9906	Test Acc: 0.9930
71	Model 3	Epoch 143	Train Loss: 1.4623	Train Acc: 0.9989	Val Loss: 1.4705	Val Acc: 0.9909	Test Acc: 0.9932
72	Model 3	Epoch 144	Train Loss: 1.4623	Train Acc: 0.9989	Val Loss: 1.4699	Val Acc: 0.9910	Test Acc: 0.9933
73	Model 3	Epoch 145	Train Loss: 1.4623	Train Acc: 0.9990	Val Loss: 1.4698	Val Acc: 0.9912	Test Acc: 0.9931
74	Model 3	Epoch 146	Train Loss: 1.4622	Train Acc: 0.9990	Val Loss: 1.4703	Val Acc: 0.9910	Test Acc: 0.9935
75	Model 3	Epoch 147	Train Loss: 1.4622	Train Acc: 0.9990	Val Loss: 1.4699	Val Acc: 0.9913	Test Acc: 0.9936
76	Model 3	Epoch 148	Train Loss: 1.4623	Train Acc: 0.9990	Val Loss: 1.4700	Val Acc: 0.9914	Test Acc: 0.9933
77	Model 3	Epoch 149	Train Loss: 1.4622	Train Acc: 0.9991	Val Loss: 1.4697	Val Acc: 0.9914	Test Acc: 0.9933
78	Model 3	Epoch 150	Train Loss: 1.4622	Train Acc: 0.9990	Val Loss: 1.4696	Val Acc: 0.9918	Test Acc: 0.9932

- مجموعه‌داده Fashion-MNIST: نتایج نمودار دقت و اتلاف مربوط به این مجموعه‌داده در شکل ۱۰ نشان داده شده است و بخشی از نتایج نمایش داده‌شده برای هر اپیک به‌صورت زیر است.

1	Training model 1						
2	Model 1	Epoch 1	Train Loss: 1.7288	Train Acc: 0.7462	Val Loss: 1.6414	Val Acc: 0.8242	Test Acc: 0.8234
3	Model 1	Epoch 2	Train Loss: 1.6255	Train Acc: 0.8407	Val Loss: 1.6050	Val Acc: 0.8607	Test Acc: 0.8518
4	Model 1	Epoch 3	Train Loss: 1.6093	Train Acc: 0.8552	Val Loss: 1.5992	Val Acc: 0.8644	Test Acc: 0.8599
5	Model 1	Epoch 4	Train Loss: 1.5977	Train Acc: 0.8654	Val Loss: 1.5980	Val Acc: 0.8629	Test Acc: 0.8581
6	Model 1	Epoch 5	Train Loss: 1.5900	Train Acc: 0.8736	Val Loss: 1.6031	Val Acc: 0.8589	Test Acc: 0.8532
7	Model 1	Epoch 6	Train Loss: 1.5858	Train Acc: 0.8769	Val Loss: 1.5814	Val Acc: 0.8802	Test Acc: 0.8772
8	Model 1	Epoch 7	Train Loss: 1.5810	Train Acc: 0.8816	Val Loss: 1.5772	Val Acc: 0.8847	Test Acc: 0.8789
9	Model 1	Epoch 8	Train Loss: 1.5767	Train Acc: 0.8864	Val Loss: 1.5858	Val Acc: 0.8764	Test Acc: 0.8678
10	Model 1	Epoch 9	Train Loss: 1.5741	Train Acc: 0.8884	Val Loss: 1.5730	Val Acc: 0.8888	Test Acc: 0.8812
11	Model 1	Epoch 10	Train Loss: 1.5696	Train Acc: 0.8926	Val Loss: 1.5725	Val Acc: 0.8894	Test Acc: 0.8825
12							
13							
14							
15	Model 1	Epoch 140	Train Loss: 1.4885	Train Acc: 0.9741	Val Loss: 1.5371	Val Acc: 0.9247	Test Acc: 0.9188
16	Model 1	Epoch 141	Train Loss: 1.4876	Train Acc: 0.9751	Val Loss: 1.5397	Val Acc: 0.9217	Test Acc: 0.9194
17	Model 1	Epoch 142	Train Loss: 1.4883	Train Acc: 0.9741	Val Loss: 1.5369	Val Acc: 0.9244	Test Acc: 0.9192
18	Model 1	Epoch 143	Train Loss: 1.4866	Train Acc: 0.9759	Val Loss: 1.5375	Val Acc: 0.9240	Test Acc: 0.9191
19	Model 1	Epoch 144	Train Loss: 1.4871	Train Acc: 0.9752	Val Loss: 1.5360	Val Acc: 0.9252	Test Acc: 0.9187
20	Model 1	Epoch 145	Train Loss: 1.4866	Train Acc: 0.9757	Val Loss: 1.5362	Val Acc: 0.9252	Test Acc: 0.9201
21	Model 1	Epoch 146	Train Loss: 1.4866	Train Acc: 0.9758	Val Loss: 1.5392	Val Acc: 0.9227	Test Acc: 0.9184
22	Model 1	Epoch 147	Train Loss: 1.4866	Train Acc: 0.9760	Val Loss: 1.5363	Val Acc: 0.9256	Test Acc: 0.9193
23	Model 1	Epoch 148	Train Loss: 1.4867	Train Acc: 0.9758	Val Loss: 1.5362	Val Acc: 0.9249	Test Acc: 0.9189
24	Model 1	Epoch 149	Train Loss: 1.4859	Train Acc: 0.9767	Val Loss: 1.5370	Val Acc: 0.9245	Test Acc: 0.9186
25	Model 1	Epoch 150	Train Loss: 1.4859	Train Acc: 0.9763	Val Loss: 1.5373	Val Acc: 0.9236	Test Acc: 0.9211
26							
27	Training model 2						
28	Model 2	Epoch 1	Train Loss: 1.7368	Train Acc: 0.7408	Val Loss: 1.6379	Val Acc: 0.8316	Test Acc: 0.8200
29	Model 2	Epoch 2	Train Loss: 1.6219	Train Acc: 0.8451	Val Loss: 1.6051	Val Acc: 0.8612	Test Acc: 0.8556
30	Model 2	Epoch 3	Train Loss: 1.6041	Train Acc: 0.8602	Val Loss: 1.5950	Val Acc: 0.8685	Test Acc: 0.8648
31	Model 2	Epoch 4	Train Loss: 1.5951	Train Acc: 0.8692	Val Loss: 1.5970	Val Acc: 0.8646	Test Acc: 0.8645
32	Model 2	Epoch 5	Train Loss: 1.5891	Train Acc: 0.8738	Val Loss: 1.5828	Val Acc: 0.8784	Test Acc: 0.8771
33	Model 2	Epoch 6	Train Loss: 1.5813	Train Acc: 0.8813	Val Loss: 1.5827	Val Acc: 0.8801	Test Acc: 0.8749

