یادگیری عمیق

تمرین چهارم

استاد درس: دکتر محمدی

مهسا موفق بهروزی

بهار ۱۴۰۲

سوال یک)

الف) در این مقاله، کپسول‌ها نوعی عنصر معرفی شده‌اند که برای نشان دادن ویژگی‌های یک موجودیت استفاده می‌شوند. کپسول‌ها را می‌توان به‌عنوان جایگزینی برای نورون‌های سنتی در نظر گرفت، به طوری که به‌جای اینکه خروجی صرفا یک مقدار scalar باشد، یک بردار خواهد بود که جنبه‌های مختلف یک موجودیت یا ویژگی، مانند حالت، تغییر شکل و بافت و ... را نشان می‌دهد. طول این بردار برای نشان دادن احتمال وجود موجودیت استفاده می‌شود و جهت آن نشان دهنده پارامترهای آن است.

کپسول‌ها از مکانیسم‌های مسیریابی پویا برای ارسال خروجی‌های خود به کپسول والد مناسب در لایه‌ی بالا استفاده می‌کنند. کپسول‌های سطح پایین خروجی خود را به کپسول‌های سطح بالاتری ارسال می‌کنند که بردارهای آن‌ها scalar product بالایی با استفاده از پیش‌بینی دریافتی از کپسول‌های لایه‌ی پایین تولید می‌کنند.

ب) در شبکه‌های CNN، هر نورون بر اساس تشخیص یک الگوی خاص، فعال می‌شود و به property های یک feature مانند اندازه، سرعت، رنگ و .... توجه نمی‌کند و بر اساس روابط بین feature ها آموزش داده نمی‌شوند. این مساله robustness مدل را کاهش می‌دهد. مانند مثال ارائه شده در لینک، که یک rotation در عکس پاندا باعث شده است مدل پیش بینی نادرست داشته باشد. در صورتی‌که اگر بر اساس روابط بین فیچرها مانند تعیین رابطه بین بینی و چشم آموزش دیده بود، ممکن بود این مساله پیش نیاید و نیاز به دیتای فراوان برای robust کردن مدل نداشته باشیم. تعیین رابطه بین بینی و چشم نیازمند داشتن مکان دقیق این ویژگی‌ها در تصویر ورودی است. لایه‌ی maxpooling در cnn ها باعث می‌شود این اطلاعات مکانی از بین برود.

شبکه‌های کپسولی برای حل این مشکل مطرح شده‌اند.

ج)

بازنمایی: در CNN ها، هر لایه از مجموعه‌ای از feature map تشکیل شده است که ویژگی‌های مختلف تصویر ورودی را encode می‌کند. در مقابل، CapsNets از کپسول‌ها استفاده می‌کند، که گروه‌هایی از نورون‌ها هستند که بخش‌های مختلف شی ورودی و وضعیت آن (جهت، اندازه، موقعیت ...) را نشان می‌دهند.

ساختار سلسله مراتبی: CapsNet ها به صورت سلسله مراتبی سازماندهی می‌شوند که کپسول‌های سطح بالاتر نشان دهنده ویژگی‌های پیچیده‌تری هستند که از کپسول‌های سطح پایین‌تر تشکیل شده‌اند. این به CapsNets اجازه می‌دهد تا ویژگی‌هایی را بیاموزد که نسبت به تغییر حالت و تغییر شکل شی ورودی، robustتر باشد.

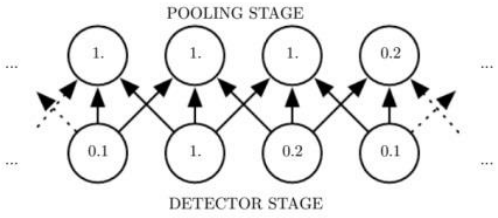
مسیریابی پویا: CapsNets از مسیریابی پویا بین کپسول‌ها برای محاسبه خروجی هر کپسول استفاده می‌کند.

تابع هزینه: CapsNets از یک تابع ضرر متفاوت (Margin Loss) در مقایسه با CNN ها استفاده می‌کند.

