



به نام خدا
دانشگاه تهران
دانشکده مهندسی برق و کامپیوتر



درس شبکه‌های عصبی و یادگیری عمیق

تمرین دوم

نام و نام خانوادگی	فرید سیاهکلی – سعید شکوفا
شماره دانشجویی	810198418 – 810198510
تاریخ ارسال گزارش	1401.09.02

فهرست

پاسخ 1. تاثیر تغییر رزولوشن در طبقه بندی در شبکه CNN.....1

1-1. تهیه سه استایل با رزولوشن مختلف

2

1-2. نحوه تقسیم داده های آموزش تست و ارزیابی

3

1-3. پیاده سازی روش TOTV

4

1-4. پیاده سازی روش TVTV

4

پاسخ ۲ - آشنایی با معماری شبکه CNN.....6

1-2. لود دیتاست مقاله

6

2-2. انتخاب معماری

6

2-3. توضیح لایه های مختلف معماری

8

2-4. مقایسه نتایج دو معماری مختلف

9

2-5. مقایسه نتایج استفاده از بهینه ساز مختلف

10

2-6. استفاده از دراپ اوت

11

شکل‌ها

- 1 شکل 1 دست گرمی.....
- 2 شکل 2 رزولوشن مختلف از دیتاست CIFAR10.....
- 3 شکل 3 تقسیم بندی داده های آموزشی و تست.....
- 4 شکل 4 نتایج شبکه بر روی دیتای تست پس از آموزش به شیوه TOTV.....
- 4 شکل 5 نتایج شبکه بر روی دیتای تست پس از آموزش به شیوه TVTV.....
- 6 شکل 6 لود دیتاست.....
- 7 شکل 7 معماری های انتخاب شده.....
- 8 شکل 8 پیاده سازی معماری ها.....
- 9 شکل 9 هایپرپارامترها برای دو معماری انتخابی.....
- 9 شکل 10 نتیجه آموزش برای بهینه ساز adam معماری پنجم.....
- 9 شکل 11 نتیجه آموزش برای بهینه ساز adam معماری چهارم.....
- 10 شکل 12 نتایج برای بهینه ساز adam و معماری پنجم.....
- 10 شکل 13 نتایج برای بهینه ساز adam و معماری چهارم.....
- 10 شکل 14 نتایج برای بهینه ساز SGD و معماری پنجم.....
- 10 شکل 15 نتایج برای بهینه ساز SGD و معماری چهارم.....

جدول‌ها

- جدول 1 نتایج برای دو روش TOTV و TVTV 5
- جدول 2 نتایج آموزش 9
- جدول 3 نتایج نهایی 11

پاسخ 1. تاثیر تغییر رزولوشن در طبقه بندی در شبکه CNN

ردیف اول \rightarrow

$$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix} \quad \begin{bmatrix} 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

ردیف دوم \rightarrow

$$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \quad \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

فیلترها \rightarrow

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix} \quad \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

(1) bias = -2 (2) bias = -2

فیلتر (1) بر ردیف اول به همراه ReLU \rightarrow

$$\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \quad \begin{bmatrix} 2 & 1 \\ 1 & 1 \end{bmatrix} \xrightarrow{\text{max pooling}} [1], [2]$$

فیلتر (2) بر ردیف اول به همراه ReLU \rightarrow

$$\begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix} \quad \begin{bmatrix} 4 & 2 \\ 2 & 1 \end{bmatrix} \xrightarrow{\text{max pooling}} [5] \quad [6]$$

فیلتر (1) بر ردیف دوم به همراه ReLU \rightarrow

$$\begin{bmatrix} 0 & 2 \\ 0 & 0 \end{bmatrix} \quad \begin{bmatrix} 0 & 0 \\ 3 & 1 \end{bmatrix} \xrightarrow{\text{max pooling}} [2] \quad [3]$$