34	Model 2	Epoch	7	Train Loss:	1.5777	Train Acc:	0.8848	Val Loss:	1.5751	Val Acc:	0.8878	Test Acc:	0.8793
35	Model 2	Epoch	8	Train Loss:	1.5745	Train Acc:	0.8876	Val Loss:	1.5884	Val Acc:	0.8732	Test Acc:	0.8668
36	Model 2	Epoch	9	Train Loss:	1.5700	Train Acc:	0.8924	Val Loss:	1.5722	Val Acc:	0.8896	Test Acc:	0.8849
37	Model 2	Epoch	10	Train Loss:	1.5674	Train Acc:	0.8949	Val Loss:	1.5696	Val Acc:	0.8920	Test Acc:	0.8872
38	.												
39	.												
40	.												
41	Model 2	Epoch	140	Train Loss:	1.4859	Train Acc:	0.9765	Val Loss:	1.5375	Val Acc:	0.9226	Test Acc:	0.9187
42	Model 2	Epoch	141	Train Loss:	1.4862	Train Acc:	0.9761	Val Loss:	1.5383	Val Acc:	0.9226	Test Acc:	0.9173
43	Model 2	Epoch	142	Train Loss:	1.4853	Train Acc:	0.9773	Val Loss:	1.5382	Val Acc:	0.9228	Test Acc:	0.9161
44	Model 2	Epoch	143	Train Loss:	1.4850	Train Acc:	0.9773	Val Loss:	1.5370	Val Acc:	0.9242	Test Acc:	0.9179
45	Model 2	Epoch	144	Train Loss:	1.4856	Train Acc:	0.9768	Val Loss:	1.5364	Val Acc:	0.9252	Test Acc:	0.9202
46	Model 2	Epoch	145	Train Loss:	1.4857	Train Acc:	0.9765	Val Loss:	1.5384	Val Acc:	0.9234	Test Acc:	0.9203
47	Model 2	Epoch	146	Train Loss:	1.4848	Train Acc:	0.9776	Val Loss:	1.5359	Val Acc:	0.9263	Test Acc:	0.9170
48	Model 2	Epoch	147	Train Loss:	1.4845	Train Acc:	0.9779	Val Loss:	1.5364	Val Acc:	0.9247	Test Acc:	0.9174
49	Model 2	Epoch	148	Train Loss:	1.4843	Train Acc:	0.9780	Val Loss:	1.5375	Val Acc:	0.9229	Test Acc:	0.9202
50	Model 2	Epoch	149	Train Loss:	1.4839	Train Acc:	0.9784	Val Loss:	1.5351	Val Acc:	0.9261	Test Acc:	0.9183
51	Model 2	Epoch	150	Train Loss:	1.4842	Train Acc:	0.9782	Val Loss:	1.5384	Val Acc:	0.9230	Test Acc:	0.9189
52	.												
53	Training model 3												
54	Model 3	Epoch	1	Train Loss:	1.8729	Train Acc:	0.6091	Val Loss:	1.7040	Val Acc:	0.7598	Test Acc:	0.7635
55	Model 3	Epoch	2	Train Loss:	1.6827	Train Acc:	0.7817	Val Loss:	1.6567	Val Acc:	0.8070	Test Acc:	0.8013
56	Model 3	Epoch	3	Train Loss:	1.6583	Train Acc:	0.8051	Val Loss:	1.6434	Val Acc:	0.8187	Test Acc:	0.8148
57	Model 3	Epoch	4	Train Loss:	1.6439	Train Acc:	0.8199	Val Loss:	1.6298	Val Acc:	0.8329	Test Acc:	0.8270
58	Model 3	Epoch	5	Train Loss:	1.6282	Train Acc:	0.8350	Val Loss:	1.6265	Val Acc:	0.8372	Test Acc:	0.8276
59	Model 3	Epoch	6	Train Loss:	1.6187	Train Acc:	0.8440	Val Loss:	1.6093	Val Acc:	0.8532	Test Acc:	0.8448
60	Model 3	Epoch	7	Train Loss:	1.6115	Train Acc:	0.8514	Val Loss:	1.6041	Val Acc:	0.8582	Test Acc:	0.8538
61	Model 3	Epoch	8	Train Loss:	1.6047	Train Acc:	0.8579	Val Loss:	1.6077	Val Acc:	0.8548	Test Acc:	0.8500
62	Model 3	Epoch	9	Train Loss:	1.5993	Train Acc:	0.8635	Val Loss:	1.5936	Val Acc:	0.8692	Test Acc:	0.8599
63	Model 3	Epoch	10	Train Loss:	1.5971	Train Acc:	0.8649	Val Loss:	1.5930	Val Acc:	0.8696	Test Acc:	0.8602
64	.												
65	.												
66	.												
67	Model 3	Epoch	140	Train Loss:	1.4935	Train Acc:	0.9686	Val Loss:	1.5421	Val Acc:	0.9179	Test Acc:	0.9163
68	Model 3	Epoch	141	Train Loss:	1.4927	Train Acc:	0.9696	Val Loss:	1.5390	Val Acc:	0.9208	Test Acc:	0.9186
69	Model 3	Epoch	142	Train Loss:	1.4927	Train Acc:	0.9695	Val Loss:	1.5399	Val Acc:	0.9213	Test Acc:	0.9162
70	Model 3	Epoch	143	Train Loss:	1.4935	Train Acc:	0.9686	Val Loss:	1.5381	Val Acc:	0.9233	Test Acc:	0.9163
71	Model 3	Epoch	144	Train Loss:	1.4925	Train Acc:	0.9696	Val Loss:	1.5388	Val Acc:	0.9227	Test Acc:	0.9176
72	Model 3	Epoch	145	Train Loss:	1.4923	Train Acc:	0.9698	Val Loss:	1.5408	Val Acc:	0.9195	Test Acc:	0.9188
73	Model 3	Epoch	146	Train Loss:	1.4919	Train Acc:	0.9701	Val Loss:	1.5379	Val Acc:	0.9246	Test Acc:	0.9191
74	Model 3	Epoch	147	Train Loss:	1.4921	Train Acc:	0.9703	Val Loss:	1.5387	Val Acc:	0.9215	Test Acc:	0.9204
75	Model 3	Epoch	148	Train Loss:	1.4917	Train Acc:	0.9705	Val Loss:	1.5373	Val Acc:	0.9244	Test Acc:	0.9146
76	Model 3	Epoch	149	Train Loss:	1.4917	Train Acc:	0.9704	Val Loss:	1.5395	Val Acc:	0.9211	Test Acc:	0.9169
77	Model 3	Epoch	150	Train Loss:	1.4910	Train Acc:	0.9711	Val Loss:	1.5376	Val Acc:	0.9234	Test Acc:	0.9191