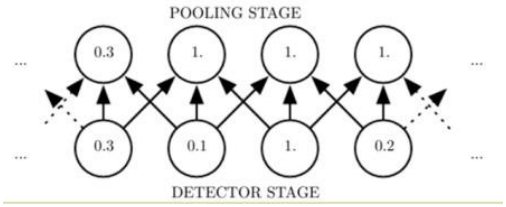
سوال دو)

الف) یکی از راه‌های جلوگیری از overfitting، ساده کردن مدل یا به عبارتی کاهش پارامترهای مدل است. این کار با استفاده از stride (پرش) در لایه‌های کانولوشنی امکان‌پذیر است. پرش باعث می‌شود تا اطلاعاتی را دور بریزیم. یک راه برای جلوگیری از دور ریختن کامل اطلاعات استفاده از لایه‌های pooling می‌باشد. تابع ادغام، خروجی شبکه را در یک موقعیت مشخص گرفته و آن را با یک مشخصه آماری از همسایگی‌اش جایگزین می‌کند. به عنوان مثال لایه ادغام مانگین، مقدار میانگین در یک محدوده‌ی مشخص و ادغام حداکثر مقدار بیشینه در یک محدوده‌ی مشخص را محاسبه می‌کند. با این کار، استفاده از پرش اطلاعات کمتری را حذف می‌کند.

همچنین استفاده از لایه ادغام، کمک می‌کند که بازنمایی نسبت به جابجایی‌های کوچک حساسیت کمتری داشته باشد. مانند شکل زیر از اسلایدها:



الف) استفاده از لایه ادغام حداکثر



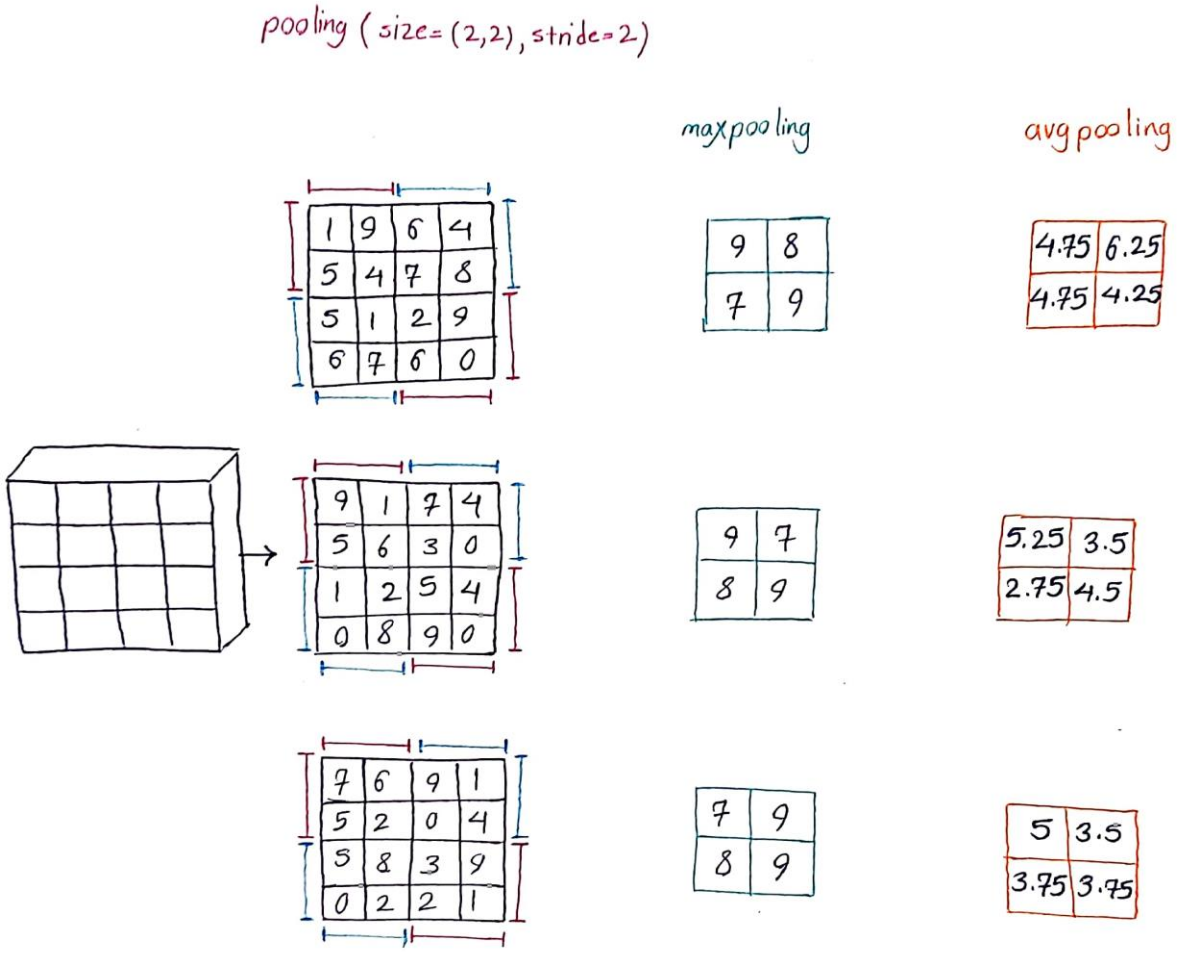
ب) استفاده از لایه ادغام حداکثر بعد از یک شیفت در ورودی