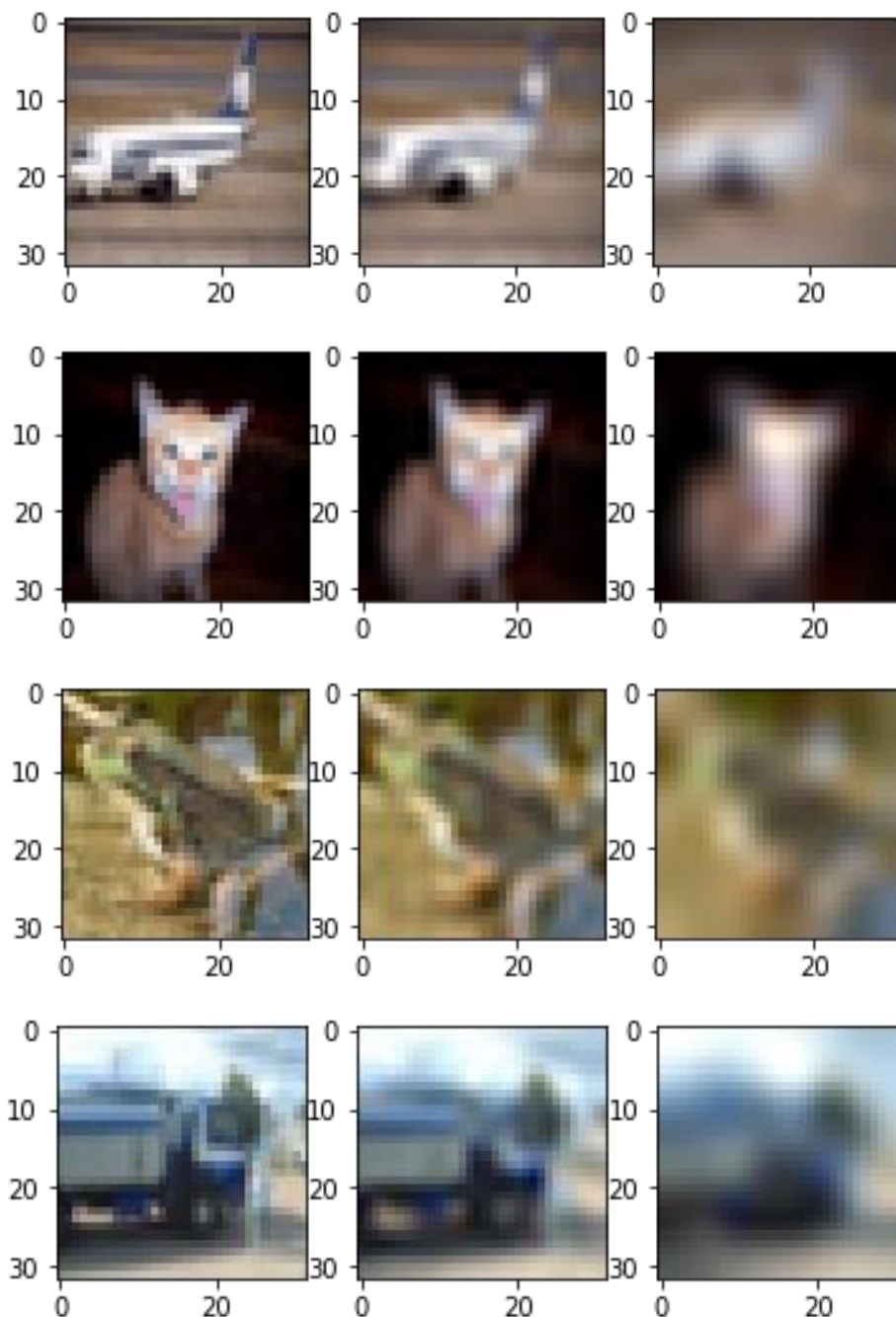
فیلتر (2) بر ردیف دوم به همراه ReLU \rightarrow

$$\begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix} \quad \begin{bmatrix} 1 & 3 \\ 2 & 2 \end{bmatrix} \xrightarrow{\text{max pooling}} [1] \quad [3]$$

شکل 1 دست گرمی

1-1. تهیه سه استایل دیتاست با رزولوشن های مختلف

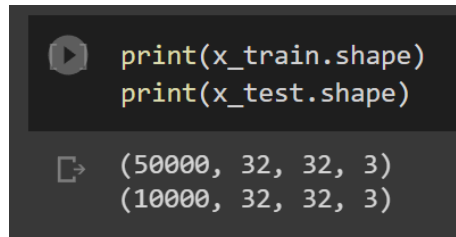
برای انجام اینکار ابتدا تصاویر را به مقدار مدنظر ریسایز کرده و سپس آنها به 32×32 بازگردانده شده تا برای آموزش شبکه دچار مشکل نشویم.