- مجموعه‌داده CIFAR10 - آزمایش اول: نتایج نمودار دقت و اتلاف مربوط به این مجموعه‌داده در شکل ۱۱ نشان داده شده است و بخشی از نتایج نمایش داده‌شده برای هر اپیک به‌صورت زیر است.

1	Training model 1												
2	Model 1	Epoch	1	Train Loss:	2.1159	Train Acc:	0.3423	Val Loss:	2.0460	Val Acc:	0.4142	Test Acc:	0.4039
3	Model 1	Epoch	2	Train Loss:	1.9903	Train Acc:	0.4700	Val Loss:	1.9523	Val Acc:	0.5109	Test Acc:	0.5009
4	Model 1	Epoch	3	Train Loss:	1.9448	Train Acc:	0.5161	Val Loss:	1.9259	Val Acc:	0.5323	Test Acc:	0.5306
5	Model 1	Epoch	4	Train Loss:	1.9134	Train Acc:	0.5486	Val Loss:	1.8935	Val Acc:	0.5658	Test Acc:	0.5610
6	Model 1	Epoch	5	Train Loss:	1.8937	Train Acc:	0.5675	Val Loss:	1.9353	Val Acc:	0.5216	Test Acc:	0.5214
7	Model 1	Epoch	6	Train Loss:	1.8762	Train Acc:	0.5866	Val Loss:	1.8699	Val Acc:	0.5868	Test Acc:	0.5853
8	Model 1	Epoch	7	Train Loss:	1.8583	Train Acc:	0.6036	Val Loss:	1.8661	Val Acc:	0.5937	Test Acc:	0.5917
9	Model 1	Epoch	8	Train Loss:	1.8401	Train Acc:	0.6238	Val Loss:	1.8508	Val Acc:	0.6107	Test Acc:	0.6103
10	Model 1	Epoch	9	Train Loss:	1.8273	Train Acc:	0.6365	Val Loss:	1.8357	Val Acc:	0.6229	Test Acc:	0.6197
11	Model 1	Epoch	10	Train Loss:	1.8121	Train Acc:	0.6520	Val Loss:	1.8302	Val Acc:	0.6315	Test Acc:	0.6350
12	Model 1	Epoch	11	Train Loss:	1.8009	Train Acc:	0.6643	Val Loss:	1.8505	Val Acc:	0.6087	Test Acc:	0.6087
13	.												
14	.												
15	.												
16	Model 1	Epoch	140	Train Loss:	1.5000	Train Acc:	0.9629	Val Loss:	1.7308	Val Acc:	0.7294	Test Acc:	0.7324
17	Model 1	Epoch	141	Train Loss:	1.4998	Train Acc:	0.9636	Val Loss:	1.7252	Val Acc:	0.7337	Test Acc:	0.7312
18	Model 1	Epoch	142	Train Loss:	1.4992	Train Acc:	0.9644	Val Loss:	1.7290	Val Acc:	0.7320	Test Acc:	0.7295
19	Model 1	Epoch	143	Train Loss:	1.5006	Train Acc:	0.9628	Val Loss:	1.7275	Val Acc:	0.7326	Test Acc:	0.7351
20	Model 1	Epoch	144	Train Loss:	1.4986	Train Acc:	0.9644	Val Loss:	1.7264	Val Acc:	0.7328	Test Acc:	0.7351
21	Model 1	Epoch	145	Train Loss:	1.4990	Train Acc:	0.9639	Val Loss:	1.7278	Val Acc:	0.7317	Test Acc:	0.7343

