به نام خدا

درس یادگیری عمیق

تمرین چهارم

غزل زمانی نژاد

401722244

1. الف) کپسول مجموعه ای از نورون‌هاست که بردار فعالیت آنها نشان دهنده پارامترهای instantiation یک نوع خاص از entity (مانند یک شی یا قطعه ای از یک شی) است. از طول بردار فعالیت برای نشان دادن احتمال وجود موجودیت و از جهت گیری آن برای نشان دادن پارامترهای instantiation استفاده می کنیم. کپسول های فعال در یک سطح از طریق ماتریس های transformation، پارامترهای instantiation کپسول های سطح بالاتر را پیش بینی می کنند. هنگامی که چندین پیش‌بینی موافق باشند، یک کپسول سطح بالاتر فعال می شود. نحوه کار شبکه CapsNet به صورت زیر است:

ابتدا داده ورودی از چند لایه کانولوشنی عبور می کند تا نقشه ویژگی داده محاسبه شود. سپس نقشه های ویژگی به بردارهایی reshape می شوند و کپسول های اولیه را تشکیل می دهند. از یک لایه فعال سازی squashing استفاده می شود که این لایه طول بردار را به بازه ۰ تا ۱ مپ می کند (تا نشان دهنده احتمال وجود موجودیت باشد). Squashing بر سایر پارامترها (مثل جهت و اندازه) تاثیری نخواهد داشت. سپس تمامی کپسول های لایه اول باید خروجی کپسول های لایه بعدی را پیش بینی کنند. برای هر parent ممکن، کپسول یک «بردار پیش بینی» محاسبه می کند که حاصل ضرب خروجی خودش در ماتریس وزن است. اگر حاصل ضرب عددی بردار پیش بینی در خروجی مقدار والد مقدار بزرگی باشد، فیدبک top-down خواهیم داشت بدین معنی که ضریب coupling آن والد را بیشتر و سایر والدها را کمتر می کند. به این روش routing by agreement گفته می شود که از routing با روش max pooling بسیار موثرتر است.

ب) محدودیت CNN این است که نورون های آن بر اساس شانس تشخیص ویژگی خاص فعال می شوند. نورون ها ویژگی های featureها از قبیل جهت، اندازه، سرعت، رنگ و ... را در نظر نمی گیرند. یعنی بر روی رابطه میان ویژگی ها آموزش داده نشده اند. به طور مثال اگر به روابط spatial میان ویژگی ها (تصویر سمت چپ) و یا جهت گیری آنها (تصویر سمت راست) دقت می کرد و آنها را آموزش دیده بود تصاویر را به درستی طبقه بندی می کرد و دیگر نیازی به دیتاست های بزرگ augment شده از جهت های مختلف تصاویر نبود. این نوع از شبکه بدلیل محدودیت خود نمی تواند تصاویر زیر را به درستی برچسب بزند:

بدلیل استفاده از لایه max pooling اطلاعات در مورد مکان ویژگی ها از دست می رود. دلیل استفاده از لایه max pooling، بدست آوردن translation invariance است بدین معنی که شبکه کانولوشنی نمونه ها را بدون در نظر گرفتن میزان شیفت اطلاعات آنها طبقه بندی می کند.

ج) شبکه های کانولوشنی از تکرار آشکارسازهای آموخته شده برای یافتن ویژگی های محلی در داده استفاده می کنند و باعث می شود دانش مربوط به مقادیر وزن های خوبی که در یک موقعیت تصویر بدست آمده، در سایر موقعیت های تصویر نیز استفاده شود. به همین دلیل در شبکه های کپسولی برای همه لایه ها بجز لایه آخر از لایه کانولوشنی استفاده می شود.

در شبکه کپسولی همچون CNN، کپسول های سطح بالاتر نواحی بزرگتری از تصویر را پوشش می دهند. خروجی یک آشکارساز در شبکه کانولوشنی یک مقدار اسکالر است اما در شبکه کپسولی خروجی بردار کپسول است.

در شبکه کانولوشنی از لایه max pooling استفاده می شود که نسبت به شیفت حساس نیست اما اطلاعاتی از داده ورودی را دور می ریزد در حالیکه شبکه کپسولی از روش routing by agreement استفاده می کند و اطلاعات ویژگی های موجود در نواحی تصویر از بین نمی رود.

در کپسول های سطح پایین تر اطلاعات موقعیت به صورت place-coded و در کپسول های سطح بالاتر به صورت rate-coded ذخیره می شود

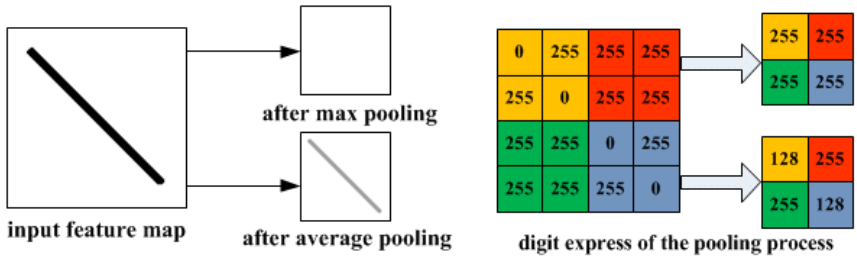
2. الف) نقشه ویژگی که پس از عبور از کرنل های لایه کانولوشنی بدست می آید، وابسته به مکان است. به طور مثال اگر در یک تصویر مکان یک شی را اندکی شیفت دهیم ممکن است دیگر توسط لایه کانولوشنی قابل تشخیص نباشد. این بدین معنی است که نقشه ویژگی مکان دقیق ویژگی های ورودی را حفظ می کند. دلیل استفاده از لایه ادغام Translational Invariance است یعنی حساسیت شبکه

CNN را نسبت به مکان کم می کند. مثلا اگر مکان شی تصویر را شیفت بدهیم، با وجود لایه ادغام ویژگی های آن همچنان قابل شناسایی هستند.