شکل ب نشان می‌دهد که حتی بعد از یک شیفت در ورودی، خروجی لایه‌ی ادغام دچار تغییرات زیادی نشده است و ۳ مقدار ۱ که قبل از شیفت نیز در خروجی ادغام مشاهده می‌شد، پس از شیفت نیز مشاهده می‌شود.

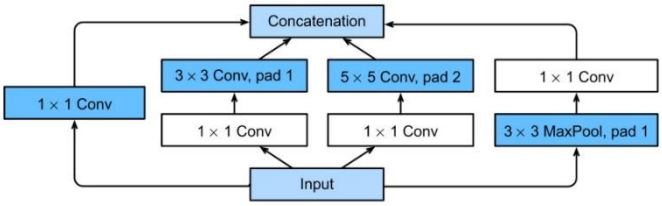
یکی دیگر از مواردی که استفاده از padding ضروری‌ست، برای زمانی‌ست که قصد داریم خروجی‌ها را با هم concatenate کنیم. مشابه کاری که در googlNet انجام می‌شود. در اینصورت لازم است تا ابعاد همه خروجی‌هایی که با اعمال کرنل‌هایی با سایز مختلف به دست آمده، یکسان باشد و نیاز به padding داریم.

لایه ادغام حداکثر، در هر window، بیشترین مقدار را نگه می‌دارد. به عبارتی مهم‌ترین ویژگی را حفظ می‌کند. اما لایه ادغام میانگین، در هر window، میانگین گرفته یا به عبارتی ساختار کلی ورودی را حفظ می‌کند. بنابراین می‌توان گفت که از ادغام حداکثر برای شناسایی ویژگی‌ها یا الگوهای خاص (مثلا شناسایی لبه) استفاده می‌شود و از ادغام میانگین برای کارهایی که نیاز به درک کلی‌تری از میانگین دارند.

ب)



ج) در شبکه googleNet برای کاهش بعد از فیلترهای ۱ * ۱ استفاده شده است. مانند شکل زیر از اسلایدها، که قبل از اعمال فیلتر ۵ * ۵، فیلتر ۱ * ۱ استفاده کرده و تعداد ویژگی‌ها را کاهش می‌دهیم و سپس ۵ * ۵ را اعمال می‌کنیم. این کار باعث می‌شود تعداد پارامترهای این لایه کاهش پیدا کند.



همچنین از Global Avg Pooling برای کاهش پارامترها استفاده می‌کند. به طوریکه بعد از ۵ مرحله pooling، یک pooling سراسری با همان ابعاد feature map اعمال می‌کند. به طور مثال اگر تصویر ۲۲۴ * ۲۲۴ پس از ۵ لایه pooling به ابعاد ۷ * ۷ رسیده است، در Global Avg Pooling، یک pooling ۷ * ۷ اعمال شده و یک ویژگی اسخراج می‌شود و ۴۹ برابر ابعاد را کاهش می‌دهد.