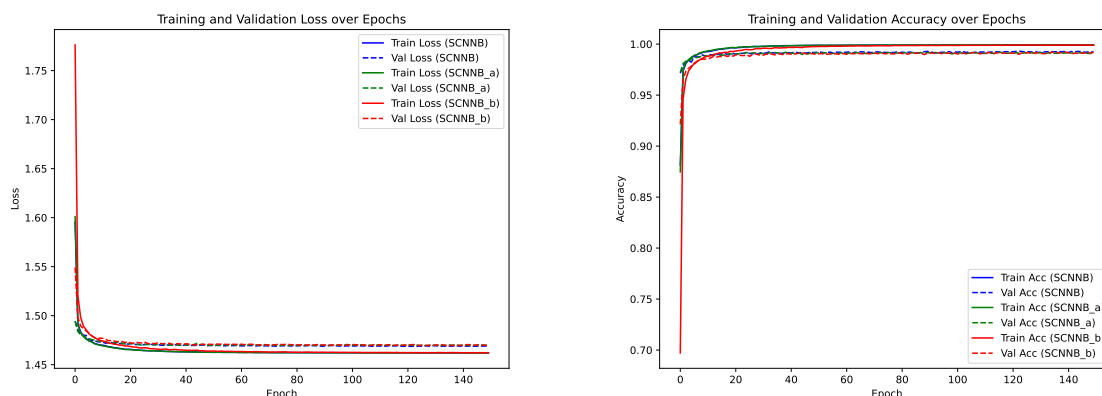
22	Model 1	Epoch 146	Train Loss: 1.4984	Train Acc: 0.9642	Val Loss: 1.7316	Val Acc: 0.7276	Test Acc: 0.7317
23	Model 1	Epoch 147	Train Loss: 1.4987	Train Acc: 0.9645	Val Loss: 1.7281	Val Acc: 0.7339	Test Acc: 0.7327
24	Model 1	Epoch 148	Train Loss: 1.4986	Train Acc: 0.9644	Val Loss: 1.7281	Val Acc: 0.7308	Test Acc: 0.7334
25	Model 1	Epoch 149	Train Loss: 1.4981	Train Acc: 0.9649	Val Loss: 1.7290	Val Acc: 0.7299	Test Acc: 0.7285
26	Model 1	Epoch 150	Train Loss: 1.4978	Train Acc: 0.9651	Val Loss: 1.7265	Val Acc: 0.7340	Test Acc: 0.7291
27	Training model 2						
28	Model 2	Epoch 1	Train Loss: 2.1249	Train Acc: 0.3361	Val Loss: 2.0071	Val Acc: 0.4607	Test Acc: 0.4471
29	Model 2	Epoch 2	Train Loss: 2.0070	Train Acc: 0.4541	Val Loss: 1.9518	Val Acc: 0.5101	Test Acc: 0.4992
30	Model 2	Epoch 3	Train Loss: 1.9533	Train Acc: 0.5071	Val Loss: 1.9165	Val Acc: 0.5452	Test Acc: 0.5322
31	Model 2	Epoch 4	Train Loss: 1.9229	Train Acc: 0.5379	Val Loss: 1.9062	Val Acc: 0.5547	Test Acc: 0.5506
32	Model 2	Epoch 5	Train Loss: 1.8991	Train Acc: 0.5616	Val Loss: 1.8921	Val Acc: 0.5631	Test Acc: 0.5649
33	Model 2	Epoch 6	Train Loss: 1.8772	Train Acc: 0.5842	Val Loss: 1.8739	Val Acc: 0.5852	Test Acc: 0.5813
34	Model 2	Epoch 7	Train Loss: 1.8553	Train Acc: 0.6064	Val Loss: 1.8764	Val Acc: 0.5834	Test Acc: 0.5748
35	Model 2	Epoch 8	Train Loss: 1.8377	Train Acc: 0.6251	Val Loss: 1.8443	Val Acc: 0.6156	Test Acc: 0.6158
36	Model 2	Epoch 9	Train Loss: 1.8219	Train Acc: 0.6420	Val Loss: 1.8537	Val Acc: 0.6059	Test Acc: 0.6105
37	Model 2	Epoch 10	Train Loss: 1.8074	Train Acc: 0.6566	Val Loss: 1.8207	Val Acc: 0.6401	Test Acc: 0.6419
38							
39							
40							
41	Model 2	Epoch 140	Train Loss: 1.4970	Train Acc: 0.9661	Val Loss: 1.7264	Val Acc: 0.7343	Test Acc: 0.7302
42	Model 2	Epoch 141	Train Loss: 1.4973	Train Acc: 0.9660	Val Loss: 1.7225	Val Acc: 0.7380	Test Acc: 0.7315
43	Model 2	Epoch 142	Train Loss: 1.4964	Train Acc: 0.9665	Val Loss: 1.7235	Val Acc: 0.7361	Test Acc: 0.7307
44	Model 2	Epoch 143	Train Loss: 1.4965	Train Acc: 0.9664	Val Loss: 1.7231	Val Acc: 0.7357	Test Acc: 0.7300
45	Model 2	Epoch 144	Train Loss: 1.4961	Train Acc: 0.9667	Val Loss: 1.7232	Val Acc: 0.7362	Test Acc: 0.7281
46	Model 2	Epoch 145	Train Loss: 1.4953	Train Acc: 0.9674	Val Loss: 1.7236	Val Acc: 0.7347	Test Acc: 0.7304
47	Model 2	Epoch 146	Train Loss: 1.4958	Train Acc: 0.9669	Val Loss: 1.7230	Val Acc: 0.7372	Test Acc: 0.7287
48	Model 2	Epoch 147	Train Loss: 1.4955	Train Acc: 0.9673	Val Loss: 1.7240	Val Acc: 0.7359	Test Acc: 0.7312
49	Model 2	Epoch 148	Train Loss: 1.4952	Train Acc: 0.9677	Val Loss: 1.7259	Val Acc: 0.7327	Test Acc: 0.7297
50	Model 2	Epoch 149	Train Loss: 1.4949	Train Acc: 0.9677	Val Loss: 1.7251	Val Acc: 0.7339	Test Acc: 0.7338
51	Model 2	Epoch 150	Train Loss: 1.4950	Train Acc: 0.9676	Val Loss: 1.7230	Val Acc: 0.7354	Test Acc: 0.7280
52	Training model 3						
53	Model 3	Epoch 1	Train Loss: 2.2394	Train Acc: 0.2002	Val Loss: 2.1669	Val Acc: 0.2856	Test Acc: 0.2848
54	Model 3	Epoch 2	Train Loss: 2.1610	Train Acc: 0.2928	Val Loss: 2.1289	Val Acc: 0.3259	Test Acc: 0.3310
55	Model 3	Epoch 3	Train Loss: 2.1096	Train Acc: 0.3467	Val Loss: 2.0663	Val Acc: 0.3933	Test Acc: 0.3935
56	Model 3	Epoch 4	Train Loss: 2.0622	Train Acc: 0.3967	Val Loss: 2.0275	Val Acc: 0.4324	Test Acc: 0.4205
57	Model 3	Epoch 5	Train Loss: 2.0308	Train Acc: 0.4278	Val Loss: 2.0085	Val Acc: 0.4500	Test Acc: 0.4428
58	Model 3	Epoch 6	Train Loss: 2.0052	Train Acc: 0.4537	Val Loss: 1.9882	Val Acc: 0.4707	Test Acc: 0.4638
59	Model 3	Epoch 7	Train Loss: 1.9826	Train Acc: 0.4775	Val Loss: 1.9657	Val Acc: 0.4950	Test Acc: 0.4824
60	Model 3	Epoch 8	Train Loss: 1.9579	Train Acc: 0.5020	Val Loss: 1.9450	Val Acc: 0.5169	Test Acc: 0.5062
61	Model 3	Epoch 9	Train Loss: 1.9381	Train Acc: 0.5240	Val Loss: 1.9294	Val Acc: 0.5333	Test Acc: 0.5228
62	Model 3	Epoch 10	Train Loss: 1.9193	Train Acc: 0.5426	Val Loss: 1.9090	Val Acc: 0.5499	Test Acc: 0.5504
63							
64							
65							
66	Model 3	Epoch 140	Train Loss: 1.5083	Train Acc: 0.9543	Val Loss: 1.7658	Val Acc: 0.6909	Test Acc: 0.6923
67	Model 3	Epoch 141	Train Loss: 1.5083	Train Acc: 0.9539	Val Loss: 1.7662	Val Acc: 0.6890	Test Acc: 0.6954
68	Model 3	Epoch 142	Train Loss: 1.5083	Train Acc: 0.9540	Val Loss: 1.7642	Val Acc: 0.6954	Test Acc: 0.6918
69	Model 3	Epoch 143	Train Loss: 1.5077	Train Acc: 0.9542	Val Loss: 1.7657	Val Acc: 0.6923	Test Acc: 0.6928
70	Model 3	Epoch 144	Train Loss: 1.5078	Train Acc: 0.9546	Val Loss: 1.7650	Val Acc: 0.6929	Test Acc: 0.6913
71	Model 3	Epoch 145	Train Loss: 1.5072	Train Acc: 0.9554	Val Loss: 1.7687	Val Acc: 0.6894	Test Acc: 0.6882
72	Model 3	Epoch 146	Train Loss: 1.5066	Train Acc: 0.9558	Val Loss: 1.7641	Val Acc: 0.6941	Test Acc: 0.6901
73	Model 3	Epoch 147	Train Loss: 1.5060	Train Acc: 0.9565	Val Loss: 1.7625	Val Acc: 0.6945	Test Acc: 0.6923
74	Model 3	Epoch 148	Train Loss: 1.5052	Train Acc: 0.9569	Val Loss: 1.7632	Val Acc: 0.6945	Test Acc: 0.6953
75	Model 3	Epoch 149	Train Loss: 1.5058	Train Acc: 0.9564	Val Loss: 1.7667	Val Acc: 0.6914	Test Acc: 0.6891
76	Model 3	Epoch 150	Train Loss: 1.5057	Train Acc: 0.9567	Val Loss: 1.7661	Val Acc: 0.6914	Test Acc: 0.6947