شکل 2 رزولوشن مختلف از دیتاست CIFAR10

1-2. نحوه تقسیم داده های آموزش تست و ارزیابی

پس از دانلود دیتاست آماده در پلتفرم پایتورچ، داده ها به دو دسته آموزش و ارزیابی جدا شده اند. در حین آموزش شبکه، تمامی داده های آموزشی به عنوان داده های validation نیز در نظر گرفته شده است.



```
print(x_train.shape)
print(x_test.shape)

(50000, 32, 32, 3)
(10000, 32, 32, 3)
```

شکل 3 تقسیم بندی داده های آموزشی و تست

برای سهولت کار برای رزولوشن های متفاوت دیتالودر جداگانه تعریف شده تا ارزیابی شبکه آسانتر شود.

```
[ ] class DatasetDelivery():
    def __init__(self, x, y):
        self.x = x
        self.y = y
        #self.transform = transform

    def __len__(self):
        return (self.x.shape[0])

    def __getitem__(self, idx):
        return self.x[idx], self.y[idx]

trainset = DatasetDelivery(x_train, targets)
testset = DatasetDelivery(x_test, test_tar)

trainloader = torch.utils.data.DataLoader(trainset, batch_size=batch_size, shuffle=True)
testloader = torch.utils.data.DataLoader(testset, batch_size=batch_size, shuffle=False)

trainset16 = DatasetDelivery(x_train16, targets)
testset16 = DatasetDelivery(x_test16, test_tar)

trainloader16 = torch.utils.data.DataLoader(trainset16, batch_size=batch_size, shuffle=True)
testloader16 = torch.utils.data.DataLoader(testset16, batch_size=batch_size, shuffle=False)

trainset8 = DatasetDelivery(x_train8, targets)
testset8 = DatasetDelivery(x_test8, test_tar)

trainloader8 = torch.utils.data.DataLoader(trainset8, batch_size=batch_size, shuffle=True)
testloader8 = torch.utils.data.DataLoader(testset8, batch_size=batch_size, shuffle=False)
```

شکل 4 دیتالودر برای دیتاست های 8x8 16x16 و 32x32

1-3. پیاده سازی روش TOTV

ابتدا شبکه را بر روی دیتاست 32x32 آموزش داده و سپس بر روی دیتاست 16x16 و 8x8 ارزیابی به عمل آمده است:

<pre> with torch.no_grad(): model.eval() for x_tes, testtar in testloader: x_tes = x_tes.to(device) testtar = testtar.to(device) outt = model(x_tes) outt = softmax(outt) fl_list.append(metric(outt.cpu(), testtar.cpu())) acc_list.append(metric1(outt.cpu(), testtar.cpu())) pre_list.append(metric2(outt.cpu(), testtar.cpu())) loss_t = cri(outt, testtar) los_test.append(loss_t.item()) print(f'test loss : {sum(los_test)/len(los_test)}') print(f'F1 : {sum(fl_list)/len(fl_list)}') print(f'Acc : {sum(acc_list)/len(acc_list)}') print(f'Pre : {sum(pre_list)/len(pre_list)}')</pre>	<pre> [23] with torch.no_grad(): model.eval() for x_tes, testtar in testloader16: x_tes = x_tes.to(device) testtar = testtar.to(device) outt = model(x_tes) outt = softmax(outt) fl_list.append(metric(outt.cpu(), testtar.cpu())) acc_list.append(metric1(outt.cpu(), testtar.cpu())) pre_list.append(metric2(outt.cpu(), testtar.cpu())) loss_t = cri(outt, testtar) los_test.append(loss_t.item()) print(f'test loss : {sum(los_test)/len(los_test)}') print(f'F1 : {sum(fl_list)/len(fl_list)}') print(f'Acc : {sum(acc_list)/len(acc_list)}') print(f'Pre : {sum(pre_list)/len(pre_list)}')</pre>	<pre> with torch.no_grad(): model.eval() for x_tes, testtar in testloader8: x_tes = x_tes.to(device) testtar = testtar.to(device) outt = model(x_tes) outt = softmax(outt) fl_list.append(metric(outt.cpu(), testtar.cpu())) acc_list.append(metric1(outt.cpu(), testtar.cpu())) pre_list.append(metric2(outt.cpu(), testtar.cpu())) loss_t = cri(outt, testtar) los_test.append(loss_t.item()) print(f'test loss : {sum(los_test)/len(los_test)}') print(f'F1 : {sum(fl_list)/len(fl_list)}') print(f'Acc : {sum(acc_list)/len(acc_list)}') print(f'Pre : {sum(pre_list)/len(pre_list)}')</pre>
<pre> test loss : 1.7139142668187617 F1 : 0.7848180532455444 Acc : 0.9592000246047974 Pre : 0.8303082585334778</pre>	<pre> test loss : 1.9286193169414996 F1 : 0.5443117022514343 Acc : 0.9214524830685425 Pre : 0.5958520174026489</pre>	<pre> test loss : 1.917346421869596 F1 : 0.5500407419624320 Acc : 0.922716557979837 Pre : 0.5881116390228271</pre>