معماری NiN به عنوان جایگزینی برای شبکه‌های عصبی کانولوشنی سنتی مانند LeNet، AlexNet و VGG پیشنهاد شد که به شدت بر لایه‌های fully connected متکی هستند. و این مساله باعث افزایش بسیار زیاد تعداد پارامترها می‌شود. NiN از فیلترهای ۱ * ۱ برای اضافه کردن غیرخطی بودن محلی و همچنین از ایده‌ی GAP که در googleNet توضیح داده شد استفاده می‌کند که باعث می‌شود تعداد پارامترها به شدت کاهش پیدا کند.
بلوک‌های NiN از تعدادی کانولوشن اولیه و سپس کانولوشن‌های ۱×۱ و یک لایه max pooling تشکیل شده‌اند. همچنین NiN به طور کلی از لایه‌های FC اجنتاب می‌کند و در عوض، از یک بلوک NiN با تعدادی کانال خروجی برابر با تعداد کلاس‌های label و به دنبال آن یک GAP استفاده می‌کند. این مساله باعث کاهش بسیار زیاد تعداد پارامترها می‌شود.

در مورد تعداد پارامترهای شبکه‌های مختلف می‌توان گفت شبکه‌ی VGG 19 در حدود ۱۳۸ میلیون پارامتر، AlexNet با ۸ لایه (۵ لایه کانولوشنی و ۳ لایه کاملا متصل) حدودا ۶۱ میلیون پارامتر و GoogleNet با ۲۲ لایه، ۷ میلیون پارامتر و NiN با ۸ لایه کانولوشنی و ۱ لایه GAP، فقط در حدود یک میلیون پارامتر دارد.

سوال سه)

Conv Layer output size:

$$W_2 = \frac{W_1 - FilterSize + 2 * Padding}{Stride} + 1$$

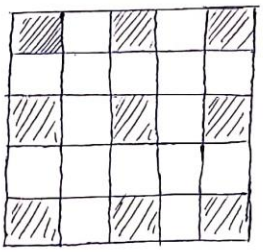
$$H_2 = \frac{H_1 - FilterSize + 2 * Padding}{Stride} + 1$$

$$D_2 = number\ of\ filters$$

$$\longrightarrow output = W_2 \times H_2 \times D_2$$

Conv Layer with dilation:

برای مثال، با یک فیلتر 3*3 با dilation = 2، میتوان به یک ناحیه 5x5 نگاه کرد.



بخش‌های سفید داخل وزن صفر است.

برای محاسبه‌ی خروجی میتوان kernel-size را برابر با

dilation x (kernel-size -1) +1

در نظر گرفت.

برای لایه 2، خروجی شبیه با لایه کانولوشنی با فیلتر 9x9 در نظر گرفت.
برای لایه 5، خروجی شبیه با لایه کانولوشنی با فیلتر 17x17 در نظر گرفت.
برای لایه 8، شبیه لایه 2 است.

Max pooling Layer size:

Conv مشابه $W_2 H_2$

$$D_2 = D_1$$

Layer1 : Conv (64, (3,3), stride =1, padding = 'same')

درصورتیکه stride=1 باشد، به میزان کافی padding میزنیم تا سایز ورودی و خروجی برابر شود.

output : $256 \times 256 \times 64$

num_params : $64 \times (3 \times 3 \times 3 + 1)$

Layer2 : Dilated-Conv (32, (5,5), stride = 2, dilation rate=2, padding = 'valid')

⟵ فیلتر 9x9   padding نمیزنیم.

$$H, W = \left\lfloor \frac{256 - 9 + 2 \times 0}{2} + 1 \right\rfloor = 124$$

output : $124 \times 124 \times 32$

num_params : $32 \times (5 \times 5 \times 64 + 1)$

Layer 3: Max-pool (size=(2,2), stride=2)

$$H, W = \frac{124 - 0 + 2 \times 0}{2} + 1 = 63$$

output : $63 \times 63 \times 32$, num_params = 0

Layer 4 : Conv (128, (3,3), stride = 1, padding = 'same')

output : $63 \times 63 \times 128$

num_params : $128 \times (3 \times 3 \times 32 + 1)$

Layer 5 : Dilated-Conv (64, (5,5), stride = 2, dilation rate = 4, padding = 'valid')

$$H, W = \left\lfloor \frac{63 - 17 + 2 \times 0}{2} + 1 \right\rfloor = 24$$

output: $24 \times 24 \times 64$

num_params : $64 \times (5 \times 5 \times 128 + 1)$

Layer 6 : Max -pool (size = (2,2), stride = 2)

$$H, W = \frac{24}{2} + 1 = 13$$

output = 13 x 13 x 64

num_params = 0

Layer 7 : Conv (256, (3,3), stride = 1, padding = 'same')

output : 13 x 13 x 256

num_params = 256 x (3 x 3 x 64 + 1)