- مجموعه داده CIFAR10 - آزمایش دوم: نتایج نمودار دقت و اتلاف مربوط به این مجموعه داده در شکل ۱۲ نشان داده شده است و بخشی از نتایج نمایش داده شده برای هر اپیک به صورت زیر است.

1	Training model 1						
2	Model 1	Epoch 1	Train Loss: 2.1036	Train Acc: 0.3597	Val Loss: 2.0085	Val Acc: 0.4530	Test Acc: 0.4489
3	Model 1	Epoch 2	Train Loss: 1.9745	Train Acc: 0.4867	Val Loss: 1.9263	Val Acc: 0.5390	Test Acc: 0.5289
4	Model 1	Epoch 3	Train Loss: 1.9275	Train Acc: 0.5337	Val Loss: 1.9177	Val Acc: 0.5424	Test Acc: 0.5369
5	Model 1	Epoch 4	Train Loss: 1.8979	Train Acc: 0.5625	Val Loss: 1.8760	Val Acc: 0.5852	Test Acc: 0.5736
6	Model 1	Epoch 5	Train Loss: 1.8776	Train Acc: 0.5841	Val Loss: 1.8839	Val Acc: 0.5756	Test Acc: 0.5727
7	Model 1	Epoch 6	Train Loss: 1.8620	Train Acc: 0.6008	Val Loss: 1.8775	Val Acc: 0.5816	Test Acc: 0.5807
8	Model 1	Epoch 7	Train Loss: 1.8468	Train Acc: 0.6157	Val Loss: 1.8500	Val Acc: 0.6086	Test Acc: 0.6088
9	Model 1	Epoch 8	Train Loss: 1.8294	Train Acc: 0.6330	Val Loss: 1.8548	Val Acc: 0.6020	Test Acc: 0.6033
10	Model 1	Epoch 9	Train Loss: 1.8146	Train Acc: 0.6491	Val Loss: 1.8457	Val Acc: 0.6133	Test Acc: 0.6157