شکل 5 نتایج شبکه بر روی دیتای تست پس از آموزش به شیوه TOTV

1-4. پیاده سازی روش TVTV

ابتدا شبکه را بر روی دیتاست های 32x32، 16x16 و 8x8 آموزش داده و سپس بر روی دیتاست متناظر با آن که برای آموزش استفاده شده، ارزیابی به عمل آمده:

<pre> with torch.no_grad(): model.eval() for x_tes, testtar in testloader: x_tes = x_tes.to(device) testtar = testtar.to(device) outt = model(x_tes) outt = softmax(outt) fl_list.append(metric(outt.cpu(), testtar.cpu())) acc_list.append(metric1(outt.cpu(), testtar.cpu())) pre_list.append(metric2(outt.cpu(), testtar.cpu())) loss_t = cri(outt, testtar) los_test.append(loss_t.item()) print(f'test loss : {sum(los_test)/len(los_test)}') print(f'F1 : {sum(fl_list)/len(fl_list)}') print(f'Acc : {sum(acc_list)/len(acc_list)}') print(f'Pre : {sum(pre_list)/len(pre_list)}')</pre>	<pre> [19] with torch.no_grad(): model.eval() for x_tes, testtar in testloader16: x_tes = x_tes.to(device) testtar = testtar.to(device) outt = model(x_tes) outt = softmax(outt) fl_list.append(metric(outt.cpu(), testtar.cpu())) acc_list.append(metric1(outt.cpu(), testtar.cpu())) pre_list.append(metric2(outt.cpu(), testtar.cpu())) loss_t = cri(outt, testtar) los_test.append(loss_t.item()) print(f'test loss : {sum(los_test)/len(los_test)}') print(f'F1 : {sum(fl_list)/len(fl_list)}') print(f'Acc : {sum(acc_list)/len(acc_list)}') print(f'Pre : {sum(pre_list)/len(pre_list)}')</pre>	<pre> [19] with torch.no_grad(): model.eval() for x_tes, testtar in testloader8: x_tes = x_tes.to(device) testtar = testtar.to(device) outt = model(x_tes) outt = softmax(outt) fl_list.append(metric(outt.cpu(), testtar.cpu())) acc_list.append(metric1(outt.cpu(), testtar.cpu())) pre_list.append(metric2(outt.cpu(), testtar.cpu())) loss_t = cri(outt, testtar) los_test.append(loss_t.item()) print(f'test loss : {sum(los_test)/len(los_test)}') print(f'F1 : {sum(fl_list)/len(fl_list)}') print(f'Acc : {sum(acc_list)/len(acc_list)}') print(f'Pre : {sum(pre_list)/len(pre_list)}')</pre>
<pre> test loss : 1.7139142668187617 F1 : 0.7848180532455444 Acc : 0.9592000246047974 Pre : 0.8303082585334778</pre>	<pre> test loss : 1.7754117911279206 F1 : 0.7254805564880371 Acc : 0.9489599466323853 Pre : 0.7848716974258423</pre>	<pre> test loss : 2.3214625208854676 F1 : 0.04625517129898071 Acc : 0.8681300282478333 Pre : 0.1063908189535141</pre>

شکل 6 نتایج شبکه بر روی دیتای تست پس از آموزش به شیوه TVTV

نتایج بدست آمده در جدول قرار گرفته و به شرح زیر است:

جدول 1 نتایج برای دو روش TOTV و TVTV

Dataset Resolution	TOTV			TVTV		
Metrics	F1	Accuracy	Precision	F1	Accuracy	Precision
32x32	78.48	95.92	83.03	78.48	95.92	83.03
16x16	54.42	92.14	59.58	72.54	94.89	78.48
8x8	55.00	92.27	58.81	4.62	86.81	10.64

نتیجه می شود که با افزایش کیفیت تصاویر، کیفیت یادگیری شبکه نیز افزایش می یابد چرا که با ثابت نگه داشتن مدل و کاهش کیفیت تصاویر دیده شد که شبکه در تفکیک کلاس ها دچار ضعف می شود.