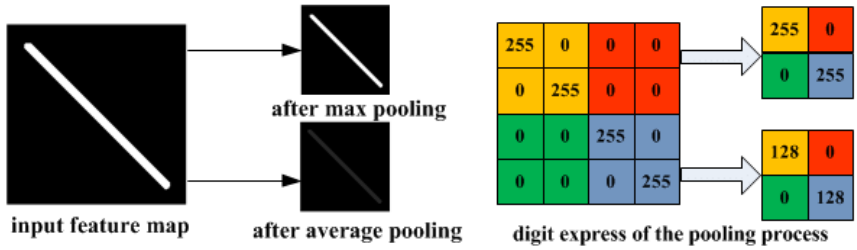
لایه ادغام حداکثر بیشترین مقادیر را در هر پنجره انتخاب می کند. این لایه برجسته ترین ویژگی های هر نقشه ویژگی را حفظ می کند و تصویر خروجی این لایه واضح تر (sharp) از تصویر اصلی است.

لایه ادغام میانگین، میانگین مقادیر هر پنجره را بدست می آورد. این لایه میانگینی از ویژگی های هر نقشه ویژگی را حفظ می کند و تصویر خروجی این لایه smoothتر از تصویر اصلی است و ماهیت ویژگی های تصویر را حفظ می کند.

در تصویر زیر ایراد هر دو لایه قابل مشاهده است. در تصویر a پس از استفاده از لایه ادغام حداکثر، عملا تصویر اصلی از بین رفته و یک صفحه سفید بدست آمده است. در تصویر b لایه ادغام میانگین سعی کرده میانگین مقادیر بخش های مختلف تصویر را بدست بیاورد و با این کار خط سفید موجود در تصویر به یک خط خاکستری تبدیل شده و به سختی قابل مشاهده است.



(a) Illustration of max pooling drawback



(b) Illustration of average pooling drawback

منبع: https://towardsai.net/p/l/introduction-to-pooling-layers-in-cnn

ب) پس از اعمال لایه ادغام، بدلیل اینکه stride = 2 است اندازه خروجی 2*2 خواهد بود اما تعداد کانال ورودی تغییر نخواهد کرد.

اعمال لایه ادغام حداکثر:

1	9	6	4
5	4	7	8
5	1	2	9
6	7	6	0

9	8
7	9

9	1	7	4
5	6	3	0
1	2	5	4
0	8	9	0

9	7
8	9

7	6	9	1
5	2	0	4
5	8	3	9
0	2	2	1

7	9
8	9

اعمال لایه ادغام میانگین:

1	9	6	4
5	4	7	8
5	1	2	9
6	7	6	0

4.75	6.25
4.75	4.25

9	1	7	4
5	6	3	0
1	2	5	4
0	8	9	0

5.25	3.5
2.75	4.5

7	6	9	1
5	2	0	4
5	8	3	9
0	2	2	1

5	3.5
3.75	3.75

ج) مقایسه میزان پارامترهای این مدل ها به صورت زیر است:

Nin < GoogleNet < AlexNet < VGG

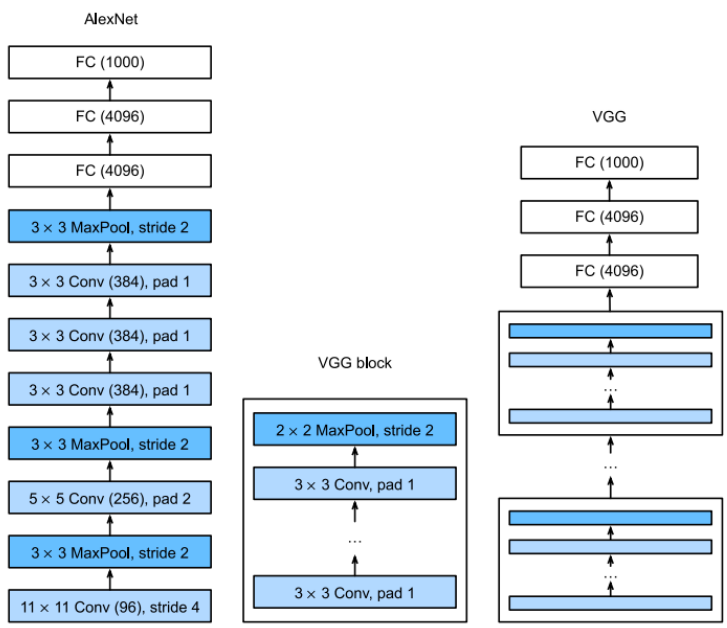
تعداد پارامترهای Nin و GoogleNet به میزان چشمگیری از دو مدل دیگر کمتر است.