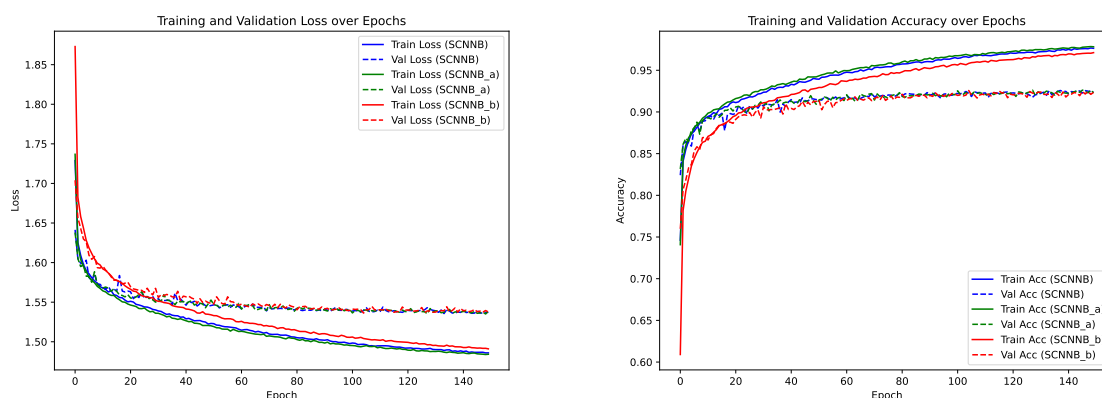
11	Model 1	Epoch 10	Train Loss: 1.8003	Train Acc: 0.6646	Val Loss: 1.8443	Val Acc: 0.6140	Test Acc: 0.6224
12	.						
13	.						
14	.						
15	Model 1	Epoch 140	Train Loss: 1.5006	Train Acc: 0.9628	Val Loss: 1.7280	Val Acc: 0.7308	Test Acc: 0.7287
16	Model 1	Epoch 141	Train Loss: 1.5003	Train Acc: 0.9627	Val Loss: 1.7263	Val Acc: 0.7333	Test Acc: 0.7328
17	Model 1	Epoch 142	Train Loss: 1.5001	Train Acc: 0.9626	Val Loss: 1.7238	Val Acc: 0.7341	Test Acc: 0.7312
18	Model 1	Epoch 143	Train Loss: 1.4995	Train Acc: 0.9635	Val Loss: 1.7238	Val Acc: 0.7333	Test Acc: 0.7343
19	Model 1	Epoch 144	Train Loss: 1.4994	Train Acc: 0.9635	Val Loss: 1.7241	Val Acc: 0.7346	Test Acc: 0.7312
20	Model 1	Epoch 145	Train Loss: 1.4990	Train Acc: 0.9639	Val Loss: 1.7281	Val Acc: 0.7311	Test Acc: 0.7317
21	Model 1	Epoch 146	Train Loss: 1.4992	Train Acc: 0.9639	Val Loss: 1.7226	Val Acc: 0.7363	Test Acc: 0.7378
22	Model 1	Epoch 147	Train Loss: 1.4987	Train Acc: 0.9642	Val Loss: 1.7233	Val Acc: 0.7357	Test Acc: 0.7348
23	Model 1	Epoch 148	Train Loss: 1.4989	Train Acc: 0.9640	Val Loss: 1.7265	Val Acc: 0.7326	Test Acc: 0.7339
24	Model 1	Epoch 149	Train Loss: 1.4983	Train Acc: 0.9647	Val Loss: 1.7241	Val Acc: 0.7361	Test Acc: 0.7354
25	Model 1	Epoch 150	Train Loss: 1.4984	Train Acc: 0.9644	Val Loss: 1.7261	Val Acc: 0.7337	Test Acc: 0.7319
26	.						
27	Training model 2						
28	Model 2	Epoch 1	Train Loss: 2.1258	Train Acc: 0.3340	Val Loss: 2.0149	Val Acc: 0.4465	Test Acc: 0.4351
29	Model 2	Epoch 2	Train Loss: 1.9828	Train Acc: 0.4775	Val Loss: 1.9835	Val Acc: 0.4722	Test Acc: 0.4679
30	Model 2	Epoch 3	Train Loss: 1.9300	Train Acc: 0.5304	Val Loss: 1.9279	Val Acc: 0.5288	Test Acc: 0.5281
31	Model 2	Epoch 4	Train Loss: 1.9006	Train Acc: 0.5611	Val Loss: 1.8913	Val Acc: 0.5664	Test Acc: 0.5650
32	Model 2	Epoch 5	Train Loss: 1.8783	Train Acc: 0.5838	Val Loss: 1.8796	Val Acc: 0.5837	Test Acc: 0.5716
33	Model 2	Epoch 6	Train Loss: 1.8601	Train Acc: 0.6016	Val Loss: 1.9129	Val Acc: 0.5466	Test Acc: 0.5402
34	Model 2	Epoch 7	Train Loss: 1.8450	Train Acc: 0.6170	Val Loss: 1.8631	Val Acc: 0.5993	Test Acc: 0.5935
35	Model 2	Epoch 8	Train Loss: 1.8265	Train Acc: 0.6378	Val Loss: 1.8347	Val Acc: 0.6235	Test Acc: 0.6198
36	Model 2	Epoch 9	Train Loss: 1.8139	Train Acc: 0.6498	Val Loss: 1.8229	Val Acc: 0.6371	Test Acc: 0.6386
37	Model 2	Epoch 10	Train Loss: 1.7996	Train Acc: 0.6641	Val Loss: 1.8170	Val Acc: 0.6436	Test Acc: 0.6384
38	.						
39	.						
40	.						
41	Model 2	Epoch 140	Train Loss: 1.4980	Train Acc: 0.9648	Val Loss: 1.7233	Val Acc: 0.7379	Test Acc: 0.7346
42	Model 2	Epoch 141	Train Loss: 1.4983	Train Acc: 0.9648	Val Loss: 1.7218	Val Acc: 0.7399	Test Acc: 0.7337
43	Model 2	Epoch 142	Train Loss: 1.4976	Train Acc: 0.9653	Val Loss: 1.7240	Val Acc: 0.7371	Test Acc: 0.7305
44	Model 2	Epoch 143	Train Loss: 1.4969	Train Acc: 0.9658	Val Loss: 1.7293	Val Acc: 0.7336	Test Acc: 0.7293
45	Model 2	Epoch 144	Train Loss: 1.4970	Train Acc: 0.9659	Val Loss: 1.7221	Val Acc: 0.7375	Test Acc: 0.7357
