تمرین پنجم شبکه عصبی

سوال اول

A) عملکرد شبکه های CNN به طور ذاتی نسبت به view تغییر نمی کنند ولی سطح این تغییرناپذیری بستگی به نحوه ساختار و آموزش شبکه دارد.

چگونه شبکه های CNN نسبت به view تغییر ناپذیرند:
شبکه های CNN به طور ذاتی در Translation Invariance خوب هستند و فیلترها بر روی ورودی حرکت می کنند و به شبکه اجازه می دهند ویژگی هایی مانند لبه ها، بافت ها را بدون توجه به موقعیت دقیق آن ها در تصویر شناسایی کند.
شبکه های CNN یک روند سلسله مراتبی برای تشخیص ویژگی ایجاد می کنند و به این صورت است که لایه های پایین تر ویژگی های ساده ای مانند لبه ها یا گوشه ها را شناسایی می کنند در حالی که لایه های بالاتر الگوهای پیچیده تر و بخش های اشیا را شناسایی می کنند و مدل را در برابر تغییرات خاص در view مقاوم می سازند.
لایه های Pooling (مانند max pooling ، average pooling) ابعاد فضای ویژگی را کاهش می دهند. این فرآیند به شبکه کمک می کند تا نسبت به جابجایی های کوچک یا چرخش ها در ورودی بی تفاوت شود، زیرا مکان دقیق یک ویژگی کمتر اهمیت پیدا می کند.

برای تغییر نکردن این شبکه نسبت به view نیز می توانیم از تکنیک هایی مانند چرخش، مقیاس بندی، برعکس کردن و برش تصادفی استفاده کنیم تا شبکه را در طول آموزش به view های مختلف تعمیم بدهیم.

B) یک لایه کانولوشنی در CNN با چندین هایپرپارامتر مشخص می شود که عملکرد و ساختار آن را تعریف می کنند. این هایپرپارامتر ها تعیین می کنند که لایه چگونه داده های ورودی را پردازش کرده و ویژگی ها را استخراج می کند. در زیر هایپرپارامترهای کلیدی را توضیح می دهیم:

1. **تعداد فیلترها (کرنل ها) :**
هر فیلتر یاد می گیرد که یک ویژگی خاص را شناسایی کند مثل لبه ها و افزایش تعداد فیلترها

ظرفیت تشخیص ویژگی های لایه را افزایش می دهد اما هزینه محاسباتی را نیز افزایش می دهد.

2. اندازه فیلتر :

ارتفاع و عرض فیلتر را مشخص می کند که فیلترهای کوچکتر مانند ۳x۳ جزئیات ریز را ثبت می کنند در حالی که فیلترهای بزرگتر ویژگی های بزرگتری را ثبت می کنند.

3. Stride :

اندازه گام فیلتر را هنگام حرکت بر روی تصویر ورودی تعریف می کند که اگر stride برابر یک باشد فیلتر را یک پیکسل در هر بار حرکت می دهد که باعث می شود چیزی از قلم نیوفتد ولی گام های بزرگ تر هزینه محاسباتی را کاهش می دهند اما ممکن است جزئیات ریزتر را از دست بدهند.

4. Padding :

پیکسل های اضافی را در اطراف لبه های ورودی اضافه می کند که استخراج ویژگی های بهتر در مرزهای تصویر را ممکن می سازد .انواع پدینگ عبارت است از Same Padding و Valid Padding.

5. Dilation :

فاصله بین عناصر فیلتر را مشخص می کند که مفید برای بزرگ کردن میدان پذیرش بدون افزایش اندازه فیلتر یا افزودن پارامترها است.

6. عمق ورودی (کانال ها) :

مربوط به تعداد کانال ها در تصویر ورودی است که برای تصاویر RGB ۳ کانال و برای تصاویر خاکستری ۱ کانال است. عمق فیلتر باید با عمق ورودی مطابقت داشته باشد.

7. تابع فعال ساز :

توابع فعال ساز غیرخطی بودن را معرفی می کنند و به شبکه اجازه می دهند الگوهای پیچیده را یاد بگیرد.

8. مقداردهی اولیه وزن ها:

مقداردهی اولیه مناسب برای جلوگیری از مشکلاتی مانند vanishing در الگوریتم گرادیان ضروری است.

C) 3 دلیل استفاده از maxpooling :

1. کاهش ابعاد :
ابعاد فضای feature map (عرض و ارتفاع) را کاهش می دهد در حالی که اطلاعات کلیدی را حفظ می کند و مزیت آن هزینه محاسباتی کمتر و مصرف حافظه کمتر در لایه های بعدی است.

2. **جلوگیری از Overfitting :**
با کاهش تعداد پارامترها و ابعاد feature map مکس پولینگ از Overfitting شبکه روی داده های آموزشی جلوگیری می کند.

3. **کاهش پیچیدگی محاسباتی :**
feature map های کوچکتر منجر به کاهش بار محاسباتی برای لایه های بعدی می شوند.

D) 2 دلیل برای استفاده نکردن از max pooling

1. از دست دادن اطلاعات مکانی :
max pooling ابعاد مکانی feature map را کاهش می دهد و اطلاعات دقیق مکان ویژگی ها در ناحیه پولینگ را نادیده می گیرد که برای وظایفی که نیاز به درک دقیق مکان آن ویژگی دارند، مانند مکان یابی اشیا، تقسیم بندی معنایی، یا پردازش تصویر با وضوح بالا، max pooling می تواند عملکرد را کاهش دهد.

2. عدم توانایی در تشخیص ویژگی های ترکیبی :

Max pooling فقط بالاترین مقدار را از هر ناحیه انتخاب می کند و بقیه ویژگی ها را نادیده می گیرد که این روش می تواند ویژگی های ترکیبی و پیچیده ای که در مقادیر کمتر ولی مهم تر در یک منطقه وجود دارند را از دست بدهد. برای مثال وقتی ویژگی های مهم مختلف در کنار یکدیگر هستند max pooling ممکن است نتواند آن ها را به درستی تشخیص دهد.

E) محاسبه ماتریس خروجی:

1	3	1
0	-1	1
2	2	-1

ماتریس ورودی

-2	-2	1
-2	0	1
1	1	1

فیلتر یا کرنل

0	0	0	0	0
0	1	3	1	0
0	0	-1	1	0
0	2	2	-1	0
0	0	0	0	0

ورودی با پدینگ

A_{11}	A_{12}	A_{13}
A_{21}	A_{22}	A_{23}
A_{31}	A_{32}	A_{33}

ماتریس خروجی

$A_{11} = (0)*(-2) + (0)*(-2) + (0)*(1) + (0)*(-2) + (1)*(0) + (3)*(1) + (0)*(1) + (0)*(1) + (-1)*(1) = 2$

$A_{12} = (0)*(-2) + (0)*(-2) + (0)*(1) + (1)*(-2) + (3)*(0) + (1)*(1) + (0)*(1) + (-1)*(1) + (1)*(1) = -1$

$A_{13} = (0)*(-2) + (0)*(-2) + (0)*(1) + (3)*(-2) + (1)*(0) + (0)*(1) + (-1)*(1) + (1)*(1) + (0)*(1) = -6$

$A_{21} = (0)*(-2) + (1)*(-2) + (3)*(1) + (0)*(-2) + (0)*(0) + (-1)*(1) + (0)*(1) + (2)*(1) + (2)*(1) = 4$

$A_{22} = (1)*(-2) + (3)*(-2) + (1)*(1) + (0)*(-2) + (-1)*(0) + (1)*(1) + (2)*(1) + (2)*(1) + (-1)*(1) = -3$

$A_{23} = (3)*(-2) + (1)*(-2) + (0)*(1) + (-1)*(-2) + (1)*(0) + (0)*(1) + (2)*(1) + (-1)*(1) + (0)*(1) = -5$

$A_{31} = (0)*(-2) + (0)*(-2) + (-1)*(1) + (0)*(-2) + (2)*(0) + (2)*(1) + (0)*(1) + (0)*(1) + (0)*(1) = 1$

$A_{32} = (0)*(-2) + (-1)*(-2) + (1)*(1) + (2)*(-2) + (2)*(0) + (-1)*(1) + (0)*(1) + (0)*(1) + (0)*(1) = -2$

$A_{33} = (-1)*(-2) + (1)*(-2) + (0)*(1) + (2)*(-2) + (-1)*(0) + (0)*(1) + (0)*(1) + (0)*(1) + (0)*(1) = -4$

جواب نهایی:

2	-1	-6
4	-3	-5
1	-2	-4

سوال 2

A) انتخاب رزولوشن تصویر در یادگیری عمیق به دلایل مختلفی مهم است که عبارت است از:

1. تعادل بین کارایی محاسباتی و عملکرد مدل
کاهش رزولوشن تصاویر به این معنا است که تعداد پیکسل ها در داده های ورودی را کاهش می دهد که این امر بار محاسباتی و زمان آموزش را کاهش می دهد. با این حال کاهش بیش از حد وضوح می تواند منجر به از دست رفتن جزئیات مهم لازم برای طبقه بندی شود که ممکن است عملکرد مدل را کاهش دهد و از طرفی دیگر رزولوشن بالاتر جزئیات بیشتری را حفظ می کند که ممکن است برای تمایز بین کلاس ها مرتبط باشند. با این حال، این امر به طور قابل توجهی حجم محاسباتی را افزایش می دهد..

2. حفظ ویژگی های مرتبط
اشیای تاریخی اغلب دارای جزئیات منحصر به فرد و ظریف مانند حکاکی ها، بافت ها یا الگوهای طراحی خاص هستند که برای تمایز بین دوره ها مهم هستند. اگر رزولوشن خیلی پایین باشد این جزئیات ممکن است از بین بروند و توانایی مدل در یادگیری الگوهای معنادار را مختل کنند.

3. محدودیت های سخت افزاری
انتخاب رزولوشن باید با قابلیت های سخت افزاری هماهنگ باشد چرا که رزولوشن های بالاتر به GPU ها یا TPU هایی با حافظه بیشتر نیاز دارند. این دیتاست تقریبا ۵۰۰۰ تصویر دارد و مدیریت بارگذاری داده ها و آموزش مدل با رزولوشن های بسیار بالا ممکن است چالش برانگیز باشد.

4. معماری مدل
بسیاری از مدل های یادگیری عمیق pretrained مانند ResNet و EfficientNet برای رزولوشن های ورودی خاص طراحی شده اند. استفاده از رزولوشنی خارج از این استاندارد ها ممکن است نیاز به تغییرات معماری یا fine-tuning داشته باشد که می تواند بر فرآیند آموزش و کیفیت نتایج تاثیر بگذارد.

B) برای تقسیم بندی دیتاست از آنجایی که داده های آموزشی باید بیشتر باشد ما 70 درصد برای train و 15 درصد برای validation و 15 درصد دیگر را برای تست در نظر میگیریم.

C) این مشکل نوعی Data Leakage است و عدم نمایندگی مجموعه داده ها است که اغلب به آن عدم تطابق دامنه یا عدم تطابق توزیع گفته می شود.

مدلی که فقط بر روی تصاویر روزانه آموزش دیده است ممکن است ویژگی های خاص مانند نورپردازی و سایه ها را یاد بگیرد و نه ویژگی های عمومی و ذاتی اشیا تاریخی. این منجر به مدلی می شود که در تصاویر شبانه در مجموعه تست عملکرد ضعیفی دارشته باشد حتی اگر اشیا از همان کلاس های تاریخی باشند.
مدل به شرایط مجموعه آموزشی که در این مثال عکس های فقط در روز است بیش از حد تطبیق می یابد و قادر به تعمیم نیست.

حالا چگونه آن را اصلاح کنیم:
باید اطمینان حاصل کنیم که تصاویر روز و شب در تمام تقسیم بندی ها (train، validation، test) وجود دارند و نسبت های آن ها در این تقسیم بندی ها مشابه باشد .به عنوان مثال اگر مجموعه داده شامل ٪۳۰ تصاویر در شب باشد هدف این است که در هر مجموعه تقریبا ٪۳۰ تصاویر در شب وجود داشته باشد.
اگر تعداد تصاویر در شب به طور قابل توجهی کمتر است باید عکس های در روز را با استفاده از چند تکنینک به شرایط عکس های در شب شبیه سازی کنیم.

تکنیک ها شامل:
کاهش روشنایی.
افزودن افکت های شبانه مصنوعی.

یک راه دیگر این است که مدل را طوری آموزش دهیم که به طور صریح شرایط نوری را با افزودن متادیتا به عنوان ورودی در نظر بگیریم.

D) Data Augmentation چیست؟

یک تکنیک است که برای گسترش مصنوعی حجم و تنوع یک دیتاست است که با اعمال تغییرات به داده های موجود بدون تغییر برچسب های آن استفاده می شود. این روش تغییراتی را شبیه سازی می کنند که می توانند در سناریوهای دنیای واقعی رخ دهند و به مدل کمک می کنند تا بهتر تعمیم یابد و در عین حال از overfitting جلوگیری کند. Data Augmentation به ویژه زمانی مفید است که دیتاست موجود کوچک باشد.