در شبکه های کانولوشنی، تعداد بسیار زیادی از پارامترها مربوط به لایه های آخر (لایه های کاملا متصل که وظیفه طبقه بندی را دارند) می شود. شبکه NiN از لایه fully connected استفاده نمی کند و به جای آن از بلوک NiN با تعداد کانال خروجی برابر با تعداد کلاس ها به همراه یک لایه global average pooling استفاده می کند. پارامترهای learnable این شبکه فیلترهای لایه کانولوشنی هستند و استفاده از آنها به جای لایه کاملا متصل (که هر نورون به تمامی نورون های لایه قبل متصل است)، تعداد پارامترها را به طور قابل توجهی کاهش می دهد اما ممکن است زمان آموزش مدل افزایش یابد.

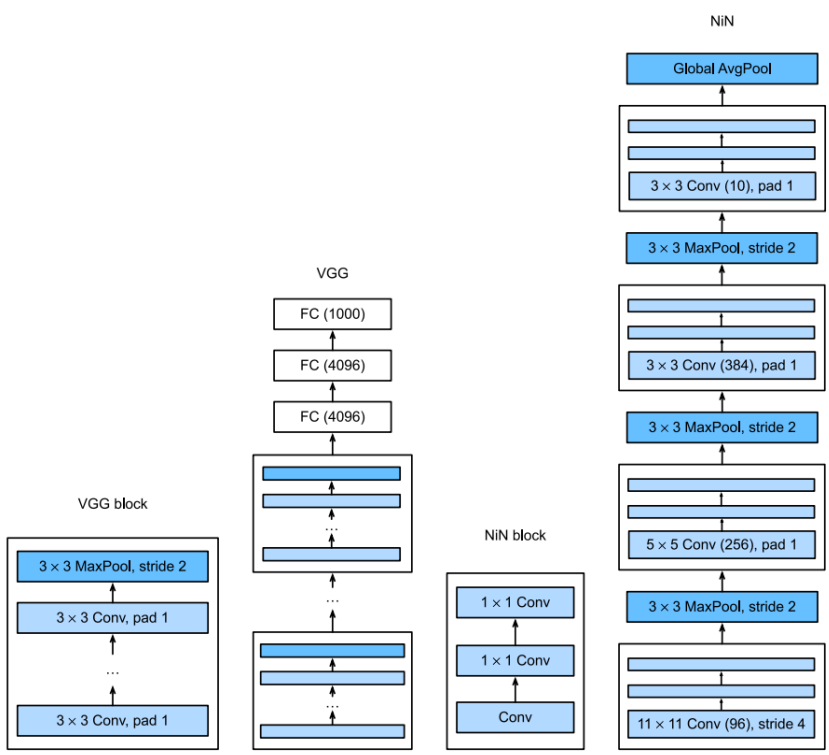
همچنین در شبکه GoogleNet از بلوک inception استفاده می شود. به صورت موازی از فیلترهای کوچک (1*1, 3*3, 5*5) و لایه های pooling استفاده می شود و تعداد پارامترهای آن کم است.

معماری شبکه AlexNet شباهت هایی به شبکه LeNet دارد ولی به دلیل ۳ لایه fully connected در انتهای شبکه، تعداد پارامترهای آن زیاد است (حدود ۶۰ میلیون).

شبکه VGG از تعداد بلوک VGG تشکیل شده است. لایه های کاملا متصل در آن مشابه لایه های AlexNet هستند ولی به دلیل تعداد زیاد لایه کانولوشنی، پارامترهای آن بیشتر از ۲برابر شبکه AlexNet است (حدود ۱۳۸ میلیون پارامتر).



مقایسه معماری AlexNet با VGG



مقایسه معماری VGG با NiN

$$\text{Output} = \left\lfloor \frac{n + 2p - k}{s} + 1 \right\rfloor$$

layer 1 $\xrightarrow{\text{pad=same}}$ output = $256 \times 256 \times 64$

$\#\text{params} = (3 \times 3 \times 3 + 1)\ 64$

layer 2 filter 5×5 $\xrightarrow{\text{dilation}=2}$ 9×9

$\text{out} = \left\lfloor \frac{256 - 9}{2} + 1 \right\rfloor = 124$ out $= 124 \times 124 \times 32$

$\#\text{params} = (5 \times 5 \times 64 + 1)\ 32$

layer 3 $\text{out} = \left\lfloor \frac{124 - 2}{2} + 1 \right\rfloor = 62$ out $= 62 \times 62 \times 32$

$\#\text{params} = 0$

layer 4 $\xrightarrow{\text{pad=same}}$ output $= 62 \times 62 \times 128$

$\#\text{params} = (3 \times 3 \times 32 + 1)\ 128$

layer 5 filter 5×5 $\xrightarrow{\text{dilation}=4}$ $5 + 3 \times 4 \Rightarrow 17 \times 17$

$\text{out} = \left\lfloor \frac{62 - 17}{2} + 1 \right\rfloor = 23$ out $= 23 \times 23 \times 64$