46	Model 2	Epoch 145	Train Loss: 1.4967	Train Acc: 0.9661	Val Loss: 1.7255	Val Acc: 0.7345	Test Acc: 0.7366
47	Model 2	Epoch 146	Train Loss: 1.4965	Train Acc: 0.9666	Val Loss: 1.7254	Val Acc: 0.7330	Test Acc: 0.7312
48	Model 2	Epoch 147	Train Loss: 1.4964	Train Acc: 0.9664	Val Loss: 1.7242	Val Acc: 0.7367	Test Acc: 0.7342
49	Model 2	Epoch 148	Train Loss: 1.4962	Train Acc: 0.9666	Val Loss: 1.7242	Val Acc: 0.7348	Test Acc: 0.7382
50	Model 2	Epoch 149	Train Loss: 1.4964	Train Acc: 0.9667	Val Loss: 1.7210	Val Acc: 0.7385	Test Acc: 0.7325
51	Model 2	Epoch 150	Train Loss: 1.4961	Train Acc: 0.9665	Val Loss: 1.7217	Val Acc: 0.7371	Test Acc: 0.7333
52	.						
53	Training model 3						
54	Model 3	Epoch 1	Train Loss: 2.2281	Train Acc: 0.2118	Val Loss: 2.1638	Val Acc: 0.2891	Test Acc: 0.2915
55	Model 3	Epoch 2	Train Loss: 2.1505	Train Acc: 0.3035	Val Loss: 2.1092	Val Acc: 0.3486	Test Acc: 0.3484
56	Model 3	Epoch 3	Train Loss: 2.0983	Train Acc: 0.3601	Val Loss: 2.0753	Val Acc: 0.3789	Test Acc: 0.3776
57	Model 3	Epoch 4	Train Loss: 2.0685	Train Acc: 0.3891	Val Loss: 2.0361	Val Acc: 0.4221	Test Acc: 0.4145
58	Model 3	Epoch 5	Train Loss: 2.0383	Train Acc: 0.4198	Val Loss: 2.0125	Val Acc: 0.4475	Test Acc: 0.4405
59	Model 3	Epoch 6	Train Loss: 2.0121	Train Acc: 0.4479	Val Loss: 1.9958	Val Acc: 0.4632	Test Acc: 0.4562
60	Model 3	Epoch 7	Train Loss: 1.9891	Train Acc: 0.4712	Val Loss: 1.9720	Val Acc: 0.4896	Test Acc: 0.4735
61	Model 3	Epoch 8	Train Loss: 1.9637	Train Acc: 0.4967	Val Loss: 1.9514	Val Acc: 0.5077	Test Acc: 0.5030
62	Model 3	Epoch 9	Train Loss: 1.9465	Train Acc: 0.5145	Val Loss: 1.9356	Val Acc: 0.5262	Test Acc: 0.5226
63	Model 3	Epoch 10	Train Loss: 1.9283	Train Acc: 0.5324	Val Loss: 1.9197	Val Acc: 0.5423	Test Acc: 0.5386
64	.						
65	.						
66	.						
67	Model 3	Epoch 140	Train Loss: 1.5081	Train Acc: 0.9543	Val Loss: 1.7603	Val Acc: 0.6984	Test Acc: 0.6899
68	Model 3	Epoch 141	Train Loss: 1.5082	Train Acc: 0.9540	Val Loss: 1.7618	Val Acc: 0.6960	Test Acc: 0.6873
69	Model 3	Epoch 142	Train Loss: 1.5077	Train Acc: 0.9546	Val Loss: 1.7578	Val Acc: 0.7035	Test Acc: 0.6908
70	Model 3	Epoch 143	Train Loss: 1.5064	Train Acc: 0.9559	Val Loss: 1.7595	Val Acc: 0.6996	Test Acc: 0.6899
71	Model 3	Epoch 144	Train Loss: 1.5072	Train Acc: 0.9548	Val Loss: 1.7592	Val Acc: 0.7001	Test Acc: 0.6937
72	Model 3	Epoch 145	Train Loss: 1.5067	Train Acc: 0.9557	Val Loss: 1.7630	Val Acc: 0.6951	Test Acc: 0.6910
73	Model 3	Epoch 146	Train Loss: 1.5065	Train Acc: 0.9560	Val Loss: 1.7605	Val Acc: 0.6985	Test Acc: 0.6901
74	Model 3	Epoch 147	Train Loss: 1.5064	Train Acc: 0.9556	Val Loss: 1.7619	Val Acc: 0.6962	Test Acc: 0.6920
75	Model 3	Epoch 148	Train Loss: 1.5062	Train Acc: 0.9559	Val Loss: 1.7618	Val Acc: 0.6972	Test Acc: 0.6934
76	Model 3	Epoch 149	Train Loss: 1.5053	Train Acc: 0.9570	Val Loss: 1.7581	Val Acc: 0.7013	Test Acc: 0.6910
77	Model 3	Epoch 150	Train Loss: 1.5054	Train Acc: 0.9571	Val Loss: 1.7603	Val Acc: 0.6992	Test Acc: 0.6892