پاسخ ۲ - آشنایی با معماری شبکه CNN

2-1. لود دیتاست مقاله

دیتاست توسط پلتفرم آماده‌های که در پایتورچ موجود است دانلود شده که 60000 دیتا برای آموزش و 10000 دیتای آن برای ارزیابی است.

```
[ ] # load dataset
(trainX, trainy), (testX, testy) = fashion_mnist.load_data()

Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-labels-idx1-ubyte.gz
29515/29515 [=====] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-images-idx3-ubyte.gz
26421880/26421880 [=====] - 2s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-labels-idx1-ubyte.gz
5148/5148 [=====] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-images-idx3-ubyte.gz
4422102/4422102 [=====] - 0s 0us/step

[ ] print(trainX.shape)
    print(trainy.shape)
    print(testy.shape)

(60000, 28, 28)
(60000,)
(10000,)
```

شکل 7 لود دیتاست

2-2. انتخاب معماری

در این گزارش دو معماری چهارم و پنجم پیاده سازی شده است. تفاوت میان این دو معماری در سائز کرنل لایه کانولوشن و همچنین وجود و یا عدم وجود تابع فعالسازی و لایه پولینگ در میان کانولوشن ها است. تعداد لایه های MLP نیز در این دو معماری شبیه یکدیگر می باشند.

Architecture 4	Architecture 5
4 convolutional layers with (2 x 2) filter size and 2 fully connected layers	4 convolutional layers with (3 x 3) filter size and 2 fully connected layers
(1) INPUT:28×28×1 (2) FC:10 Output Classes	(1) INPUT:28×28×1 (2) FC:10 Output Classes
(3) CONV2D:2×2 size,64 filters (4) POOL:2×2 size (5) DROPOUT: = 0.25 (6) CONV2D:2×2 size,64 filters (7) POOL:2×2 size (8) DROPOUT: = 0.25 (9) CONV2D:2×2 size,64 filters (10) POOL:2×2 size (11) DROPOUT: = 0.25 (12) CONV2D :2×2 size,64 filters (13) DROPOUT: = 0.25 (14) FC:64 Hidden Neurons (15) DROPOUT: = 0.25	(3) CONV2D:3×3 size,32 filters (4) CONV2D:3×3 size,32 filters (4) POOL:2×2 size (5) DROPOUT: = 0.25 (6) CONV2D:3×3 size,64 filters (7) CONV2D:3×3 size,64 filters (8) POOL:2×2 size (9) DROPOUT: = 0.25 (10) FC:512 Hidden Neurons (11) DROPOUT: = 0.5

شکل 8 معماری های انتخاب شده

```

class CNNArch5(nn.Module):
    def __init__(self,K):
        super(CNNArch5,self).__init__()
        self.convs = nn.Sequential(
            nn.Conv2d(1,32,3),
            nn.ReLU(),
            nn.Conv2d(32,32,3),
            nn.ReLU(),
            nn.MaxPool2d(2),
            nn.Dropout(p=0.25),
            nn.Conv2d(32,64,3),
            nn.ReLU(),
            nn.Conv2d(64,64,3),
            nn.ReLU(),
            nn.MaxPool2d(2),
            nn.Dropout(p=0.25),
        )
        self.deep=nn.Sequential(
            nn.Linear(1024,512),
            nn.ReLU(),
            nn.Dropout(0.5),
            nn.Linear(512,K)
        )

    def forward(self,X):
        out=self.convs(X)
        #print(out.shape)
        out=out.view(out.size(0),-1)
        out=self.deep(out)
        return out

class CNNArch4(nn.Module):
    def __init__(self,K):
        super(CNNArch4,self).__init__()
        self.convs = nn.Sequential(
            nn.Conv2d(1,64,2),
            nn.ReLU(),
            nn.MaxPool2d(2),
            nn.Dropout(p=0.25),
            nn.Conv2d(64,64,2),
            nn.ReLU(),
            nn.MaxPool2d(2),
            nn.Dropout(p=0.25),
            nn.Conv2d(64,64,2),
            nn.Dropout(p=0.25)
        )
        self.deep=nn.Sequential(
            nn.Linear(1024,64),
            nn.ReLU(),
            nn.Dropout(0.25),
            nn.Linear(64, K)
        )

    def forward(self,X):
        out=self.convs(X)
        #print(out.shape)
        out=out.view(out.size(0),-1)
        out=self.deep(out)
        return out