$\#\text{params} = (5 \times 5 \times 128 + 1)\ 64$

layer 6 $\text{out} = \left\lfloor \frac{23 - 2}{2} + 1 \right\rfloor = 11$ out $= 11 \times 11 \times 64$

$\#\text{params} = 0$

layer 7 $\xrightarrow{\text{pad=same}}$ out $= 11 \times 11 \times 256$

$\#\text{params} = (3 \times 3 \times 64 + 1)\ 256$

layer 8 filter 5×5 $\xrightarrow{\text{dilation}=2}$ 9×9

$\text{out} = \left\lfloor \frac{11 - 9}{2} + 1 \right\rfloor = 2$ out $= 2 \times 2 \times 128$

$\#\text{params} = (5 \times 5 \times 256 + 1)\ 128$

layer 9 $\text{out} = \left\lfloor \frac{2 - 2}{2} + 1 \right\rfloor = 1$ out $= 1 \times 1 \times 128$

$\#\text{params} = 0$

forward pass:

o_{11}	o_{12}
o_{21}	o_{22}

$$o_{11} = -1\times 1 + 2\times 3 + (-1)(-1) = 7$$

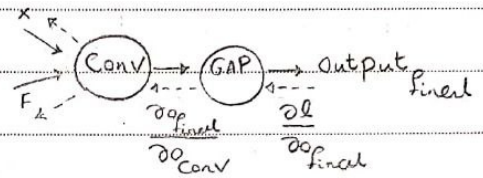
$$o_{12} = -1\times 2 + \underbrace{(-2)3}_{-6} + \underbrace{5(-2)}_{-10} = -18$$

$$o_{21} = -1(-1) + 5\times 3 + 3(-2) = 10$$

$$o_{22} = -1\times 5 + 3\times 3 = 4$$

7	-18
10	4

$\xrightarrow{GAP} \frac{3}{4}$



مشتق $\frac{\partial l}{\partial output_{final}} = 1$

$$\frac{\partial l}{\partial F} = \frac{\partial l}{\partial o_{final}} \cdot \frac{\partial o_{final}}{\partial o_{conv}} \cdot \frac{\partial o_{conv}}{\partial F}$$

$$\frac{\partial o_{final}}{\partial o_{conv}} = \frac{\partial}{\partial o_{conv}}\left(\frac{1}{n}\sum_{i=1}^{4} o_{conv_i}\right) \Rightarrow \begin{cases} \frac{\partial o_{final}}{\partial o_{11}} = \frac{1}{n} = \frac{1}{4} \\ \frac{\partial o_{final}}{\partial o_{12}} = \frac{1}{n} = \frac{1}{4} \\ \frac{\partial o_{final}}{\partial o_{21}} = \frac{1}{n} = \frac{1}{4} \\ \frac{\partial o_{final}}{\partial o_{22}} = \frac{1}{n} = \frac{1}{4} \end{cases}$$

Subject:
Year: Month: Date

$$\frac{\partial l}{\partial F_{11}} = \frac{\partial l}{\partial o_{11}} \cdot \overset{x_{11}}{\frac{\partial o_{11}}{\partial F_{11}}} + \frac{\partial l}{\partial o_{12}} \cdot \overset{x_{12}}{\frac{\partial o_{12}}{\partial F_{11}}} + \frac{\partial l}{\partial o_{21}} \cdot \overset{x_{21}}{\frac{\partial o_{21}}{\partial F_{11}}} + \frac{\partial l}{\partial o_{22}} \cdot \overset{x_{22}}{\frac{\partial o_{22}}{\partial F_{11}}}$$

$$= \frac{1}{4} \cdot 1 + \frac{1}{4} \times 2 + \frac{1}{4} \times (-1) + \frac{1}{4}(5) = 7/4 = 1.75$$

$$\frac{\partial l}{\partial F_{12}} = \frac{\partial l}{\partial o_{11}} \cdot x_{12} + \frac{\partial l}{o_{12}} x_{13} + \frac{\partial l}{\partial o_{21}} x_{22} + \frac{\partial l}{\partial o_{22}} \cdot x_{23}$$

$$= \frac{1}{4} \times 2 + \frac{1}{4} \times (-2) + \frac{1}{4}(5) + \frac{1}{4}(3) = 2$$

$$\frac{\partial l}{\partial F_{21}} = \frac{\partial l}{\partial o_{11}} \cdot x_{21} + \frac{\partial l}{\partial o_{12}} x_{22} + \frac{\partial l}{\partial o_{21}} x_{31} + \frac{\partial l}{\partial o_{22}} \cdot x_{32}$$

$$= \frac{1}{4} \times (-1) + \frac{1}{4}(5) + \frac{1}{4}(3) + \frac{1}{4}(0) = 7/4 = 1.75$$

$$\frac{\partial l}{\partial F_{22}} = \frac{\partial l}{\partial o_{11}} \cdot x_{22} + \frac{\partial l}{\partial o_{12}} x_{23} + \frac{\partial l}{\partial o_{21}} x_{32} + \frac{\partial l}{\partial o_{22}} \cdot x_{33}$$

$$= \frac{1}{4} \times 5 + \frac{1}{4} \times 3 + \frac{1}{4} \times 0 + \frac{1}{4} \times 1 = 9/4 = 2.25$$

$$\frac{\partial l}{\partial F} = \begin{bmatrix} 1.75 & 2 \\ 1.75 & 2.25 \end{bmatrix}$$