۳ تکنیک برای Data Augmentation:

1- اعمال تغییرات بر روی تصاویر مانند:

- چرخاندن عکس: (horizontal , vertical)
- چرخش عکس: مثل چرخاندن تصاویر به صورت تصادفی 15 درجه به سمت راست یا چپ
- بزرگ و کوچک کردن: برای مثال zoom in یا zoom out
- برش عکس

2- تغییر رنگ و کنتراست:

- تنظیم روشنایی: روشنایی را افزایش یا کاهش دهید.
- تنظیم کنتراست: تفاوت بین نواحی روشن و تاریک را افزایش یا کاهش دهید.
- تغییر رنگ: به طور تصادفی رنگ یا کانال های رنگی را تغییر دهید.

3- تزریق نویز:

نویز تصادفی اضافه کنید تا نواقص موجود در تصویر را شبیه سازی کنید. این روش به مدل کمک می کند تا ویژگی های پایدار و مقاومی را یاد بگیرد که کمتر تحت تاثیر تغییرات کوچک در داده ها قرار می گیرند.

انواع نویز :

Gaussian

salt-and-pepper

speckle noise

(E

Conv5-10 (first):

- Memory: $\frac{(32 - 5 + (2 \times 2))}{1} + 1 = 32 \rightarrow 32 \times 32 \times 10$
- Learnable Parameters: $(5 \times 5 \times 1 + 1) \times 10 = 260$

Pool_1:

- Memory: $\frac{32}{2} \times \frac{32}{2} \times 10 = 16 \times 16 \times 10$
- Learnable Parameters: -

Conv5-10 (second):

- Memory: $\frac{(16 - 5 + (2 \times 2))}{1} + 1 = 16 \rightarrow 16 \times 16 \times 10$
- Learnable Parameters: $(5 \times 5 \times 10 + 1) \times 10 = 2510$

Pool_2:

- Memory: $\frac{16}{2} \times \frac{16}{2} \times 10 = 8 \times 8 \times 10$
- Learnable Parameters: -

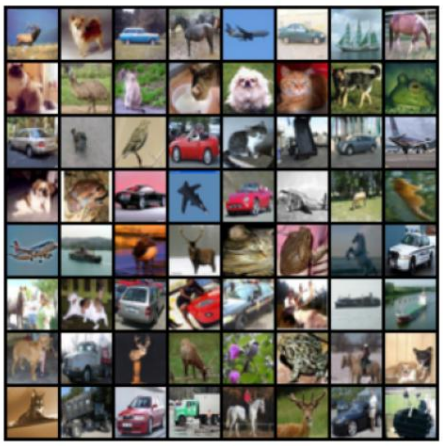
FC-10:

- Input size: $8 \times 8 \times 10 = 640$
- Learnable Parameters: $640 \times 10 + 10 = 6410$

c	Learnable Parameters	Memory
Input	-	$32 \times 32 \times 1$
Conv5-10	260	$32 \times 32 \times 10$
Pool2	-	$16 \times 16 \times 10$
Conv5-10	2510	$16 \times 16 \times 10$
Pool2	-	$8 \times 8 \times 10$
FC-10	6410	10

سوال 3

A) در نوت بوک CIFAR-10 پس از ایمپورت کتابخانه های لازم و دانلود دیتاست CIFAR-10 داده های train و test را جدا شده است.



در ادامه با استفاده از کتابخانه matplotlib نمونه هایی از عکس ها نمایش داده شده.

```
# We used data augmentation here, what is data augmentation?
train_transform = Compose([
    transforms.RandomHorizontalFlip(p=0.5),
    transforms.RandomCrop(32, padding=4),
    transforms.ToTensor(),
    transforms.Normalize([0, 0, 0], [1, 1, 1])
])

test_transform = Compose([
    transforms.ToTensor(),
    transforms.Normalize([0, 0, 0], [1, 1, 1])
])
```

سپس از تکنیک Data Augmentation استفاده شده است که به شرح زیر است:

افزایش داده (Data Augmentation):

این روش به مجموعه ای از تکنیک ها اشاره دارد که هدف آن افزایش تنوع نمونه های آموزشی است بدون اینکه نیاز باشد به جمع آوری داده های جدید فکر کنیم. این کار با استفاده از انجام تغییرات تصادفی مانند چرخش، برش، تغییر اندازه، یا تغییر رنگ روی داده هایی که داریم انجام می شود.

افزایش داده به مدل کمک می کند که با تنوع بیشتری از داده ها روبه رو شود تا در نتیجه مدل بهبود یابد و قابلیت تعمیم (Generalization) بهتری پیدا کند.