```

شکل 9 پیاده سازی معماری ها

در حین آموزش این دو معماری Optimal Parameter ها به صورت زیر در نظر گرفته شده است.

E. Results of Architecture 4

In architecture 4, the optimal parameter are 128 batch size, 50 epochs, Softmax activation function, adam optimizer, 2x2 kernel size and 0.25 dropout after each pooling layers. For MNIST dataset, best obtained training accuracy and testing accuracy are 99.02% and 99.03% respectively. For Fashion-MNIST dataset,

F. Results of Architecture 5

In architecture 5, the optimal parameter are 128 batch size, 50 epochs, softmax activation function, RMSprop optimizer, (2x2) kernel size and 0.25 dropout after each pooling layers. For MNIST dataset, best obtained training accuracy and testing accuracy are 99.26% % and 99.29% % respectively. For Fashion-MNIST dataset, best obtained training accuracy and testing accuracy are 92.67 % and 92.86 % respectively.

شکل 10 هایپر پارامترها برای دو معماری انتخابی

2-3. توضیح لایه های مختلف معماری

معماری شماره 4 در ابتدا یک CNN دوبعدی با $\text{kernel size}=2*2$ بر روی تصویر می زند و از یک کانال ورودی 64 خروجی تولید می کند که همان feature map ها هستند سپس دوباره با استفاده از یک $\text{max pull}=2*2$ سائز خروجی ها را کوچک تر کرده و 25 درصد نورون ها را dropout میکنیم. حال این 64 کانال تولید شده را به یک CNN دیگر با 64 کانال ورودی و 64 کانال خروجی میدهیم و مجددا همان عمل $\text{max pull}=2*2$ و 25 درصد dropout را انجام می دهیم و یک بار دیگر نیز CNN 64 به 64 را به همراه 25 درصد dropout انجام میدهیم و در این مرحله خروجی CNN را به MLP می دهیم که کار classification را انجام دهد. قبل از دادن خروجی CNN ها به MLP ابتدا خروجی را flat میکنیم و سپس آن را به MLP با 1024 ورودی و 64 خروجی میدهیم و در آخر از آنجا که ده کلاس داریم تین 64 تا را به 10 عدد تبدیل کرده و تابع $\text{activation function} = \text{softmax}()$ را استفاده میکنیم. (همچنین پس از هر CNN و هر لایه در MLP از ReLU استفاده شده است)

معماری شبکه 5 ابتدا یک CNN دوبعدی با $\text{kernel size}=3*3$ بر روی تصویر میزند و از یک کانال ورودی 32 تا کانال خروجی تولید می کند. سپس یک $\text{max pool}=2*2$ بر روی خروجی میزند که ابعاد را کم کند و 25 درصد نورون ها را dropout می کند سپس با استفاده از یک CNN دیگر 32 کانال را به 64 کانال میبریم با همان $\text{kernel size}=3*3$ و مجددا از یک CNN 64 به 64 کانال و با $\text{kernel size}=3*3$ استفاده میکنیم و با استفاده از $\text{max pool}=2*2$ مجددا ابعاد را کاهش می دهیم و 25 درصد نورون ها را dropout میکنیم. و خروجی این CNN ها را به MLP می دهیم تا classification را انجام دهد. به طوری که یک لایه با 1024 ورودی (لایه flatten) و 512 خروجی و سپس 512 ورودی به 10 خروجی میزنیم و در آخر از $\text{softmax}()$ استفاده می کنیم.

تفاوت شبکه ها این است که در شبکه 4 kernel size کوچک تر بوده ولی تعداد کانال ها بیشتر است و در شبکه 5 تعداد کانال ها کمتر ولی kernel size بزرگتر (برابر 3×3) می باشد. البته این تفاوت تعداد کانال ها سبب شده که لایه ورودی MLP تعداد نورون های متفاوتی داشته باشد به علت اینکه خروجی CNN ها که flat می شود ابعاد متفاوتی دارند.

2-4. مقایسه نتایج دو معماری مختلف

دیتا ها در دیتا ست دانلود شده 80 به 20 تقسیم شده بودند.

نتیجه بهترین epoch در ادامه آمده است:

بهینه ساز Adam و معماری 5:

```
epoch 45 train loss : 0.18248738102249498
epoch 45 Model Saved
epoch 45 test loss : 1.5538448095321655
epoch 45 F1 : 0.9244696497917175
epoch 45 Acc : 0.985047459602356
epoch 45 Pre : 0.9332477450370789
```

شکل 11 نتیجه آموزش برای بهینه ساز adam معماری پنجم

بهینه ساز Adam و معماری 4:

```
* * *
epoch 48 train loss : 0.2691857848467349
epoch 48 test loss : 1.5754384737980516
epoch 48 F1 : 0.9157554507255554
epoch 48 Acc : 0.9833860993385315
epoch 48 Pre : 0.92770916223526
* * *
```

شکل 12 نتیجه آموزش برای بهینه ساز adam معماری چهارم

جدول 2 نتایج آموزش

-	F1	Accuracy	Precision
Architecture 4	91.57	98.33	92.77
Architecture 5	92.44	98.50	93.32

2-5. مقایسه نتایج استفاده از بهینه ساز مختلف

بهینه ساز Adam و معماری 5:

```
* * *  
epoch 49 train loss : 0.18506677922155304  
epoch 49 test loss : 1.5590080490595177  
epoch 49 F1 : 0.9253553748130798  
epoch 49 Acc : 0.9852750897407532  
epoch 49 Pre : 0.9378481507301331  
* * *
```

شکل 13 نتایج برای بهینه ساز adam و معماری پنجم

بهینه ساز Adam و معماری 4:

```
* * *  
epoch 48 train loss : 0.2691857848467349  
epoch 48 test loss : 1.5754384737980516  
epoch 48 F1 : 0.9157554507255554  
epoch 48 Acc : 0.9833860993385315  
epoch 48 Pre : 0.92770916223526  
* * *
```

شکل 14 نتایج برای بهینه ساز adam و معماری چهارم

بهینه ساز SGD و معماری 5:

```
* * *  
epoch 48 train loss : 0.19294193808013188  
epoch 48 Model Saved  
epoch 48 test loss : 1.557877545115314  
epoch 48 F1 : 0.9251857995986938  
epoch 48 Acc : 0.9851860404014587  
epoch 48 Pre : 0.934065043926239  
* * *
```

شکل 15 نتایج برای بهینه ساز SGD و معماری پنجم

بهینه ساز SGD و معماری 4:

```
* * *  
epoch 49 train loss : 0.4132081560933514  
epoch 49 Model Saved  
epoch 49 test loss : 1.6494329202024243  
epoch 49 F1 : 0.8631143569946289  
epoch 49 Acc : 0.9737637639045715  
epoch 49 Pre : 0.900692343711853  
* * *
```

شکل 16 نتایج برای بهینه ساز SGD و معماری چهارم

جدول 3 نتایج نهایی

-	Architecture 4			Architecture 5		
Optimizer	F1	Accuracy	Precision	F1	Accuracy	Precision
Adam	91.57	98.33	92.77	92.53	98.52	93.78
SGD	86.31	97.37	90.06	92.51	98.51	93.40

از دو جدول نتیجه می گیریم که بهینه ساز آدام به طور کلی نتایج بهتری داشته و همچنین معماری پنجم به علت تعداد Feature Map بیشتر و همچنین کرنل سایز بزرگتر، نتایج بهتری به نسبت معماری چهارم داشته است.

2-6. استفاده از دراپ اوت

علت استفاده از dropout این است که از overfit کردن شبکه جلوگیری شود. در واقع dropout سبب می شود که بخشی از نورون ها در ترین نباشند و بقیه نورون ها در ترین بوده و تعداد پارامتر ها کمتر می شود که موجب می شود شبکه دیر تر overfit شود و فقط در هنگام train وجود دارد و موقع test همه نورون ها شرکت دارند.