برای آپدیت کردن مقادیر فیلتر از فرمول زیر استفاده می‌کنیم:

$$F = F - \alpha \cdot \frac{\partial l}{\partial F}$$

$\frac{\partial l}{\partial x}$ ممکن است برای محاسبه گرادیان های لایه قبلی مورد نیاز باشد

$$\frac{\partial l}{\partial x_{11}} = \frac{\partial l}{\partial o_{11}} \times F_{11} = \frac{1}{4}(-1) = \frac{-1}{4} = -0.25$$

$$\frac{\partial l}{\partial x_{12}} = \frac{\partial l}{\partial o_{11}} \cdot F_{12} + \frac{\partial l}{\partial o_{12}} \cdot F_{11} = \frac{1}{4} \times 3 + \frac{1}{4} \times (-1) = 0.5$$

$$\frac{\partial l}{\partial x_{13}} = \frac{\partial l}{\partial o_{12}} \cdot F_{12} = \frac{1}{4} \times 3 = 0.75$$

$$\frac{\partial l}{\partial x_{21}} = \frac{\partial l}{\partial o_{11}} \cdot F_{21} + \frac{\partial l}{\partial o_{21}} \cdot F_{11} = \frac{1}{4}(-2) + \frac{1}{4}(-1) = -0.75$$

$$\frac{\partial l}{\partial x_{22}} = \frac{\partial l}{\partial o_{11}} \cdot F_{22} + \frac{\partial l}{\partial o_{12}} F_{21} + \frac{\partial l}{\partial o_{21}} F_{12} + \frac{\partial l}{\partial o_{22}} F_{11}$$

$$= \frac{1}{4} \times 0 + \frac{1}{4}(-2) + \frac{1}{4}(3) + \frac{1}{4}(-1) = 0$$

$$\frac{\partial l}{\partial x_{23}} = \frac{\partial l}{\partial o_{12}} F_{22} + \frac{\partial l}{\partial o_{22}} \cdot F_{12} = \frac{1}{4} \times 0 + \frac{1}{4} \times 3 = 0.75$$

$$\frac{\partial l}{\partial x_{31}} = \frac{\partial l}{\partial o_{21}} F_{21} = \frac{1}{4} \times (-2) = -0.5$$

$$\frac{\partial l}{\partial x_{32}} = \frac{\partial l}{\partial o_{21}} F_{22} + \frac{\partial l}{\partial o_{22}} F_{21} = \frac{1}{4} \times 0 + \frac{1}{4} \times (-2) = -0.5$$

$$\frac{\partial l}{\partial x_{33}} = \frac{\partial l}{\partial o_{22}} \cdot F_{22} = \frac{1}{4} \times 0 = 0$$

$$\frac{\partial l}{\partial x} = \begin{bmatrix} -0.25 & 0.5 & 0.75 \\ -0.75 & 0 & 0.75 \\ -0.5 & -0.5 & 0 \end{bmatrix}$$

5. توضیح بخش های مختلف نوت بوک:

بخش ConvNet:

در ابتدا دیتاست MNIST را دانلود می کنیم و با استفاده از transform ورودی ها را نرمالیزه می کنیم. از Data Loader برای تبدیل داده ها به batchهای گوناگون استفاده می کنیم:

```
train_loader = torch.utils.data.DataLoader(train_set, batch_size=64, shuffle=True)  # Your code goes here.
test_loader = torch.utils.data.DataLoader(test_set, batch_size=64, shuffle=True)  # Your code goes here.
```

سپس یک مدل CNN تعریف می کنیم. لایه اول آن یک لایه کانولوشنی با ۶۴ فیلتر با سایز ۵ است. از padding نیز استفاده می کنیم تا اندازه تصاویر کوچک نشود. سپس از تابع فعال سازی relu و پس از آن Max pooling استفاده می کنیم. به جهت اینکه مدل overfit نشود از لایه dropout نیز استفاده می کنیم. بعد لایه کانولوشنی دوم دارای ۱۲۸ فیلتر با سایز ۵، تابع فعال سازی relu، ادغام حداکثر و dropout را اعمال می کنیم. سپس خروجی را flat می کنیم (به جز بعدی که مربوط به batch می شود) و از لایه خطی عبور می دهیم. در آخر از تابع فعال سازی Log Softmax استفاده می کنیم.

```
input_size = 28 * 28
classes_count = 10
```

```
class CNN(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(in_channels=1, out_channels=64, kernel_size=5, padding=2)  # output shape (28 * 28 * 64)
        self.dropout = nn.Dropout(0.1)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(in_channels=64, out_channels=128, kernel_size=5, padding=2) # output shape (14 * 14 * 128)
        self.fc = nn.Linear(7*7*128, classes_count)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.dropout(x)
        x = self.pool(F.relu(self.conv2(x)))
        x = self.dropout(x)
        x = torch.flatten(x, 1) # flatten all dimensions except batch
        x = self.fc(x)
        x = F.log_softmax(x)

        return x


net = CNN()
net.to(device)
```