Layer 8: Dilated - Conv (128, (5,5), stride = 2, dilation rate = 2, padding: 'valid')

$$H, W = \frac{13 - 9 + 2\times 0}{2} + 1 = 3$$

output : 3 x 3 x 128

num_params = 128 x (5 x 5 x 256 + 1)

Layer 9: Max-Pool(size = (2,2), stride = 2)

$$H, W = \left\lfloor \frac{3}{2} + 1 \right\rfloor = 2$$

output : 2 x 2 x 128

num - params = 0

سوال چهار)

$$X * F = O$$

$$y = GAP(O)$$

$$L = L(y, \hat{y}) \;,\; \frac{\partial L}{\partial y} = 1$$

$$\frac{\partial L}{\partial O_{ij}} = \frac{\partial L}{\partial y} \times \frac{\partial y}{\partial O_{ij}} \quad \Big| \quad \frac{\partial y}{\partial O_{ij}} = \frac{1}{4}$$

$$\Rightarrow \frac{\partial L}{\partial O_{ij}} = 1 \times \frac{1}{4} = \frac{1}{4}$$

$$\frac{\partial L}{\partial F} = X * \frac{\partial L}{\partial O}$$

$$\frac{\partial L}{\partial X} = 180^\circ \text{ rotated Filter F} \; (\text{zero-padded}) * \frac{\partial L}{\partial O}$$

$$\Rightarrow \frac{\partial L}{\partial F} = \begin{bmatrix} 1 & 2 & -2 \\ -1 & 5 & 3 \\ 3 & 0 & 1 \end{bmatrix} * \begin{bmatrix} \frac{1}{4} & \frac{1}{4} \\ \frac{1}{4} & \frac{1}{4} \end{bmatrix} = \begin{bmatrix} \frac{7}{4} & 2 \\ \frac{11}{4} & \frac{9}{4} \end{bmatrix}$$

$$\frac{\partial L}{\partial X} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & -2 & 0 \\ 0 & 3 & -1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} * \begin{bmatrix} \frac{1}{4} & \frac{1}{4} \\ \frac{1}{4} & \frac{1}{4} \end{bmatrix} = \begin{bmatrix} 0 & -\frac{1}{2} & -\frac{1}{2} \\ \frac{3}{4} & 0 & -\frac{3}{4} \\ \frac{3}{4} & \frac{1}{2} & -\frac{1}{4} \end{bmatrix}$$

سوال پنج)

بخش ابتدایی نوت‌بوک، در بخش دانلود دیتاست، نیاز به تغییرات داشت:

```
from torchvision.transforms import ToTensor

train_set = datasets.MNIST(root='./data', train=True, download=True, transform = ToTensor())
test_set = datasets.MNIST(root='./data', train=False, download=True, transform = ToTensor())
```

بخش train_loader و test_loader به شکل زیر با batch_size = 128 و True شدن قابلت shuffle برای train_loader نوشته شده است:

```
train_loader = torch.utils.data.DataLoader(train_set,
                                           batch_size=128,
                                           shuffle=True
                                           )  # Your code goes here.

test_loader = torch.utils.data.DataLoader(test_set,
                                           batch_size=128
                                           )  # Your code goes here.
```

باتوجه به نوت‌بوک، معماری شبکه با استفاده از دو لایه کانولوشنی طراحی شده است که دقیقا مشابه شبکه‌ی LeNet 5 می‌باشد. باتوجه به اینکه تصاویر دیتاست grayscale می‌باشند، تعداد input channel های لایه اول ۱ قرار داده شده و ۶ فیلتر با ابعاد ۵ * ۵ بر روی آن‌ها اعمال خواهد شد. همچنین در لایه اول padding = 2 قرار داده شده است که سایز تصاویر را به ۳۲ * ۳۲ تغییر می‌دهد. (ورودی شبکه LeNet یک تصویر ۲۸ * ۲۸ مانند تصاویر MNIST می‌باشد که در همان لایه اول zero padding انجام داده و ابعاد ورودی را به ۳۲ * ۳۲ تغییر می‌دهد).