(ب) نمودار تابع اتلاف

(آ) نمودار دقت

شکل ۹: نمودارهای دقت و اتلاف برای داده‌های آموزش و اعتبارسنجی (پیاده‌سازی روی مجموعه‌داده MNIST).



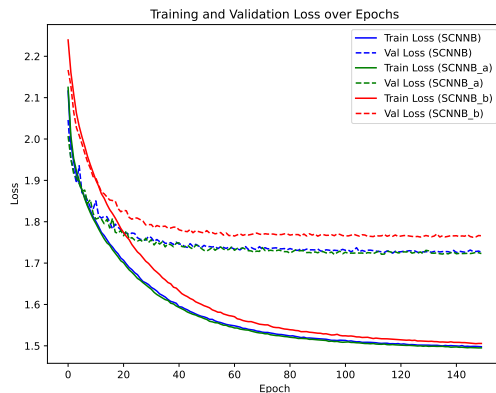
(ب) نمودار تابع اتلاف

(آ) نمودار دقت

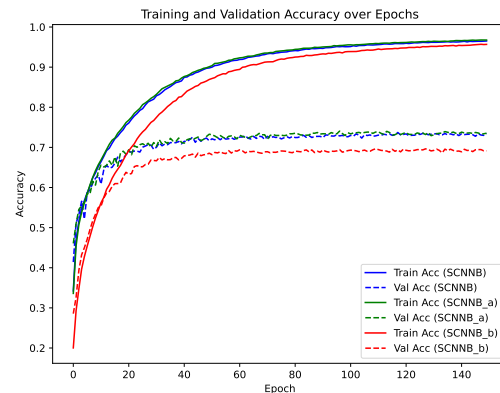
شکل ۱۰: نمودارهای دقت و اتلاف برای داده‌های آموزش و اعتبارسنجی (پیاده‌سازی روی مجموعه‌داده Fashion-MNIST).

### ۳.۴.۱ پاسخ قسمت ج

هم‌چنانی که از نتایج و نمودارهای آورده‌شده در قسمت نتایج پیاده‌سازی (بخش ۴.۱) آورده شده است، نتایج مربوط به دقت تست و سایر پارامترهای ارزیابی به مقادیر مطلوبی و کاملاً نزدیک به منطق مقاله رسیده‌اند. این موضوع علاوه بر نمودارها در جدول ۱ هم کاملاً مشخص است. هم‌چنین نتایج مربوط به ماتریس درهم‌ریختگی برای ۱۰ کلاس هر سه دیتاست نشان‌دهنده تمرکز بخش غالب پیش‌بینی‌ها روی قطر اصلی و در نتیجه عمل کرد مناسب است. هم‌چنین نتایج مربوط به نمودار دقت و تابع اتلاف داده‌های ارزیابی و اعتبارسنجی نشان می‌دهد، مدل به شکل مناسبی آموزش دیده و دچار پدیده‌هایی مانند بیش‌برازش نشده. لازم به ذکر است که مهم نیست که تابع اتلاف همیشه بزرگ‌تر از یک است. بزرگی تابع اتلاف به مقیاس داده‌ها بستگی دارد. تابع CrossEntropyLoss یک اندازه‌گیری از این است که چقدر احتمالات پیش‌بینی شده با احتمالات واقعی هم‌خوانی دارند. در وظایف طبقه‌بندی، استفاده از تابع فعال‌سازی softmax در لایه خروجی شبکه عصبی معمول است، که مقادیر خروجی

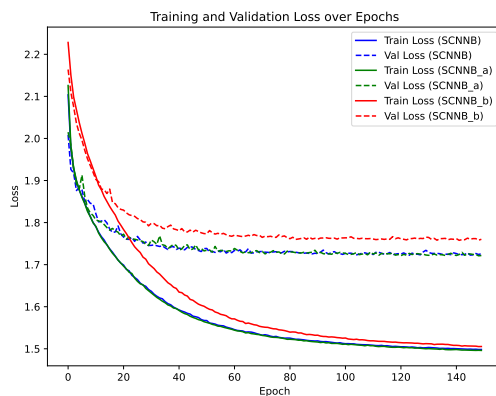


(ب) نمودار تابع اتلاف

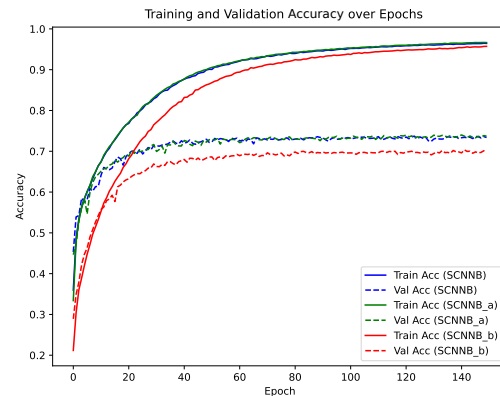


(آ) نمودار دقت

شکل ۱۱: نمودارهای دقت و اتلاف برای داده‌های آموزش و اعتبارسنجی (مجموعه داده CIFAR10 - آزمایش اول).



(ب) نمودار تابع اتلاف



(آ) نمودار دقت

شکل ۱۲: نمودارهای دقت و اتلاف برای داده‌های آموزش و اعتبارسنجی (مجموعه داده CIFAR10 - آزمایش دوم).

را در محدوده  $(0, 1)$  نگه می‌دارد و آن‌ها را به طوری نرمال می‌کند که مجموع آن‌ها به ۱ برسد. این بدان معنی است که احتمالات پیش‌بینی شده مدل بین ۰ و ۱ محدود هستند. با این حال، تابع  $CrossEntropyLoss$  به طور ضروری به این محدودیت وابسته نیست و ممکن است بیشتر از ۱ باشد. بنابراین روند کاهشی تابع اتلاف برای داده‌های آموزش و اعتبارسنجی اهمیت دارد. علاوه بر این‌ها در حالتی که لایه‌های نرمال‌سازی دسته کاملاً حذف شده‌اند، نتایج افت محسوسی پیدا کرده که این موضوع با توجه به مزایایی که از این لایه در قسمت‌های پیشین ذکر شد قابل پیش‌بینی بوده است. با استفاده از نتایج مقاله و پیاده‌سازی می‌توان نشان داد که استفاده از تکنیک نرمال‌سازی دسته موجب شتاب‌بخشیدن به آموزش و کاهش استفاده از حافظه (که در طول اجرای کدها حس شد) و بالاتر رفتن دقت می‌شود.