تابع train را پیاده سازی می کنیم. ابتدا مدل را در حالت آموزش قرار می دهیم و یک batch از داده می خوانیم. در forward پیش بینی مدل را محاسبه می کنیم و سپس مقدار خطا را بدست می آوریم. در

back propagation پارامترهای مدل را آپدیت می کنیم. میزان دقت مدل را بدست می آوریم و به همراه خطا، آنها را در یک لیست ذخیره می کنیم.

```python
from tqdm import tqdm

def accuracy(y_pred, y):
    """
    Calculating the accuracy of given batch.
    input(s):
    y_pred (ndarray): predicted logits or the output of the model.
    y (ndarray): labels
    """
    pred_idx = y_pred.max(1, keepdim=True)[1]
    correct = pred_idx.eq(y.view_as(pred_idx)).sum().item()
    return correct / len(y)

def train(trainloader, model, optimizer, criterion, n_epochs):
    history = dict()
    history['train_loss']= list()
    history['train_acc']= list()

    for epoch in range(n_epochs):
        running_loss = 0
        running_acc = 0

        model.train()
        for idx, (X, y) in enumerate(tqdm(trainloader)):
            optimizer.zero_grad()
            X, y = X.to(device), y.to(device)
            y_hat = model(X)
            loss = criterion(y_hat, y)
            loss.backward()
            optimizer.step()

            running_loss += loss.item()
            running_acc += accuracy(y_hat, y)
        else:
            running_loss = running_loss/ len(trainloader)
            running_acc = running_acc / len(trainloader)
            history['train_loss'].append(running_loss)
            history['train_acc'].append(running_acc)
            print(f'Epoch {epoch+1}/{n_epochs} : training loss: {running_loss}, training acc: {running_acc}')

    return history
```

مدل را با بهینه ساز Adam و تابع ضرر NLL به مدت ۱۰ ایپاک آموزش می دهیم. نتیجه آموزش به صورت زیر است:

```python
train(train_loader, net, optimizer, criterion, 10)
```

```
  0%|          | 0/938 [00:00<?, ?it/s]<ipython-input-15-d4995c788583>:31: UserWarning: Implicit dimension choice for log_softmax has
  x = F.log_softmax(x)
100%|██████████| 938/938 [00:27<00:00, 34.68it/s]
Epoch 1/10 : training loss: 0.11105363050188774, training acc: 0.9659681503198294
100%|██████████| 938/938 [00:20<00:00, 45.39it/s]
Epoch 2/10 : training loss: 0.040944081685157806, training acc: 0.9871901652452025
100%|██████████| 938/938 [00:21<00:00, 44.11it/s]
Epoch 3/10 : training loss: 0.029692124512995646, training acc: 0.990238539445629
100%|██████████| 938/938 [00:20<00:00, 45.04it/s]
Epoch 4/10 : training loss: 0.022234017432603614, training acc: 0.9931369936034116
100%|██████████| 938/938 [00:20<00:00, 44.80it/s]
Epoch 5/10 : training loss: 0.017411392109604264, training acc: 0.994103144989339
100%|██████████| 938/938 [00:21<00:00, 42.95it/s]
Epoch 6/10 : training loss: 0.01619337226517311, training acc: 0.9945029317697228
100%|██████████| 938/938 [00:21<00:00, 44.49it/s]
Epoch 7/10 : training loss: 0.014255861780601486, training acc: 0.9953524786780383
100%|██████████| 938/938 [00:20<00:00, 45.67it/s]
Epoch 8/10 : training loss: 0.011678307069773434, training acc: 0.996202025586354
100%|██████████| 938/938 [00:21<00:00, 44.37it/s]
Epoch 9/10 : training loss: 0.011096782769114097, training acc: 0.9961853678038379
100%|██████████| 938/938 [00:21<00:00, 44.04it/s]Epoch 10/10 : training loss: 0.010762689704556854, training acc: 0.9964019189765458
```

سپس از مدل آموزش دیده برای پیش بینی دیتاست آزمون استفاده می کنیم. خروجی به صورت زیر است:

```
def test(testloader, model):
    # Test the model
    model.eval()

    test_acc = 0
    with torch.no_grad():

        for idx, (X, y) in enumerate(tqdm(testloader)):
            X, y = X.to(device), y.to(device)
            y_hat = model(X)
            test_acc = accuracy(y_hat, y)

    print('Test Accuracy of the model', test_acc)
```

```
test(test_loader, net)
```

```
  0%|          | 0/157 [00:00<?, ?it/s]<ipython-input-15-d4995c788583>:31: UserW
  x = F.log_softmax(x)
100%|██████████| 157/157 [00:02<00:00, 60.56it/s]Test Accuracy of the model 1.0
```

این مدل تنها با دو لایه کانولوشنی توانسته به دقت ۱۰۰٪ روی داده آزمون برسد.