سپس تابع فعال‌سازی ReLU اعمال شده و یک لایه‌ی pooling از جنس میانگین گرفتن (avgpooling) قرار داده شده که با stride = 2، یکی درمیان، از هر دو پیکسل میانگین می‌گیرد. این downsampling خروجی را به ۱۴ * ۱۴ تغییر می‌دهد.

لایه‌ی کانولوشنی بعدی با ۶ input channel، که معادل با تعداد فیلترها/خروجی‌های لایه‌ی قبل است و ۱۶ فیلتر با ابعاد ۵ * ۵ طراحی شده است. تابع فعال‌سازی و pooling مشابه قبلی‌ست.

مشابه LeNet 5، ۳ لایه‌ی fully connected در انتهای شبکه تعبیه شده. ابتدا خروجی لایه‌ی کانولوشنی دوم را flat کرده و به لایه‌ی FC اول با ۱۲۰ خروجی می‌دهیم. لایه‌ی FC دوم ۸۴ خروجی و لایه‌ی FC سوم ۱۰ خروجی (معادل با تعداد کلاس‌های MNIST) خواهد داشت.

```python
from torchsummary import summary

net = nn.Sequential(

        nn.Conv2d(1, 6, 5, padding=2),
        nn.ReLU(),
        nn.AvgPool2d(2, stride=2),

        nn.Conv2d(6, 16, 5),
        nn.ReLU(),
        nn.AvgPool2d(2, stride=2),

        nn.Flatten(),

        nn.Linear(400, 120),
        nn.ReLU(),

        nn.Linear(120, 84),
        nn.ReLU(),

        nn.Linear(84, 10),
        nn.Softmax()
)

summary(net, (1, 28, 28), device = 'cpu')
```

در انتها نیز خروجی‌های هر لایه و تعداد پارامترها نمایش داده شده است.

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1           [-1, 6, 28, 28]             156
              ReLU-2           [-1, 6, 28, 28]               0
         AvgPool2d-3           [-1, 6, 14, 14]               0
            Conv2d-4          [-1, 16, 10, 10]           2,416
              ReLU-5          [-1, 16, 10, 10]               0
         AvgPool2d-6            [-1, 16, 5, 5]               0
           Flatten-7                 [-1, 400]               0
            Linear-8                 [-1, 120]          48,120
              ReLU-9                 [-1, 120]               0
           Linear-10                  [-1, 84]          10,164
             ReLU-11                  [-1, 84]               0
           Linear-12                  [-1, 10]             850
          Softmax-13                  [-1, 10]               0
================================================================
Total params: 61,706
Trainable params: 61,706
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 0.11
Params size (MB): 0.24
Estimated Total Size (MB): 0.35
----------------------------------------------------------------
```

برای آموزش مدل، تابع train که ۴ ورودیِ dataloader، مدل، تابع هزینه و بهینه‌ساز را دریافت می‌کند به صورت زیر نوشته شده است:

مدل به فاز train برده شده و به ازای هر batch از دیتای آموزشی، پیش‌بینی مدل و مقدار loss بر اساس آن پیش‌بینی محاسبه می‌شود. سپس این مقدار loss به شبکه backpropagate می‌شود.

به ازای هر ۱۰۰ batch (هر batch برابر با ۱۲۸ بود)، گزارشی از مقدار loss چاپ می‌شود.

```python
def train(dataloader, model, loss_fn, optimizer):

    model.train();

    size = len(dataloader.dataset)

    for batch, (X, y) in enumerate(dataloader):

        pred = model(X)
        loss = loss_fn(pred, y)

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        if batch % 100 == 0:
            loss, current = loss.item(), batch * len(X)
            print(f"loss: {loss:>7f}  [{current:>5d}/{size:>5d}]")
```

برای بخش تست، تابع test با ورودی‌های data_loader و مدل به صورت زیر نوشته شده است:

مدل به فاز eval برده شده و پیش‌بینی مدل به ازای داده‌های ورودی محسابه می‌شود. سپس مقدار loss و accuracy محاسبه شده است.