در این کد داده های train با استفاده از برخی تکنیک های Data Augmentation استفاده شده که به شرح زیر است:

RandomHorizontalFlip(p=0.5) : به صورت رندوم با احتمال 50 درصد تصویر را به صورت افقی برعکس می کند.

RandomCrop(32, padding=4): به صورت رندوم تصویر را برش می زند و از پدینگ 4 پیکسلی برای اطمینان از اینکه تصویر برش خورده اندازه مورد نیاز (32x32) را داشته باشد استفاده می کند.

در ادامه به tensor تبدیل می شوند و نرمال سازی می شوند.

```python
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(3,   64,  3)
        self.conv2 = nn.Conv2d(64,  128, 3)
        self.conv3 = nn.Conv2d(128, 256, 3)
        self.pool = nn.MaxPool2d(2, 2)
        self.fc1 = nn.Linear(64 * 4 * 4, 128)
        self.fc2 = nn.Linear(128, 256)
        self.fc3 = nn.Linear(256, 10)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = self.pool(F.relu(self.conv3(x)))
        x = x.view(-1, 64 * 4 * 4)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return F.log_softmax(x, dim=1)
```

تعریف لایه ها به شرح زیر است:

:Conv1

- تعداد ورودی: 3 تا (RBG)
- تعداد فیلترها (کانال های خروجی): 64
- اندازه فیلتر: 3 در 3

:Conv2

- تعداد ورودی: 64 تا (خروجی لایه conv1)
- تعداد فیلترها (کانال های خروجی): 128
- اندازه فیلتر: 3 در 3

:Conv3

- تعداد ورودی: 128 تا (خروجی لایه conv2)
- تعداد فیلترها (کانال های خروجی): 256
- اندازه فیلتر: 3 در 3

به صورت پیشفرض پدینگ 0 و استراید 1 در نظر گرفته می شود.

لایه Pooling:

نوع آن: nn.MaxPool2d

- اندازه کرنل: 2 در 2
- استراید: 2 (پیمایش با گام ۲)

لایه های Fully Connected :

FC1:

- تعداد ورودی: 1024 تا (تعداد ویژگی های خروجی از لایه های conv و pooling)
- تعداد خروجی: 128

FC2:

- تعداد ورودی: 128 تا (خروجی لایه fc1)
- تعداد خروجی: 256

FC3:

- تعداد ورودی: 256 تا (خروجی لایه fc2)
- تعداد خروجی: 10

در ادامه تابع فعالساز ReLU برای لایه های Conv و Fully Connected استفاده شده و در نهایت برای خروجی نهایی از log_softmax برای پیش بینی احتمالات استفاده شده است.

```
criterion = nn.CrossEntropyLoss() # also can use "Mean Squared Error Loss"
optimizer = optim.SGD(model.parameters(), lr=0.001, momentum=0.9)#also can use "Adam Optimizer"
val_per_epoch = 5
n_epochs = 30
batch_size = 4
```

سپس از CrossEntropyLoss برای تابع زیان و از بهینه ساز SGD با نرخ یادگیری 0.001 و momentum 0.9 استفاده شده است.

در نهایت مدل آموزش دیده می شود و نتایج با دقت 82 درصد به دست آمده است.

```
Accuracy: 0.821600
```

برای مدل بعدی از dropout استفاده شده است که به شرح زیر است:

Dropout چیست؟

Dropout یک تکنیک است در شبکه های عصبی است که به منظور جلوگیری از overfitting در فرایند training استفاده می شود. در هر مرحله به صورت تصادفی تعدادی از نورون ها (neurons) به طور موقت غیرفعال می شوند. یعنی که نورون ها و اتصالات بین آن ها حذف می شوند تا شبکه نتواند به شدت به ویژگی های خاصی در داده های train وابسته شود و به جای آن ویژگی های عمومی تری را یاد بگیرد.

مزایا:

- جلوگیری از Overfitting
- بهبود تعمیم پذیری (Generalization)
- کمک به استفاده بهینه از منابع

معایب:

- آموزش کندتر
- نیاز به تنظیم دقیق نرخ Dropout

تاثیر استفاده از Dropout :

مدل احتمالا در ابتدا عملکرد ضعیف تری در داده های train خواهد داشت زیرا بخشی از نورون ها غیرفعال شده اند و نمی توانند به سرعت ویژگی های خاص را یاد بگیرند.اما در طول زمان مدل قادر خواهد بود که ویژگی های عمومی تری یاد بگیرد و در نهایت دقت بهتری در داده های تست خواهد داشت.

بدون استفاده از Dropout :

مدل می تواند در ابتدا دقت بالاتری در داده های آموزشی داشته باشد اما در طول زمان ممکن است بیش از حد روی داده های آموزشی آموزش ببیند و دقت مدل روی داده های تست یا داده های جدید کاهش یابد.

در این مدل از 3 لایه dropout با که 2 تای آن با نرخ 20 درصد و یکی 10 درصد استفاده شده است.

B) برای این بخش ما از دیتاست MNIST استفاده کردیم که نمونه های آن در عکس زیر قابل مشاهده است:

```
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5,), (0.5,))
])
train_loader = torch.utils.data.DataLoader(dataset=trainset, batch_size=64, shuffle=True)
test_loader = torch.utils.data.DataLoader(dataset=valset, batch_size=64, shuffle=True)
```

در این بخش عکس ها به تنسور تبدیل میشوند و نرمال سازی می شوند. در ادامه داده ها را در دسته های کوچک (هر دسته 64 نمونه) با ترتیب تصادفی فراهم می کنیم.

```
class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        self.conv1 = nn.Conv2d(1, 32, kernel_size=3, padding=1)
        self.conv2 = nn.Conv2d(32, 64, kernel_size=3, padding=1)
        self.pool = nn.MaxPool2d(2, 2)
        self.fc1 = nn.Linear(14 * 14 * 64, 128)
        self.fc2 = nn.Linear(128, 10)
        self.relu = nn.ReLU()
        self.dropout = nn.Dropout(0.5)

    def forward(self, x):
      x = self.relu(self.conv1(x))
      x = self.pool(self.relu(self.conv2(x)))
      x = x.view(x.size(0), -1)
      x = self.dropout(self.relu(self.fc1(x)))
      x = self.fc2(x)
      return x
```

تعریف لایه ها به شرح زیر است:

:Conv1

- تعداد ورودی: 1 (grayscale)
- تعداد فیلترها (کانال های خروجی): 32
- اندازه فیلتر: 3 در 3
- پدینگ 1

:Conv2

- تعداد ورودی: 32 تا (خروجی لایه conv1)
- تعداد فیلترها (کانال های خروجی): 64
- اندازه فیلتر: 3 در 3
- پدینگ 1

به صورت پیش فرض استراید 1 در نظر گرفته می شود.

لایه Pooling:

نوع آن: nn.MaxPool2d

- اندازه کرنل: 2 در 2
- استراید: 2 (پیمایش با گام ۲)

لایه های Fully Connected :

FC1:

- تعداد ورودی: 14 * 14 * 64
- تعداد خروجی: 128

FC2:

- تعداد ورودی: 128 تا (خروجی لایه fc1)
- تعداد خروجی: 10

در نهایت از تابع فعال ساز Relu و لایه ی Dropout با نرخ 50 درصد استفاده شده است.

در تابع forward از لایه اول conv رد میشود و سپس از Relu رد می شود.

در ادامه از لایه دوم conv رد میشود و روی آن Relu اعمال میشود و سپس Pooling انجام می شود.

سپس داده ی چندبعدی به یک بردار تک بعدی مسطح تبدیل می شود سپس خروجی وارد لایه ی fc1شده و تابع ReLU و Dropout اعمال می شود.

در نهایت به لایه ی خروجی fc2 منتقل می شود که احتمال تعلق به هر یک از 10 کلاس را تولید می کند.

```
epochs = 15
batch_size = 64
learning_rate = 0.001
model = CNN().to(device)
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(), lr=learning_rate, momentum=0.9)
```

در این بخش تعداد تکرار دوره آموزشی و Batch Size و نرخ یادگیری را ست میکنیم. مدل را تعریف می کنیم و به gpu میفرستیم. از تابع هزینه CrossEntropyLoss استفاده کردیم و بهینه ساز SGD را ست کردیم.

```
for epoch in range(epochs):
    model.train()
    running_loss = 0.0
    for images, labels in train_loader:
        images, labels = images.to(device), labels.to(device)

        optimizer.zero_grad()
        outputs = model(images)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        running_loss += loss.item()
    print(f"Epoch [{epoch+1}/{epochs}], Loss: {running_loss / len(train_loader):.4f}")
```

در این کد به تعداد تکرار دوره آموزشی (epochs) مدل ما آموزش می بیند. سپس خروجی مدل محاسبه و خطا اندازه گیری می شود. گرادیان ها محاسبه و وزن ها به روزرسانی می شوند. خطای هر دسته جمع آوری شده و میانگین آن برای کل دوره محاسبه می شود.

در نهایت خطای میانگین هر دوره در آرایه ای ذخیره و نمایش داده می شود.

خروجی کد بالا :

```
Epoch [1/15], Loss: 0.7870
Epoch [2/15], Loss: 0.3164
Epoch [3/15], Loss: 0.2647
Epoch [4/15], Loss: 0.2402
Epoch [5/15], Loss: 0.2211
Epoch [6/15], Loss: 0.2049
Epoch [7/15], Loss: 0.1967
Epoch [8/15], Loss: 0.1872
Epoch [9/15], Loss: 0.1792
Epoch [10/15], Loss: 0.1738
Epoch [11/15], Loss: 0.1686
Epoch [12/15], Loss: 0.1612
Epoch [13/15], Loss: 0.1558
Epoch [14/15], Loss: 0.1510
Epoch [15/15], Loss: 0.1459
```

```python
#Review the results
from sklearn.metrics import confusion_matrix ,f1_score ,precision_score ,recall_score, accuracy_score

def test_label_predictions(model, device, test_loader):
    model.eval()
    actuals = []
    predictions = []
    with torch.no_grad():
        for data, target in test_loader:
            data, target = data.to(device), target.to(device)
            output = model(data)
            prediction = output.argmax(dim=1, keepdim=True)
            actuals.extend(target.view_as(prediction))
            predictions.extend(prediction)
    return [i.item() for i in actuals], [i.item() for i in predictions]

actuals, predictions = test_label_predictions(model, device, test_loader)
print('Accuracy: %f' % accuracy_score(actuals, predictions))
print('Confusion matrix:')
print(confusion_matrix(actuals, predictions))
print('F1 score: %f' % f1_score(actuals, predictions, average='weighted'))
print('Precision: %f' % precision_score(actuals, predictions, average='weighted'))
print('Recall: %f' % recall_score(actuals, predictions, average='weighted'))
```

در نهایت مدل آموزش دیده و نوبت به ارزیابی مدل میرسد که از همان تابعی که در فایل CIFAR-10 بود استفاده کردیم که به دقت زیر رسیدیم:

```
Accuracy: 0.975000
Confusion matrix:
[[ 971    0    1    0    0    2    3    1    2    0]
 [   0 1120    3    1    0    0    3    1    7    0]
 [   7    0 1007    3    3    0    1    4    7    0]
 [   1    0    2  991    0    4    0    8    4    0]
 [   0    0    5    0  952    0    3    3    2   17]
 [   2    0    0    8    0  872    4    1    3    2]
 [   5    3    0    1    4    4  937    1    3    0]
 [   2    6   15    1    0    0    0  992    1   11]
 [   2    1    2    8    3    2    3    4  939   10]
 [   3    5    0    8   12    3    1    4    4  969]]
F1 score: 0.974991
Precision: 0.975006
Recall: 0.975000
```

سوال 4

در این بخش ما با استفاده از **Transfer Learning** سعی کردیم دقت مدل برای دیتاست Cifar-10 را بهبود دهیم. در فایل keras دیتاست دانلود شده و به دو مجموعه از 5 کلاس اول و دوم تقسیم شده است. سپس بخش های train و val و test جدا شده اند و باید روی مجموعه اول با توجه با لایه های گفته شده یک شبکه cnn با کراس ایجاد کنیم که به شرح زیر است:

```
from keras.layers import MaxPooling2D
model = Sequential()
model.add(Conv2D(32, kernel_size=(4, 4), input_shape=(32, 32, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(32, kernel_size=(4, 4), input_shape=(32, 32, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dense(5, activation='softmax'))
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

در اینجا ما برای لایه آخر از تابع فعالساز softmax استفاده کردیم. در ادامه از بهینه ساز adam و تابع زیان categorical_crossentropy استفاده کردیم.

در ادامه مدل را آموزش می دهیم که نتیجه آن به شرح زیر است:

```
Epoch 1/10
200/200 - 21s - 105ms/step - accuracy: 0.5411 - loss: 1.1082 - val_accuracy: 0.6202 - val_loss: 0.9377
Epoch 2/10
200/200 - 22s - 109ms/step - accuracy: 0.6431 - loss: 0.8935 - val_accuracy: 0.6688 - val_loss: 0.8527
Epoch 3/10
200/200 - 42s - 208ms/step - accuracy: 0.6780 - loss: 0.8194 - val_accuracy: 0.6874 - val_loss: 0.8061
Epoch 4/10
200/200 - 39s - 194ms/step - accuracy: 0.7090 - loss: 0.7501 - val_accuracy: 0.7110 - val_loss: 0.7498
Epoch 5/10
200/200 - 22s - 110ms/step - accuracy: 0.7297 - loss: 0.6993 - val_accuracy: 0.7038 - val_loss: 0.7529
Epoch 6/10
200/200 - 41s - 206ms/step - accuracy: 0.7481 - loss: 0.6560 - val_accuracy: 0.7250 - val_loss: 0.7205
Epoch 7/10
200/200 - 20s - 101ms/step - accuracy: 0.7589 - loss: 0.6311 - val_accuracy: 0.7166 - val_loss: 0.7578
Epoch 8/10
200/200 - 22s - 109ms/step - accuracy: 0.7844 - loss: 0.5715 - val_accuracy: 0.7488 - val_loss: 0.6856
Epoch 9/10
200/200 - 20s - 98ms/step - accuracy: 0.7976 - loss: 0.5365 - val_accuracy: 0.7540 - val_loss: 0.6449
Epoch 10/10
200/200 - 21s - 105ms/step - accuracy: 0.8098 - loss: 0.5038 - val_accuracy: 0.7448 - val_loss: 0.6883


 --------------------

Time taken for first training:  0:04:29.738770

 --------------------
```

سپس طبق سوال که گفته شده 5 لایه اول مدل قبلی را ثابت نگه داریم و دو لایه دیگر با مشخصات داده شده ایجاد کنیم به کد زیر رسیدیم:

```
previous_model = model
new_model = Sequential()
for layer in previous_model.layers[:5]:
    layer.trainable = False
    new_model.add(layer)

new_model.add(Dense(128, activation='relu'))
new_model.add(Dense(10, activation='softmax'))
new_model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

در اینجا 5 لایه اولی که از مدل قبلی train شده بودند freeze شده بود و وزن های آن ها تغییر نمی کنند و در این بخش نیز از بهینه ساز adam و تابع زیان categorical_crossentropy استفاده کردیم و مدل را با بخش دوم دیتاست آموزش می دهیم که نتیجه آن به شکل زیر است:

```
Epoch 1/10
200/200 - 8s - 38ms/step - accuracy: 0.7272 - loss: 0.7558 - val_accuracy: 0.7824 - val_loss: 0.5955
Epoch 2/10
200/200 - 9s - 46ms/step - accuracy: 0.7915 - loss: 0.5699 - val_accuracy: 0.7904 - val_loss: 0.5783
Epoch 3/10
200/200 - 11s - 53ms/step - accuracy: 0.8077 - loss: 0.5234 - val_accuracy: 0.7978 - val_loss: 0.5491
Epoch 4/10
200/200 - 11s - 55ms/step - accuracy: 0.8163 - loss: 0.5002 - val_accuracy: 0.8062 - val_loss: 0.5349
Epoch 5/10
200/200 - 11s - 53ms/step - accuracy: 0.8262 - loss: 0.4737 - val_accuracy: 0.8122 - val_loss: 0.5178
Epoch 6/10
200/200 - 6s - 28ms/step - accuracy: 0.8348 - loss: 0.4523 - val_accuracy: 0.8176 - val_loss: 0.5025
Epoch 7/10
200/200 - 10s - 51ms/step - accuracy: 0.8417 - loss: 0.4358 - val_accuracy: 0.8176 - val_loss: 0.5050
Epoch 8/10
200/200 - 11s - 56ms/step - accuracy: 0.8469 - loss: 0.4178 - val_accuracy: 0.8252 - val_loss: 0.4984
Epoch 9/10
200/200 - 11s - 56ms/step - accuracy: 0.8577 - loss: 0.3963 - val_accuracy: 0.8264 - val_loss: 0.4892
Epoch 10/10
200/200 - 9s - 43ms/step - accuracy: 0.8629 - loss: 0.3791 - val_accuracy: 0.8234 - val_loss: 0.4992


 --------------------

Time taken for final training:  0:01:36.176614

 --------------------
```

این کار باعث میشود به جای آموزش مجدد از صفر از ویژگی های استخراج شده توسط مدل قبلی برای سریع تر کردن آموزش استفاده شود. و هم چنین چون لایه های قبلی نیازی به آموزش نداشتند زمان کمتری صرف یادگیری شد.

در نهایت مدل جدید با تکیه بر لایه های ثابت شده و اضافه کردن لایه های تخصصی تر توانست دقت بیشتری در طبقه بندی ایجاد کند.

محمد حقیقت - 403722042