بخش Interpretation:

مدل resnet را با وزن های خواسته شده لود می کنیم و یک لیست از اسامی کلاس هایی که مدل روی آن داده آموزش دیده تشکیل می دهیم.

```
from torchvision.models import resnet50, ResNet50_Weights

weights = ResNet50_Weights.IMAGENET1K_V2
resnet = resnet50(weights=weights)  # Your code goes here.
class_names = weights.meta["categories"]  # Your code goes here.
```

سپس برای تست کردن، یک تصویر را لود می کنیم، آن را پیش پردازش می کنیم و به مدل می دهیم. ۳ خروجی برتر مدل را چاپ می کنیم:

```
sample = preprocess(im)
batch = sample.unsqueeze(0)
logits = resnet(batch)
logits = logits.squeeze()
```

will get a 1000-D vector known as *logits*. These are th
efore, are not probability values in the $[0, 1]$ range. Y
vector in order to find the categories with the highes

```
values, indices = torch.topk(logits, 3)
indices
```

```
tensor([264, 263, 227])
```

can translate the class IDs using the dictionary we lo

```
proba = torch.softmax(logits, 0)
for index in indices:
    print(class_names[index], f'Likelihc
```

```
Cardigan Likelihood: 0.27% (0.9956)
Pembroke Likelihood: 0.26% (0.9551)
kelpie Likelihood: 0.18% (0.6140)
```

بخش Kernel Visualization:

وزن های لایه conv1 مدل را در یک متغیر ذخیره می کنیم. سپس برای اینکه بهتر قابل مشاهده باشد با استفاده از interpolate آنها را up scale می کنیم:

```
kernels = torch.clone(resnet.conv1.weight).detach() # Your code goes here.
```

```
kernels.shape
```

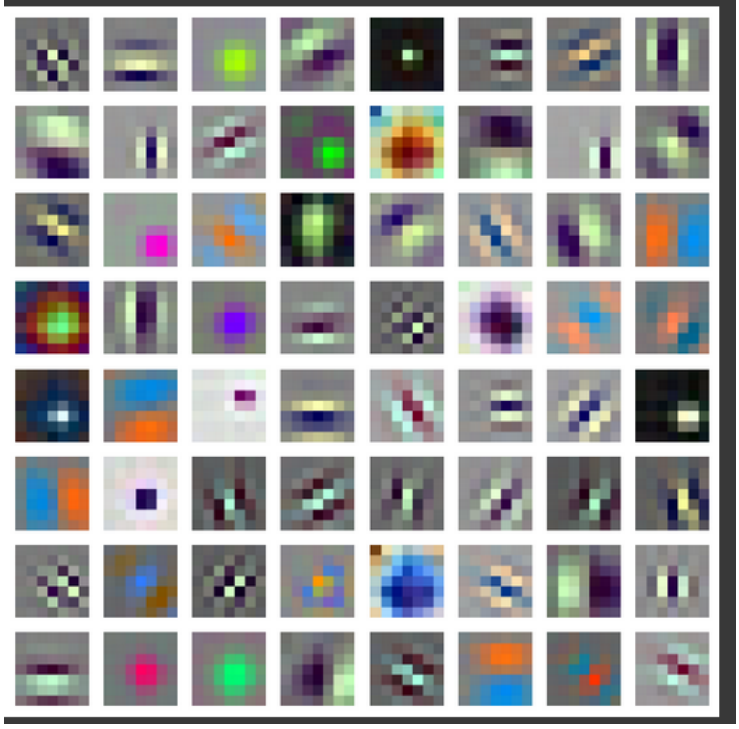
```
torch.Size([64, 3, 7, 7])
```

hould upscale the maps for better visualization:

```
kernels = F.interpolate(kernels, size=64)
```

و بعد در یک حلقه آنها را نمایش می دهیم. با استفاده از این تصاویر می توانیم پی ببریم که هر فیلتر چه چیزی را آموزش دیده است.

```python
figure, axes = plt.subplots(8, 8, figsize=(8, 8))
for i in range(8):
    for j in range(8):
        index = i * 8 + j
        kernel = kernels[index]
        kernel = kernel - kernel.min()
        kernel = kernel / kernel.max()
        kernel = kernel.permute(1, 2, 0).numpy()
        axes[i, j].imshow(kernel)
        axes[i, j].axis('off')
plt.show()
```



بخش Feature Maps:

با استفاده از IntermediateLayerGetter خروجی ۴ لایه نخست را روی داده پیش بینی می کنیم:

```python
feature_extractor = IntermediateLayerGetter(resnet,
                                            {'layer1': 'layer1',
                                             'layer2': 'layer2',
                                             'layer3': 'layer3',
                                             'layer4': 'layer4'})  # Your code goes here.
```

now pass our image through the model, we can expect a `dict` containing the feature maps:

```python
di = feature_extractor(batch)
for key, tensor in di.items():
    print(key, tensor.shape)
```

```
layer1 torch.Size([1, 256, 90, 125])
layer2 torch.Size([1, 512, 45, 63])
layer3 torch.Size([1, 1024, 23, 32])
layer4 torch.Size([1, 2048, 12, 16])
```