```python
def test(dataloader, model):

    model.eval();

    size = len(dataloader.dataset)
    test_loss, correct = 0, 0
    with torch.no_grad():
        for X, y in dataloader:

            pred = model(X)
            # print(pred.shape)
            test_loss += criterion(pred, y).item()
            correct += (pred.argmax(1) == y).type(torch.float).sum().item()
    test_loss /= size
    correct /= size
    # print(f"Test Error: \n Accuracy: {(100*correct):>0.1f}%, Avg loss: {test_loss:>8f} \n")
    return 100*correct
```

به ازای ۵۰ ایپاک، تابع تست با پارامترهای تعریف شده فراخوانی شده و در نهایت پس از آموزش مدل، تابع تست با دیتای تست فراخوانی می‌شود. که خروجی برابر با ۹۸.۸۳ (بیش از ۹۷ درصد) دقت بر روی داده‌های تست است.

```
epochs = 50

for t in range(epochs):
    print(f"Epoch {t+1}\n-------------------------------")
    print(train(train_loader, net, criterion, optimizer))

print(test(test_loader, net))

print("Done!")
```

```
Epoch 50
-------------------------------
loss: -0.991907  [    0/60000]
loss: -0.984641  [12800/60000]
loss: -1.000000  [25600/60000]
loss: -0.992186  [38400/60000]
loss: -1.000000  [51200/60000]
None
98.83999999999999
Done!
```

برای بخش Interpretation:

ابتدا resnet 50 لود شده است. سپس اسم کلاس‌ها (کلاس‌های دیتاست ImageNet که resnet بر روی آن train شده) از فایل txt ای شامل نام آن‌ها خوانده شده است.

```
import pprint
import requests
import torchvision
from torchvision.models import resnet50, ResNet50_Weights

resnet = resnet50(weights=ResNet50_Weights.IMAGENET1K_V2)  # Your code goes here.
print(ResNet50_Weights.IMAGENET1K_V2)
class_names = requests.get("https://raw.githubusercontent.com/pytorch/hub/master/imagenet_classes.txt").text
class_names = str(class_names).split('\n')
print(class_names)
```

در بخش زیر، تصویر از گوگل درایو بارگذاری شده و سپس نمایش داده شده است.

```
from google.colab import drive
drive.mount("/content/drive")
im = Image.open('/content/drive/My Drive/DLAssignments/04/a-cat-and-a-dog.png')
```

```
Mounted at /content/drive
```

```
im
```

با توجه به نوت‌بوک، وزن‌های لایه‌ی conv1 از Resnet را به دست آورده‌ایم.

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

در بخش زیر، ۴ لایه اول تا چهارم استخراج شده است:

```
feature_extractor = IntermediateLayerGetter(resnet, {'layer1':'layer1','layer2':'layer2','layer3':'layer3','layer4':'layer4'})
```

در این بخش وزن‌ها و بایاس لایه‌ی fully connected به دست آمده:

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

در بخش زیر ابتدا X را reshape می‌کنیم تا بتوانیم در W (ماتریس وزن‌ها) ضرب کنیم.

```
X = torch.reshape(X, (2048, 12*16))
R = torch.matmul(W,X)  # Your code goes here.
R = torch.reshape(R,(1000,1,12,16))
```

در این بخش، خروجی کانولوشنی لایه آخر که در ابعاد ۱۲ * ۱۶ است را به ۳۶۰ * ۵۰۰ (سایز تصویر اصلی) resize می‌کنیم که روی تصویر اصلی قرار دهیم تا بدانیم خروجی لایه آخر چه اطلاعاتی به ما می‌دهد.

```
R = F.interpolate(R, [360,500], mode='bilinear', align_corners=True)
```

بخش آخر نوت‌بوک (Masking) با توجه به برداشت زیر نوشته شده است:

در این بخش تصویر با ابعاد ۳۶۰ * ۵۰۰ را گرفته، و مربع‌های مشکی با ابعاد ۳۲ * ۳۲ را بر روی آن می‌لغزانیم به طوری که هم‌پوشانی نداشته باشند. سپس تصاویر به دست آمده را بررسی می‌کنیم تا متوجه شویم سیاه شدن کدام بخش از تصویر تاثیر بیشتری بر خروجی دارد. یا به عبارتی مدل به کدام بخش توجه بیشتری داشته است.

مقدار rectCount همان تعداد تصاویر به دست آمده از این عمل است که برابر با ۱۹۲ است. اما به دلیل مشکل ram در colab کد با تعداد کمتری اجرا شده است.

و images یک آرایه‌ی نامپای به تعداد rectCount و تصاویر رنگی با عمق ۳ و سایز ۳۶۰ * ۵۰۰ می‌باشد.

در حلقه for، هر بار عکس خوانده می‌شود. این کار به این علت است که اگر خارج از loop تصویر لود شود، هر بار با اعمال ماسک جدید، ماسک قبلی نیز بر روی تصویر باقی می‌ماند. پس از خواندن عکس و اعمال ماسک، آن را تبدیل به آرایه نامپای می‌کنیم و در [images[index قرار می‌دهم.

زمانی که تصویر به آرایه نامپای تبدیل شده است، شکل ذخیره‌سازی به این صورت است که ابتدا طول، عرض و سپس تعداد channel ها ذخیره شده است. در حالی‌که در شبکه ما ابتدا باید channel و سپس طول و عرض باشد. به همین دلیل از moveaxis استفاده می‌کنیم تا channel را جایجا کنیم.

سپس لیست images را به تنسور تبدیل کرده و بعد در batch قرار می‌دهیم. با استفاده از feature_extractor تمام لایه‌های resnet را استخراج کرده و در di قرار می‌دهیم. و وزن‌ها و بایاس‌های لایه fc اخر را درمیاوریم. سپس X لایه آخر را squeeze می‌کنیم (dimension را یکی کم می‌کنیم تا بتوانیم از حالت تنسور خارج کرده و در ادامه ضرب کنیم.) وزن‌ها و بایاس‌ها را reshape کرده و ضرب می‌کنیم. در نهایت یک تنسور خواهیم داشت که بعد اول آن batch است (که اگر کامل اجرا می‌کردیم ۱۹۲ بود)، بعد دوم آن کلاس‌ها می‌باشد که ۱۰۰۰ کلاس داریم، بعد سوم و چهارم نیز طول و عرض خروجی لایه‌ی کانولوشنی‌ست که بعدا آن را به طول و عرض عکس resize خواهیم کرد. چون فقط یک کلاس مشخص را نیاز داریم، یک بعدمان اضافی‌ست که آن را حذف می‌کنیم. و فقط آیتم‌هایی که مقدار کلاس‌شان برابر با class_id ای که قبلا مشخص کرده‌ایم را نگه می‌داریم. در اینصورت shape ما به‌صورت: Batch 1 طول و عرض عکس تغییر می‌کند. و برای حذف ۱، R را squeeze می‌کنیم. سپس میانگین میگیریم به این‌صورت که خروجی ۱۹۲ عکس را میانگین گرفته و heatmap درآورده و بر روی عکس اصلی نمایش می‌دهیم.

```
# Your code goes here.
from PIL import Image, ImageDraw

rectCount = 5 # should be 192 but i have memory overflow
images = np.zeros(shape=(rectCount,3,360, 500))

index = 0
for x in range(0,500,32):
  for y in range(0,360,32):
      if index >= rectCount : break
      im = Image.open('/content/drive/My Drive/DLAssignments/04/a-cat-and-a-dog.png')
      draw = ImageDraw.Draw(im)
      draw.rectangle(((x, y), (x+32, y+32)), fill="black")
      im = np.array(im)
      im = np.moveaxis(im, -1, 0)
      images[index] = np.array(im)
      index = index + 1


images =  torch.tensor(images)

batch = images
di = feature_extractor(batch.float())
```

```
weights = resnet.fc.weight.detach().clone()
biases = resnet.fc.bias.detach().clone()
X = di['layer4'].squeeze()  # To drop the batch dim.
W = weights.clone()
b = biases.clone()
X = torch.reshape(X, (rectCount,2048, 12*16))
R = torch.matmul(W,X)  # Your code goes here.
R = torch.reshape(R,(rectCount,1000,1,12,16))
R = R[:,class_id:class_id+1,:,:]
R = R.squeeze()
R = torch.mean(R, 0)
logits = R.cpu().detach().numpy()
print(R.shape)
im = Image.open('/content/drive/My Drive/DLAssignments/04/a-cat-and-a-dog.png')
logits.shape
```