وزن ها و بایاس لایه کاملا متصل را در یک متغیر ذخیره می کنیم. برای محاسبه نقشه ویژگی از فرمول W.X + b استفاده می کنیم.

```
weights = torch.clone(resnet.fc.weight).detach()  # Your code goes here.
biases = torch.clone(resnet.fc.bias).detach()  # Your code goes here.
```

denote the weights and biases of `fc` as W and b, and the `layer4` feature map as X, we s

```
X = di['layer4'].squeeze()  # To drop the batch dim.
W = weights.clone()
b = biases.clone()
```

```
W.shape, X.shape, b.shape
```

```
(torch.Size([1000, 2048]), torch.Size([2048, 12, 16]), torch.Size([1000]))
```

can collapse the spatial dimensions of F and reshape it into a 2048×49 matrix, so that yo
natively, you can use `torch.einsum`. The choice is yours.

```
X = torch.reshape(X, (2048, -1))
print("new shape of X", X.shape)

R = (torch.matmul(W, X).T + b).T  # Your code goes here.
print("shape of R", R.shape)
R = torch.reshape(R, (1000, 1, 12, 16))
```

```
new shape of X torch.Size([2048, 192])
shape of R torch.Size([1000, 192])
```

```
R.shape
```

```
torch.Size([1000, 1, 12, 16])
```

سپس آن را up sample می کنیم تا بتوانیم بر روی تصویر اولیه قرار دهیم:

```
shape = im.size[::-1]
print(shape)
```

```
(360, 500)
```

```
R = F.interpolate(R, (shape[0], shape[1]))  # Your code goes here.
```

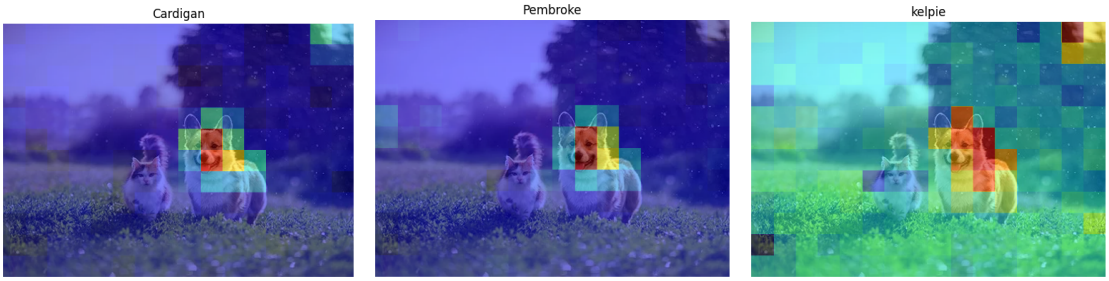
```
R.shape
```

```
torch.Size([1000, 1, 360, 500])
```

that PyTorch has automatically recorded the operations to be able to calculate the

```
R = R.detach()
```

حاصل را برای ۳ کلاس با بیشترین احتمال به تصویر می کشیم:



می توانیم این نقشه های ویژگی را برای سایر انواع کلاس های گربه رسم کنیم. به طور مثال برای گربه tabby:

بخش masking:

در یک حلقه یک مستطیل ۳۲ * ۳۲ مشکی رنگ می سازیم و آن را روی بخش های مختلف تصویر قرار می دهیم. تصویر بدست آمده را پیش پردازش کرده و خروجی مدل را برای آن پیش بینی می کنیم. با استفاده از logit و لایه فعال سازی softmax میزان احتمال را پیش بینی می کنیم. آن را در تنسور masked_probs ذخیره می کنیم.

```
i = int(np.ceil((width+1)/patch))
j = int(np.ceil((height+1)/patch))

masked_probs = np.zeros(shape=(i, j))
print(masked_probs.shape)
```

```
16, 12)
```

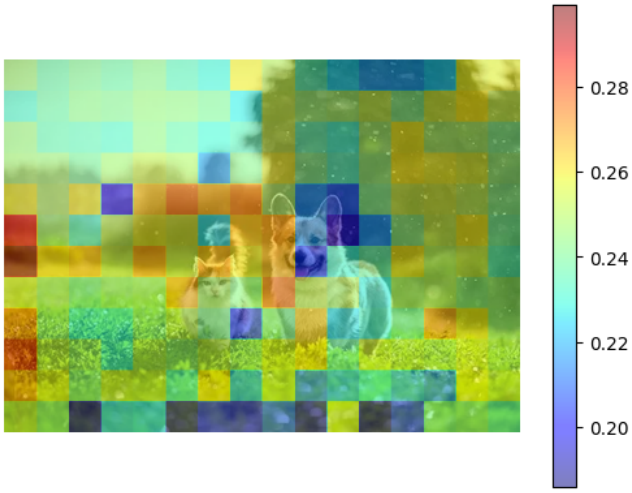
```
# Your code goes here.

activation = torch.nn.Softmax(dim=-1)

for w in range(0, width+1, patch):
    for h in range(0, height+1, patch):

        masked_img = im.copy()
        draw = ImageDraw.Draw(masked_img)
        draw.rectangle(
            (w, h, w+patch, h+patch),
            fill='black',
            outline='black')

        sample = preprocess(masked_img)
        batch = sample.unsqueeze(0)
        logits = resnet(batch).detach()
        prob = activation(logits)
        masked_probs[w//patch, h//patch] = prob[0, class_id]
```

سپس خروجی را به شکل خواسته شده تبدیل می کنیم و آن را up scale می کنیم تا بتوانیم آن را روی تصویر نمایش دهیم.

```
tensor = torch.from_numpy(masked_probs.T)
tensor = tensor.view(1, 1, 12, 16)
tensor.shape
```

```
torch.Size([1, 1, 12, 16])
```

we can use `F.interpolate`:

```
tensor = F.interpolate(tensor, (height, width))  # Your code goes here.
```

حاصل: