# به نام خدا

گزارش تمرین سری ششم درس بینایی کامپیوتر
نام مدرس: دکتر محمدرضا محمدی

فرزان رحمانی ۹۹۵۲۱۲۷۱

## سوال ۱

الف)
ناپدید شدن گرادیان و انفجار گرادیان دو مشکل رایجی هستند که می توانند هنگام آموزش شبکه های عصبی عمیق رخ دهند. مشکل ناپدید شدن گرادیان زمانی اتفاق می‌افتد که گرادیان تابع تلفات(loss) نسبت به وزن لایه‌های قبلی بسیار کوچک می‌شود و یادگیری را برای شبکه دشوار می‌کند. این به این دلیل است که شیب‌های کوچک باعث می‌شوند وزن‌ها در طول انتشار پس‌پشتی(backpropagation) تغییر بسیار کمی داشته باشند و یادگیری از داده‌ها را برای شبکه دشوار می‌کند. از سوی دیگر، مشکل انفجار گرادیان زمانی رخ می‌دهد که شیب‌ها بسیار بزرگ می‌شوند و باعث می‌شود وزن‌ها به طور چشمگیری تغییر کنند و همگرایی شبکه را دشوار کند.
در واقع در شبکه ای از n لایه پنهان، n مشتق با هم ضرب خواهند شد. اگر مشتقات بزرگ باشند، شیب به صورت تصاعدی(exponentially) افزایش می‌یابد همانطور که مدل را به عقب منتشر می‌کنیم (propagate down the model) تا زمانی که در نهایت منفجر شوند، و این همان چیزی است که ما آن را مشکل انفجار گرادیان می‌نامیم. همچنین، اگر مشتقات کوچک باشند، شیب به صورت تصاعدی(exponentially) کاهش می‌یابد، همانطور که در مدل منتشر می‌شویم( propagate down the model) تا در نهایت ناپدید شود، و این مشکل ناپدید شدن گرادیان است.

در زیر خلاصه ای از مقاله داده شده (مفاهیم و نحوه رخداد هر کدام) به علاوه چهار راهکار آمده است.

***exploding gradients:***
In the case of exploding gradients, the accumulation of large derivatives results in the model being very unstable and incapable of effective learning, The large changes in the models' weights creates a very unstable network, which at extreme values the weights become so large that is causes overflow resulting in NaN weight values of which can no longer be updated.

**How to know?**

- The model is not learning much on the training data therefore resulting in a poor loss.
- The model will have large changes in loss on each update due to the model's instability.
- The models loss will be NaN during training.

When faced with these problems, to confirm whether the problem is due to exploding gradients, there are some much more transparent signs, for instance:

- Model weights grow exponentially and become very large when training the model.
- The model weights become NaN in the training phase.
- The derivatives are constantly

***Vanishing Gradient:***

The accumulation of small gradients results in a model that is incapable of learning meaningful insights since the weights and biases of the initial layers, which tends to learn the core features from the input data (X), will not be updated effectively. In the worst-case scenario, the gradient will be 0 which in turn will stop the network will stop further training.

**How to know?**

- The model will improve very slowly during the training phase and it is also possible that training stops very early, meaning that any further training does not improve the model.
- The weights closer to the output layer of the model would witness more of a change whereas the layers that occur closer to the input layer would not change much (if at all).
- Model weights shrink exponentially and become very small when training the model.
- The model weights become 0 in the training phase.

رویکردهای زیادی برای پرداختن به انفجار گرادیان و ناپدید شدن گرادیان وجود دارد. در این بخش ۴ رویکردی که می توانید از آنها استفاده کنید فهرست شده است.

راهکار ها:

1. **Reducing the amount of Layers**
2. **Gradient Clipping (Exploding Gradients)**
3. **Weight Initialization**
4. **ResNet (RNN structure)**

به بیانی دیگر :
ناپدید شدن گرادیان و انفجار گرادیان دو مسئله رایجی هستند که می توانند در طول آموزش شبکه های عصبی عمیق رخ دهند.

ناپدید شدن گرادیان: مسئله ناپدید شدن گرادیان به پدیده ای اشاره دارد که در آن شیب های محاسبه شده در حین انتشار پس از انتشار در لایه های یک شبکه عصبی عمیق بسیار کوچک می شوند. در نتیجه، وزن‌ها و سوگیری‌های لایه‌های قبلی شبکه حداقل به‌روزرسانی را دریافت می‌کنند و شبکه نمی‌تواند به طور مؤثر یاد بگیرد. این موضوع به ویژه در شبکه های عمیق با لایه های زیاد رایج است.

انفجار گرادیان: از طرف دیگر، مسئله گرادیان انفجاری، مخالف مسئله گرادیان ناپدید است. زمانی اتفاق می‌افتد که شیب‌ها در حین انتشار پس از انتشار در لایه‌های شبکه بسیار بزرگ می‌شوند. هنگامی که گرادیان ها بیش از حد بزرگ می شوند، می تواند منجر به یادگیری ناپایدار شود و شبکه را به جای همگرایی واگرا کند.

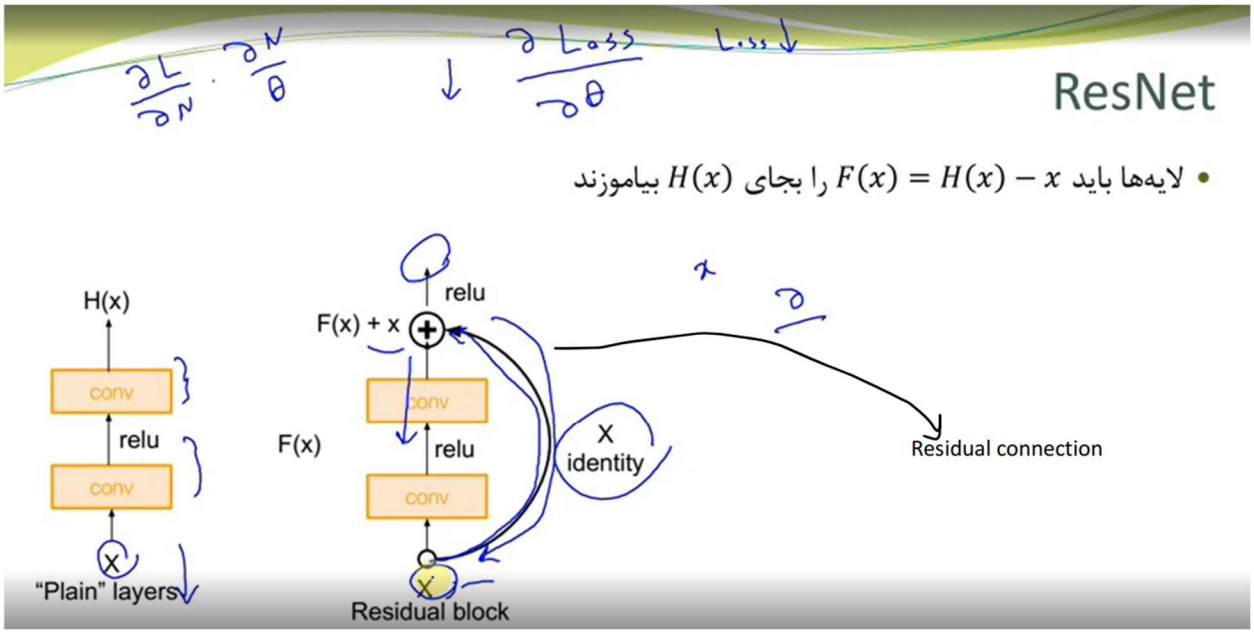
هم ناپدید شدن گرادیان و هم انفجار گرادیان می توانند به دلیل ماهیت توابع فعال سازی مورد استفاده در شبکه و همچنین عمق و معماری شبکه ایجاد شوند.

در مورد ناپدید شدن گرادیان ، یکی از عوامل اصلی کمک کننده تابع فعال سازی سیگموئید است. تابع sigmoid ورودی خود را در محدوده ای بین ۰ و ۱ فشرده می کند، به این معنی که مشتق آن (که در پس انتشار استفاده می شود) زمانی که ورودی از ۰ دور باشد کوچک می شود. همانطور که گرادیان ها به سمت عقب در شبکه منتشر می شوند، گرادیان های هر لایه می توانند تبدیل شوند. به تدریج کوچک‌تر می‌شود و به‌روزرسانی موثر وزن‌های لایه‌های قبلی را دشوار می‌کند.

از سوی دیگر، انفجار گرادیان زمانی رخ می دهد که وزن ها در شبکه با مقادیر بزرگ مقداردهی اولیه شوند یا زمانی که معماری شبکه اجازه به روز رسانی وزن زیادی را در طول انتشار پس زمینه می دهد. این می‌تواند منجر به شیب‌های بسیار بزرگ شود که باعث به‌روزرسانی قابل توجه وزن‌ها می‌شود که به طور بالقوه منجر به بی‌ثباتی در فرآیند یادگیری می‌شود.

ب) منابع:

https://towardsdatascience.com/the-vanishing-gradient-problem-69bf08b15484
https://towardsdatascience.com/introduction-to-resnets-c0a830a288a4



Residual networks are another solution, as they provide residual connections straight to earlier layers. As seen in Image 2, the residual connection directly adds the value at the beginning of the block, **x**, to the end of the block (F(x)+x). This residual connection doesn't go through activation functions that "squashes" the derivatives, resulting in a higher overall derivative of the block.

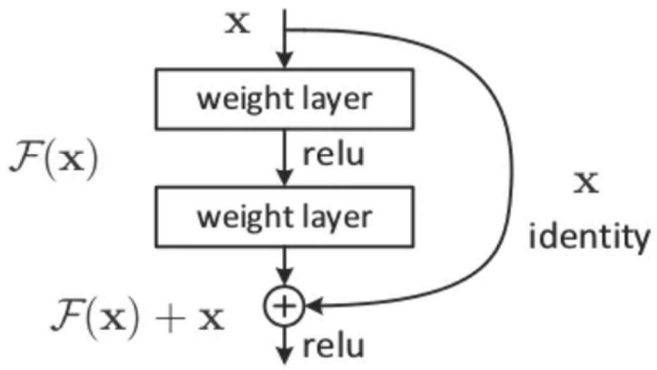



Image 2: A residual block

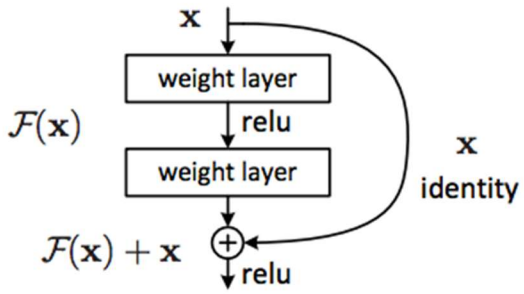
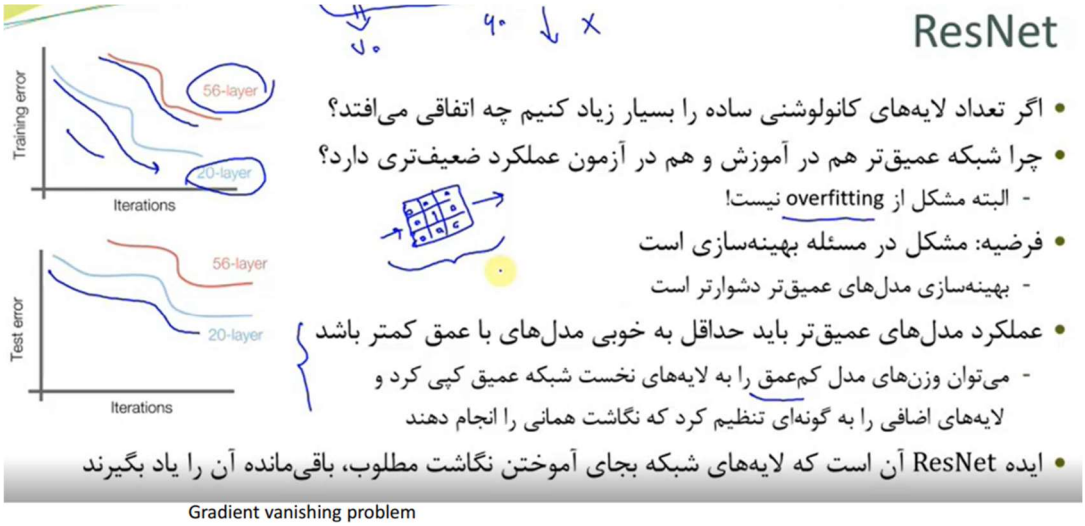



Figure 2. Residual learning: a building block.

The picture above is the most important thing to learn from this article. For developers looking to quickly implement this and test it out, the most important modification to understand is the 'Skip Connection', identity mapping. This identity mapping does not have any parameters and is just there to add the output from the previous layer to the layer ahead. However, sometimes x and F(x) will not have the same dimension. Recall that a convolution operation typically shrinks the spatial resolution of an image, e.g. a 3x3 convolution on a 32 x 32 image results in a 30 x 30 image. The identity mapping is multiplied by a linear projection W to expand the channels of shortcut to match the residual. This allows for the input x and F(x) to be combined as input to the next layer.

$$\mathbf{y} = \mathcal{F}(\mathbf{x}, \{W_i\}) + W_s\mathbf{x}.$$




Gradient vanishing problem

قبل از ResNet، شبکه‌های عصبی عمیق به دلیل مشکل ناپدید شدن گرادیان ( vanishing gradient problem) ، عمق خود را محدود می‌کردند. ResNet معماری جدیدی به نام بلوک باقیمانده (residual block) را معرفی کرد که به شبکه‌ها اجازه می‌دهد با استفاده از اتصالات پرش ( skip connections) که لایه‌ها را دور می‌زنند، عمیق‌تر شوند. این اتصالات پرش به گرادیان ها اجازه می دهد تا مستقیماً از طریق شبکه جریان داشته باشند(flow directly through the network)، مشکل ناپدید شدن گرادیان را کاهش داده و به شبکه اجازه می دهد تا به طور مؤثرتری یاد بگیرد.

در واقع همانطور که لایه‌های بیشتری با استفاده از توابع فعال‌سازی خاص به شبکه‌های عصبی اضافه می‌شوند، گرادیان تابع تلفات(loss function) به صفر نزدیک می‌شود و آموزش شبکه را سخت می‌کند. برخی از توابع فعال سازی، مانند تابع سیگموئید(sigmoid)، یک فضای ورودی بزرگ را به یک فضای ورودی کوچک بین ۰ و ۱ کوبیده می کنند. بنابراین، تغییر زیاد در ورودی تابع سیگموئید باعث تغییر کوچکی در خروجی می شود. از این رو، مشتق کوچک می شود.

برای شبکه های کم عمق با تنها چند لایه که از این فعال سازی ها استفاده می کنند، این مشکل بزرگی نیست. با این حال، زمانی که از لایه‌های بیشتری استفاده می‌شود، می‌تواند باعث شود که گرادیان بسیار کوچک باشد تا آموزش به طور موثر کار کند.

گرادیان های شبکه های عصبی با استفاده از پس انتشار(backpropagation) پیدا می شوند. به بیان ساده، پس انتشار مشتقات شبکه را با حرکت لایه به لایه از لایه نهایی به لایه اولیه پیدا می کند. بر اساس قانون زنجیره، مشتقات هر لایه در شبکه ضرب می شوند (از لایه نهایی تا لایه اولیه) تا مشتقات لایه های اولیه را محاسبه کنند.

با این حال، وقتی n لایه پنهان از یک فعال سازی مانند تابع سیگموئید(sigmoid) استفاده می کنند، n مشتق کوچک با هم ضرب می شوند. بنابراین، با انتشار به لایه های اولیه، گرادیان به صورت تصاعدی(exponentially) کاهش می یابد.

یک گرادیان کوچک به این معنی است که وزن ها و سوگیری(biases) های لایه های اولیه به طور موثر در هر train session به روز نمی شوند. از آنجایی که این لایه های اولیه اغلب برای شناسایی عناصر اصلی داده های ورودی بسیار مهم هستند، می تواند منجر به عدم دقت کلی کل شبکه شود.

ساده ترین راه حل استفاده از توابع فعال سازی دیگر است، مانند ReLU، که مشتق کوچکی ایجاد نمی کند.

Residual Networks راه حل دیگری هستند، زیرا اتصالات باقی مانده(residual connections) را مستقیماً به لایه های قبلی ارائه می دهند. همانطور که در تصاویر بالا مشاهده می شود، اتصال باقیمانده مستقیماً مقدار ابتدای بلوک، x، را به انتهای بلوک اضافه می کند (F(x)+x). این اتصال باقی‌مانده از طریق توابع فعال‌سازی ای که مشتقات را له می‌کنند(squashes) نمی‌گذرد و در نتیجه مشتق کلی بلوک بالاتر می‌رود و صفر نمی شود.

به بیان دیگر:

Prior to the introduction of ResNet (Residual Network), deep neural networks faced the problem of degradation, where increasing the network's depth would lead to diminishing accuracy. The networks introduced before ResNet to go deeper were traditional feedforward networks with increasing numbers of layers. However, as the networks became deeper, they suffered from the vanishing gradient problem. The gradients propagated through the layers became extremely small, making it difficult for the earlier layers to learn meaningful representations and limiting the network's overall performance.

ResNet addressed this problem by introducing the concept of residual learning. In ResNet, skip connections (also known as shortcut connections) are added to the network, allowing the gradient to flow directly from the later layers to the earlier layers. This way, the vanishing gradient problem is mitigated, as the gradients have a shorter path to propagate through the network.

The skip connections in ResNet enable the network to learn residual mappings, which capture the difference between the desired mapping and the identity mapping. By learning these residuals, ResNet can optimize the network's weights to focus on the residual information, allowing for better representation learning and enabling the network to go deeper without suffering from the degradation problem.

In summary, ResNet introduced skip connections to address the vanishing gradient problem and degradation issue faced by deep neural networks. By allowing gradients to flow directly from later layers to earlier layers, ResNet enabled the training of much deeper networks, leading to improved accuracy and performance in tasks such as image classification.

سوال ۲

Source: https://towardsdatascience.com/understanding-and-calculating-the-number-of-parameters-in-convolution-neural-networks-cnns-fc88790d530d

1. **Input layer:** Input layer has nothing to learn, at it's core, what it does is just provide the input image's shape. So no learnable parameters here. Thus number of **parameters = 0.**
2. **CONV layer:** This is where CNN learns, so certainly we'll have weight matrices. To calculate the learnable parameters here, all we have to do is just multiply the by the shape of **width m, height n, previous layer's filters d** and account for all such filters **k in the current layer.** Don't forget the bias term for each of the filter. Number of parameters in a CONV layer would be : **((m * n * d)+1)* k)**, added 1 because of the bias term for each filter. The same expression can be written as follows: **((shape of width of the filter * shape of height of the filter * number of filters in the previous layer+1)*number of filters).** Where the term "filter" refer to the number of filters in the current layer.
3. **POOL layer:** This has got no learnable parameters because all it does is calculate a specific number, no backprop learning involved! Thus number of **parameters = 0.**
4. **Fully Connected Layer (FC):** This certainly has learnable parameters, matter of fact, in comparison to the other layers, this category of layers has the highest number of parameters, why? because, every neuron is connected to every other neuron! So, how to calculate the number of parameters here? You probably know, it is the product of the number of neurons in the current layer **c** and the number of neurons on the previous layer **p** and as always, do not forget the bias term. Thus number of parameters here are: **((current layer neurons c * previous layer neurons p)+1*c).**

الف) این ماژول Inception-ResNet-A است که در معماری inception v4 معرفی شده است.

برای محاسبه تعداد پارامترهای قابل آموزش و میدان تاثیر(receptive field) برای مربوط به این ماژول ک تصویر سه کاناله n×n به عنوان ورودی می گیرد، باید معماری ماژول و لایه های درگیر در آن را تحلیل کنیم. (میدانیم که Relu activation و pointwise add پارامتری ندارند.)

این ماژول از شاخه های زیر تشکیل شده است:

۱. شاخه ۰: یک کانولوشن ۱×۱.
۲. شاخه ۱: یک کانولوشن ۱×۱ و به دنبال آن یک کانولوشن ۳×۳.
۳. شاخه ۲: یک کانولوشن ۱×۱ و به دنبال آن یک کانولوشن ۳×۳ و به دنبال آن یک کانولوشن ۳×۳ دیگر.

علاوه بر این، پس از به هم پیوستن شاخه ها(concatenating the branches)، یک کانولوشن ۱×۱ نیز وجود دارد. بیایید تعداد پارامترهای هر شاخه را تجزیه و محاسبه کنیم.

**Trainable Parameters**

The number of trainable parameters for each convolutional layer can be calculated as (kernel_size * kernel_size * input_channels + 1) * output_channels, where the +1 accounts for the bias term.

ورودی دارای C_in=3 کانال می باشد، تعداد کانال های خروجی از هر شاخه به شرح زیر است:

۱. شاخه ۰: ۳۲ کانال.
۲. شاخه ۱: ۳۲ کانال.
۳. شاخه ۲: ۳۲ کانال.

پارامترهای هر شاخه:

1. Branch 0: 1×1×C_in×32 + 32 (bias) = 32×(C_in+1) = 32×(3+1) = 32×4 = 128
2. Branch 1:
    - 1×1×C_in×32 + 32 (bias) = 32×(C_in+1) = 32×(3+1) = 32×4
    - 3×3×32×32 + 32 (bias) = 32×(9×32+1) = 32×289
    - Total = 32×(C_in+1) + 32×289 = 32×4 + 32×289 = 32×293
3. Branch 2:
    - 1×1×C_in×32 + 32 (bias) = 32×(C_in+1) = 32×(3+1) = 32×4
    - 3×3×32×32 + 32 (bias) = 32×(9×32+1) = 32×289

- 3×3×32×32 + 32 (bias) = 32×(9×32+1) = 32×289
- Total = 32×(C_in+1) + 32×289 + 32×289 = 32×4 + 32×289 + 32×289 = 32×582

پس از الحاق (concatenating) تعداد کانال های ورودی لایه (1×1 Conv 256 linear) 32+32+32=96 می باشد. سپس، یک کانولوشن ۱×۱ با 256 کانال خروجی وجود دارد:

- 1×1×96×C_in + 256 (bias) = 256×(96+1) = 256×(96+1) = 256×97

مجموع پارامترهای قابل آموزش در این ماژول:

[32×(C_in+1)] + [32×(C_in+1) + 32×289 ]+ [32×(C_in+1) + 32×289 + 32×289] + [256×(96+1)] = 3×32×(C_in+1) + 97×256 + 3×32×289
= 3×32×(3+1) + 97×256 + 3×32×289 = 3×32×4 + 97×256 + 3×32×289
= 384 + 24,832 + 27,744 = 52,960

**Receptive Field**

The receptive field is the region in the input space that a particular CNN's feature is looking at (i.e., influenced by).

در این ماژول:

1. Branch 0: 1×1 convolution has a receptive field of 1×1.
2. Branch 1: 1×1 convolution followed by a 3×3 convolution has a receptive field of 3×3.
3. Branch 2: 1×1 convolution followed by two 3×3 convolutions has a receptive field of 5×5.

از آنجایی که شاخه ها در این ماژول به هم می پیوندند(concatenated)، میدان تاثیر موثر (effective receptive field) حداکثر در بین شاخه ها است که در این حالت 5×5 است.

به طور خلاصه، این ماژول دارای

3×32×4 + 97×256 + 3×32×289
= 384 + 24,832 + 27,744 = 52,960

پارامترهای قابل آموزش و میدان تاثیر(receptive field) 5×5 است.

ب)

برای محاسبه تعداد پارامترهای قابل آموزش در یک لایه Conv2D، عوامل زیر را در نظر می گیریم:

• تعداد فیلترها(Number of filters): این عمق نقشه ویژگی خروجی را تعیین می کند.

• اندازه هسته(Kernel size): اندازه فضایی فیلتر کانولوشن را مشخص می کند.

• تعداد کانال های ورودی(Number of input channels): در این حالت یک تصویر ورودی سه کاناله داریم.

تعداد پارامترهای قابل آموزش برای هر لایه کانولوشن(convolutional layer) را می توان به صورت (اندازه_کرنل * اندازه_کرنل * کانالهای_ورودی + ۱) * کانال_خروجی محاسبه کرد، که در آن ۱+ برای عبارت bias محاسبه می شود.

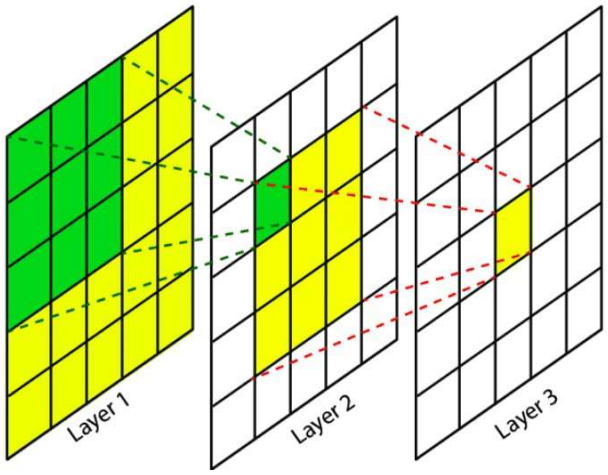
$$(kernel\_size * kernel\_size * input\_channels + 1) * output\_channels$$

A)

در حالت A، اولین لایه Conv2D دارای 448 = 16 * (1 + 3 * 3 * 3) پارامتر قابل آموزش(یادگیری) است که 3 * 3 اندازه هسته، 3 تعداد کانال های ورودی و 16 تعداد فیلترها است. لایه دوم Conv2D دارای 4640 = 32 * (1 + 16 * 3 * 3) پارامتر قابل آموزش است که 16 تعداد کانال های ورودی لایه قبلی است. بنابراین در مجموع، حالت A دارای 5088 = 4640 + 448 پارامتر قابل یادگیری است.

میدان تاثیر(receptive field)، ناحیه ای در فضای ورودی است که یک ویژگی خاص CNN به آن نگاه می کند (یعنی تحت تأثیر آن).

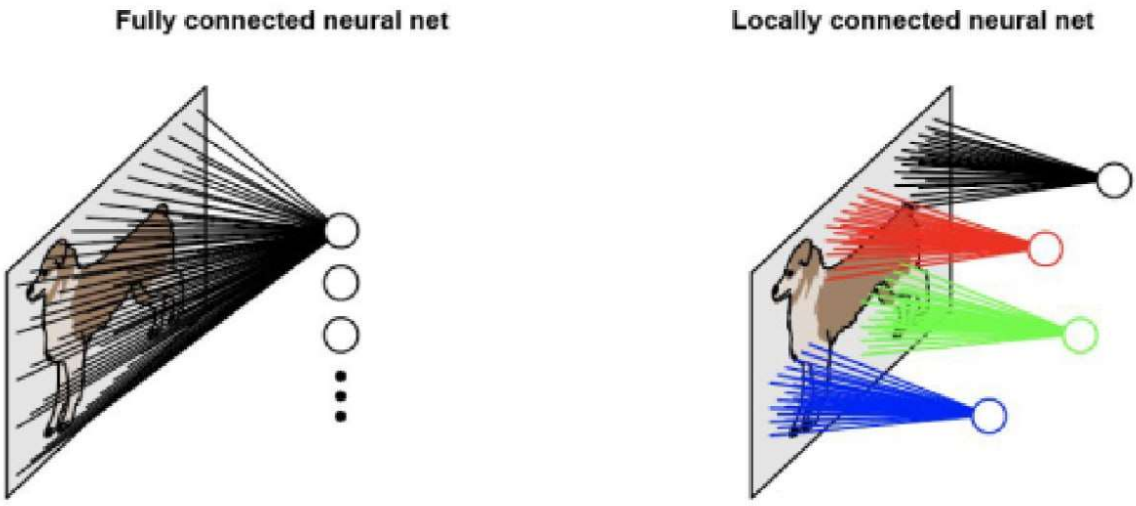
در این ماژول کانولوشن ۳×۳ و به دنبال آن کانولوشن ۳×۳ داریم که در نهایت میدان تاثیر ۵×۵ دارد. بنابراین میدان تاثیر( receptive field) کل ماژول در این مورد ۵×۵ است.



B)

حال بیایید حالت B را در نظر بگیریم که از لایه های LocallyConnected2D استفاده می کند.

LocallyConnected2D layers have a similar structure to Conv2D layers but with the difference that they do not share weights across different spatial locations. Each spatial location in the input has its own set of weights.



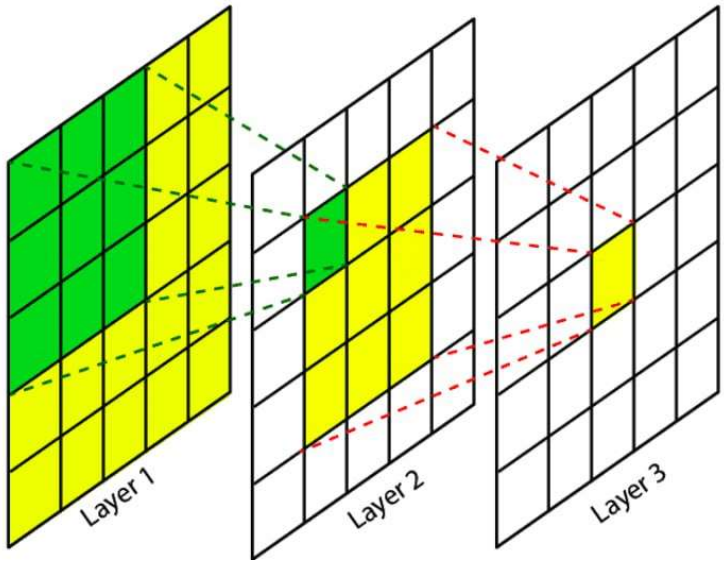

در واقع میدان تاثیر لایه LocallyConnected2D مانند لایه کانولوشنی (*Conv2D*) ولی پارامتر های آن در تعداد نورون های لایه بعد ضرب می شوند چرا که هر نورون لایه بعد به صورت محلی و با توجه به سایز kernel به نورون های لایه قبل وصل است و دارای وزن های ویژه خودش است.

در حالت B، اولین لایه LocallyConnected2D دارای (3 * 3 * 3 + 1) * 16 * (n - 2) * (n - 2) پارامترهای قابل آموزش است که (n - 2) * (n - 2) تعداد فیلدهای دریافتی مختلف در خروجی (number of different receptive fields in the output) است. دومین لایه LocallyConnected2D دارای (3 * 3 * 16 + 1) * 32 * (n - 4) * (n - 4) پارامترهای قابل یادگیری است. بنابراین در مجموع حالت B دارای

[ (3 * 3 * 3 + 1) * 16 * (n - 2) * (n - 2) ] + [ (3 * 3 * 16 + 1) * 32 * (n - 4) * (n - 4) ]
= [ (28) * 16 * (n - 2) * (n - 2) ] + [ (145) * 32 * (n - 4) * (n - 4) ]
= [ 448 * (n - 2) * (n - 2) ] + [ 4640 * (n - 4) * (n - 4) ]
= ( 448 * (n - 2)^2 ) + ( 4640 * (n - 4)^2 )

پارامترهای قابل یادگیری است.

در این ماژول لایه ۳×۳ LocallyConnected2D و به دنبال آن لایه ۳×۳ LocallyConnected2D داریم که در نهایت میدان پذیرنده ۵×۵ دارد.
بنابراین میدان تاثیر( receptive field) کل ماژول در این مورد ۵×۵ است.




receptive field برای هر لایه برای هر دو حالت یکسان است و برابر با اندازه هسته است که (۳،۳) است. از آنجایی که از دو لایه (۳،۳) استفاده کردیم، receptive field در انتها ۵×۵ شد. به این معنی که هر واحد در خروجی هر لایه به یک ناحیه (۵،۵) در ورودی آن لایه متصل است.

به طور خلاصه:

- حالت A در مجموع 5088 پارامتر قابل آموزش دارد.
- حالت B مجموعاً ( (4 - n) * (4 - n) * 4640 ) + ( (2 - n) * (2 - n) * 448) پارامترهای قابل یادگیری دارد.
- هر دو حالت در مجموع دارای میدان تاثیر(receptive field) 5x5 هستند.

## سوال ۳

الف)

کد من، یک شبکه عصبی پیچشی ساده را برای دسته‌بندی تصاویر در مجموعه داده CIFAR-10 آموزش می‌دهد و سپس نمودارهای مربوط به دقت و تابع هزینه را نمایش می‌دهد. دسته‌بندی تصاویر CIFAR-10، یک مسئله شناخت الگو در بین ده کلاس مختلف است.

در این کد:

۱. ابتدا بسته و کتابخانه های مورد نیاز برای اجرای کد شاملnumpy ، tensorflowو keras فراخوانی می‌شوند.

۲. مجموعه داده CIFAR-10 با استفاده از تابع **load_data()**در متغیرهای **x_train**و **y_train** برای آموزش و **x_val**و **y_val**برای اعتبارسنجی بارگیری می‌شود.

۳. اندازه مجموعه داده‌ها و برچسب‌ها با استفاده از تابع **shape**نمایش داده می‌شود.

۴. سپس تعدادی نمونه از هر کلاس داده آموزشی گرفته شده و در قالب یک نمودار نمایش داده می‌شوند.

۵. مقادیر پیکسل تصاویر نرمال شده و برچسب‌ها به بردارهای one-hot تبدیل می‌شوند.

۶. معماری شبکه عصبی ساده با استفاده از کلاس **Sequential**تعریف می‌شود. این مدل شامل چندین لایه پیچشی، لایه‌های تغییرشکل، لایه‌های چگال و برچسب‌گذاری است.

۷. با استفاده از تابع **summary()**، جزئیات معماری شبکه نمایش داده می‌شود.

۸. مدل با استفاده از تابع **compile()**تنظیم می‌شود. در اینجا، بهینه‌ساز"adam" ، تابع هزینه "categorical_crossentropy"و معیار "دقت" برای ارزیابی استفاده می‌شود.

۹. مدل با استفاده از تابع **fit()**روی داده‌های آموزشی آموزش داده می‌شود. این تابع تعدادی پارامتر مانند اندازه دسته، تعداد اپاک‌ها و داده‌های اعتبارسنجی را دریافت می‌کند.

۱۰. در نهایت، تابع **plot_acc_loss()**تعریف شده است که نمودارهای دقت و تابع هزینه را بر اساس تاریخچه آموزش مدل رسم می‌کند.

ب) کد من مدل را با استفاده از تکنیک داده افزایی آموزش می‌دهد. این تکنیک با استفاده از **ImageDataGenerator**از تنوع داده‌ها برای آموزش شبکه استفاده می‌کند. توضیحات مربوط به هر پارامتر در کد آورده شده است. برای اطلاعات بیشتر در مورد هر پارامتر می‌توانید به مستندات TensorFlowمراجعه کنید.

۱. ابتدا یک شی **ImageDataGenerator**تعریف می‌شود که انواع تغییراتی را روی تصاویر اعمال می‌کند.

۲. با استفاده از تابع **()fit**بر روی مجموعه داده آموزشی، مدل آموزش داده شده است. این تابع داده‌ها را تصادفی تغییر داده و در هر دوره از آموزش، دسته‌های جدیدی از داده‌ها تولید می‌کند.

۳. یک مدل جدید با نام **augmented_model**تعریف می‌شود که همان معماری قبلی را دارد.

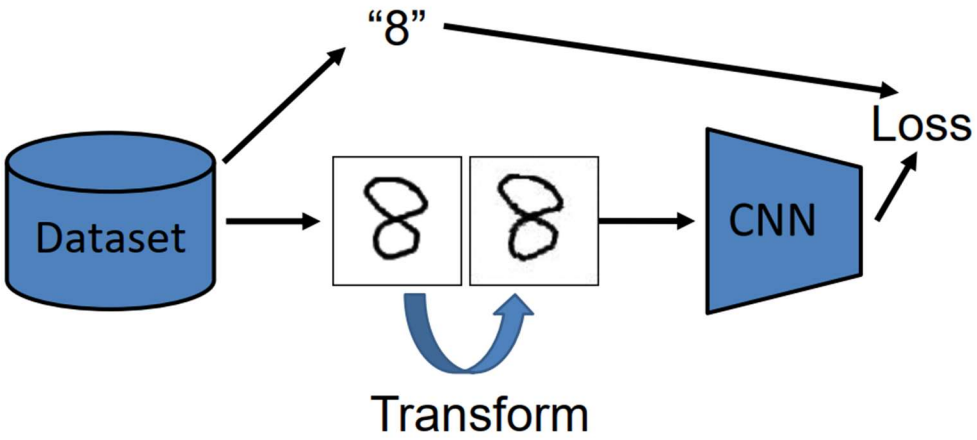
۴. معماری مدل با استفاده از تابع **()summary**نمایش داده می‌شود.

۵. مدل با استفاده از تابع **()compile**تنظیم می‌شود، همانند مدل اصلی.

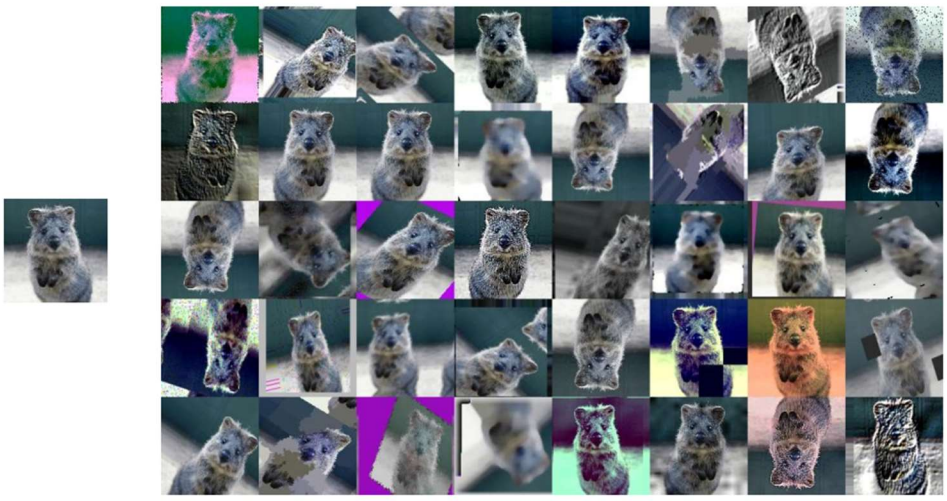
۶. مدل با استفاده از تابع **()fit**و با ورودی‌های تولید شده توسط **()datagen.flow**آموزش داده می‌شود. در این حالت، داده‌های آموزشی به صورت دسته‌هایی از داده‌های تغییر یافته تولید شده توسط **ImageDataGenerator**به مدل داده می‌شوند.

۷. تابع **()plot_acc_loss**تعریف شده است که نمودارهای دقت و تابع هزینه را بر اساس تاریخچه آموزش مدل ترسیم می‌کند. این نمودارها نشان می‌دهند که آیا استفاده از توسعه داده بهبودی در عملکرد مدل داشته است یا خیر.

## داده‌افزایی



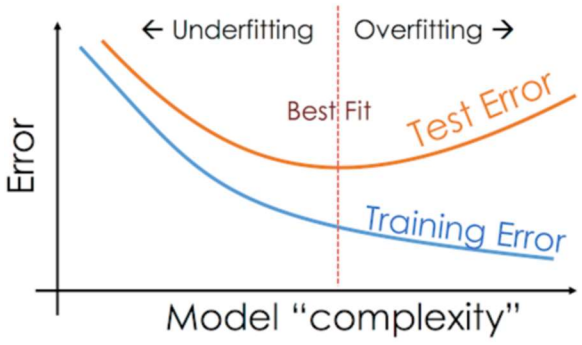
# داده‌افزایی

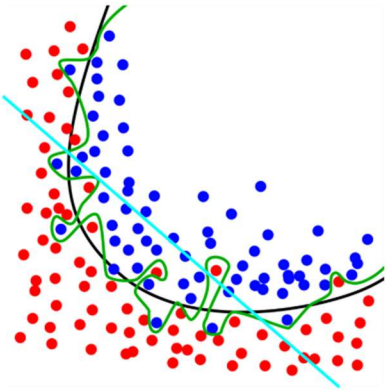


ج)

# Overfitting vs Underfitting

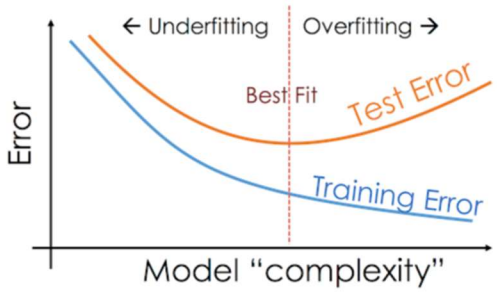
- یک مسئله اساسی در یادگیری ماشین این است که چگونه الگوریتمی بسازیم که نه تنها بر روی داده‌های آموزشی بلکه برای ورودی‌های جدید نیز به خوبی عمل کند



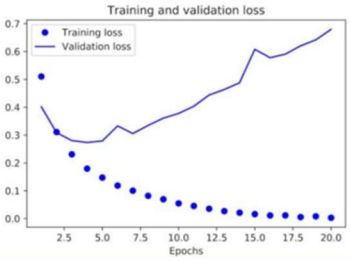
# بهینه‌سازی و تعمیم‌دهی

- بهینه‌سازی به تعیین پارامترهای مدل برای به دست آوردن بهترین عملکرد ممکن در داده‌های آموزشی (یادگیری در ML) اشاره دارد
- تعمیم‌دهی به نحوه عملکرد مناسب مدل آموزش دیده بر روی داده‌هایی که تا کنون مشاهده نکرده است اشاره دارد
- هدف دستیابی به تعمیم‌دهی مناسب است
  - اما کنترلی بر روی تعمیم‌دهی نداریم!
  - تنها می‌توانیم بر اساس داده‌های آموزشی پارامترهای مدل را تعیین کنیم




# بهینه‌سازی و تعمیم‌دهی

- در ابتدای آموزش، بهینه‌سازی و تعمیم‌دهی با هم کاملا مرتبط هستند
  - به مدل گفته می‌شود underfit است
  - شبکه هنوز تمام الگوهای مرتبط با مسئله مورد نظر در داده‌های آموزشی را یاد نگرفته است
- پس از چند تکرار، بهبود تعمیم‌دهی متوقف می‌شود و سپس شروع به تنزل می‌کند
  - مدل شروع به overfit شدن می‌کند
  - الگوهایی را می‌آموزد که مخصوص داده‌های آموزشی است اما ارتباط درستی با مسئله مورد نظر ندارد و گمراه‌کننده است





منابع:


Optimizing Convolutional Neural Network Performance by Mitigating Underfitting and Overfitting | IEEE Conference Publication | IEEE Xplore

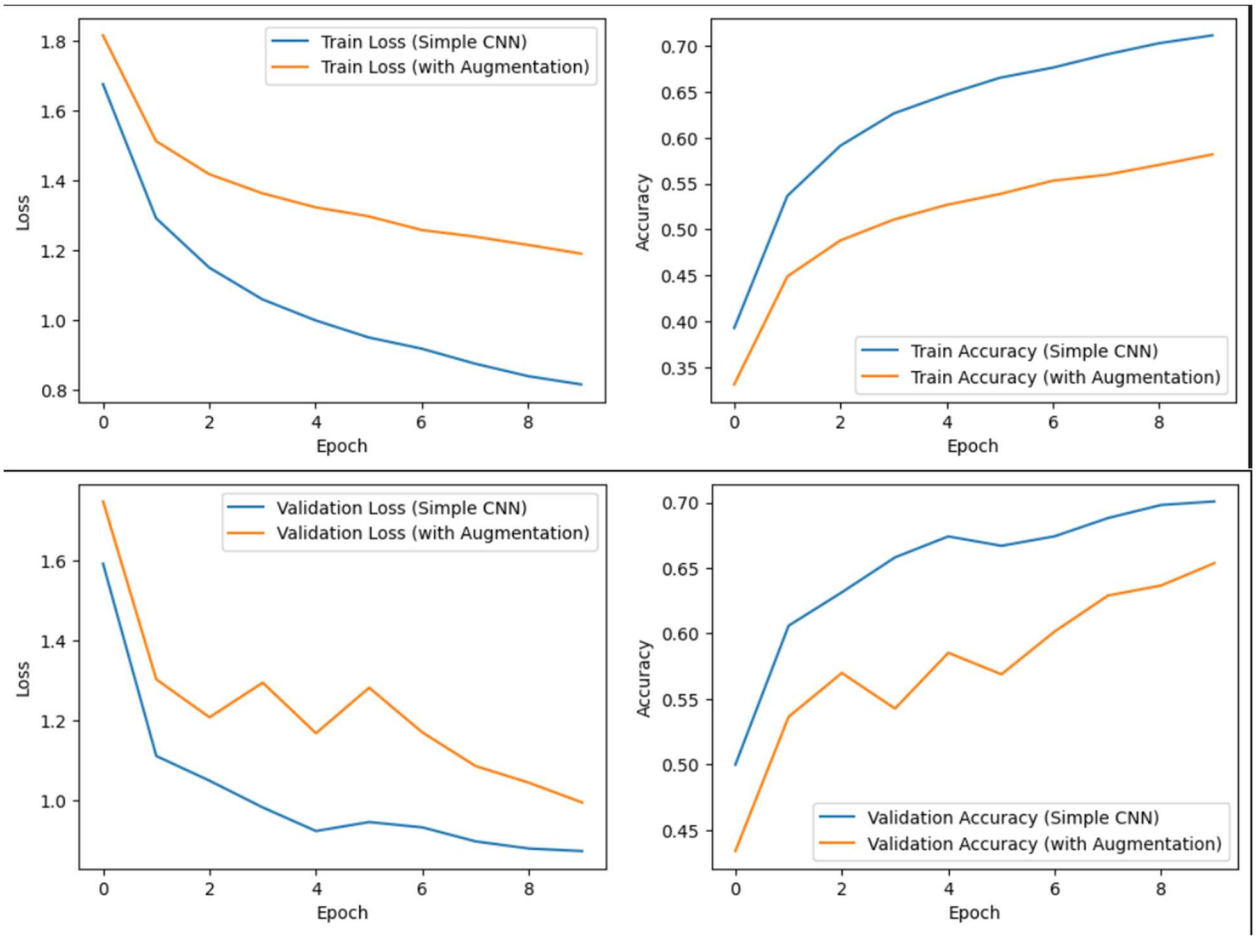
Deep Learning #3: More on CNNs & Handling Overfitting | by Rutger Ruizendaal | Towards Data Science

What is Data Augmentation in a CNN? Python Examples (nnart.org)

در فرآیند آموزش یادگیری عمیق، اغلب با ( underfitting) و بیش‌برازش( overfitting) مواجه می‌شویم که منجر به عملکرد ضعیف تعمیم شبکه(generalization performance) می‌شود. بر اساس یک شبکه عصبی کانولوشنی (CNN)، این مدل را می توان با کاهش عدم تناسب(underfitting) و اضافه برازش(mitigating underfitting and overfitting) بهینه کرد. با ترکیب رویکردهای متعدد، دقت مدل را می توان با تنظیم نرخ یادگیری(learning rate) و افزودن منظم سازی(regularization) و غیره بهبود بخشید.

داده افزایی(Data augmentation) یکی از تکنیک های کاهش اضافه برازش( reducing overfitting) است. با این حال، اگر افزایش داده ها به درستی انجام نشود، underfitting می تواند مشکل ساز شود. تعداد دوره های آموزشی( training epochs ) باید افزایش یابد تا مقدار اضافی ویژگی های داده های آموزشی را منعکس کند. اگر بهینه‌سازی روی نمونه‌های کافی انجام نشود، ممکن است (sub-optimal configuration) داشته باشد.

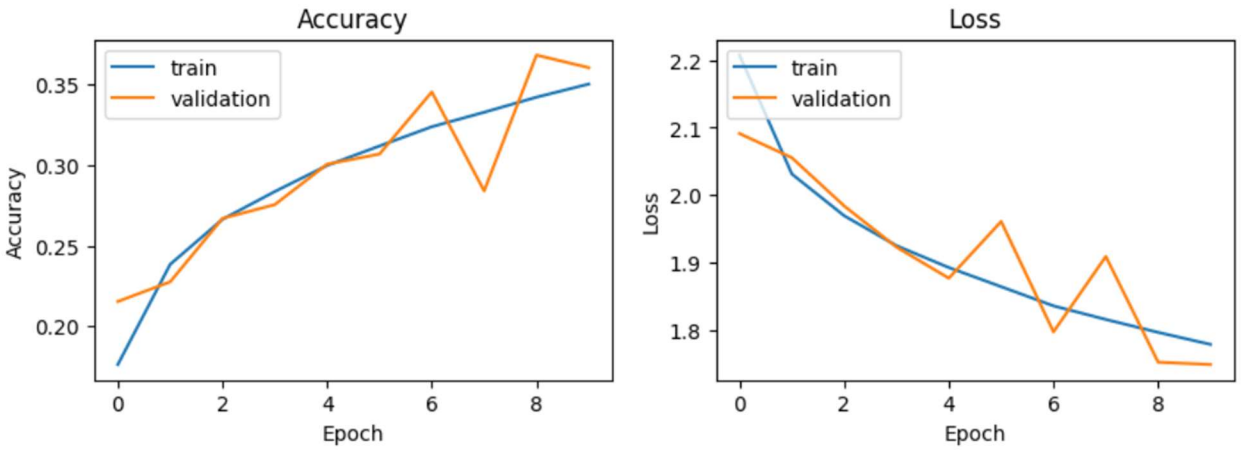
با توجه به نکات بالا به تحلیل نمودار های قسمت الف و ب می پردازیم:




معمولا از data augmentation برای جلوگیری از overfitting استفاده میشود. ولی در این سوال چون فقط گفته شده است که 10 epoch داده ها آموزش داده شوند در نمودار های بالا overfitting دیده نمی شود.

البته دلیل آن می تواند این باشد که در ساختار مدل ساده و مدل داده افزوده از لایه های Dropout و BatchNormalization استفاده کرده ام و همچنین دلیل دیگر آن این میتواند باشد تعداد داده های train کم نیست(50000). یکی دیگه از دلایل هم می تواند این باشد که من از داده افزایی بدرستی استفاده نکردم چرا که اگر افزایش داده ها به درستی انجام نشود، underfitting می تواند مشکل ساز شود.
با این حال اگر تعداد epochها بیشتر بود شاید مدل ساده زود تر از مدل data augmented دارای مشکل overfitting می شد.
همچنین با توجه به نمودار های بالا می بینیم که مدل ساده از مدل داده افزوده اندکی بهتر (در 10 epoch) عمل کرده است.

د)




همان طور که میبینیم با تکنیک انتقال یادگیری به دقت تقریبا ۳۵٪ رسیده ایم که نسبت به حالت الف و ب دقت کمتری دارد. (الف:۷۰٪ [simple model] ، ب:۶۵٪[augmented model] )
یکی از دلایل این میتواند این باشد که به تعداد کافی epoch جلو نرفته ایم. دلیل دیگر هم می تواند این باشد که کلاس های دو مسئله تفاوت زیادی دارند و همچنین ابعاد آن ها نیز متفاوت است و شاید ResNet50 برای مسئله ImageNet از ویژگی های ظریف تری استفاده می کند چرا که ورودی آن ابعاد بیشتری دارد. همچنین شاید اگر از **fine-tuning** در آخر استفاده میکردیم به نتیجه بهتری می رسیدیم. علاوه بر این چون مدل دارای لایه های زیادی است فاز training آن نیز نسبت به حالت های الف و ب بیشتر طول می کشد.

**Transfer learning** consists of taking features learned on one problem, and leveraging them on a new, similar problem. For instance, features from a model that has learned to identify racoons may be useful to kick-start a model meant to identify tanukis.

The most common incarnation of transfer learning in the context of deep learning is the following workflow:

1. Take layers from a previously trained model.
2. Freeze them, so as to avoid destroying any of the information they contain during future training rounds.
3. Add some new, trainable layers on top of the frozen layers. They will learn to turn the old features into predictions on a new dataset.
4. Train the new layers on your dataset.

A last, optional step, is **fine-tuning**, which consists of unfreezing the entire model you obtained above (or part of it), and re-training it on the new data with a very low learning rate. This can potentially achieve meaningful improvements, by incrementally adapting the pretrained features to the new data.

توضیحات کد من به شرح زیر است:

۱. از مدل **ResNet50** با استفاده از انتقال یادگیری (transfer learning) استفاده می‌شود.

۲. یک مدل پایه از مدل پیش آموزش دیده **ResNet50** ایجاد می‌شود **include_top=False**. نشان می‌دهد که لایه بالا به عنوان قسمتی از مدل استفاده نشود.

۳. مدل پایه freeze می‌شود، به این معنی که وزن‌های آموزش دیده شده در مدل پایه ثابت می‌مانند و در طول آموزش به روز نمی‌شوند.

۴. مدل جدید با استفاده از معماری مشخص شده تعریف می‌شود. ابتدا تصاویر ورودی با استفاده از لایه **Resizing** به ابعاد 224*224 تغییر اندازه داده می‌شوند. سپس مدل پایه **base_model** اضافه می‌شود.

۵. بعد از لایه پایانی مدل پایه، یک لایه **GlobalAveragePooling2D** قرار داده می‌شود تا اطلاعات فضایی از تصاویر استخراج شده و بازنمایی به صورت برداری دارای ابعاد ثابت صورت بگیرد.

۶. سپس دو لایه تمام متصل (fully connected layers) با تابع فعالسازی ReLU اضافه می‌شوند. این لایه‌ها به عنوان لایه‌های تمام متصل (fully connected layers) برای طبقه‌بندی نهایی استفاده می‌شوند.

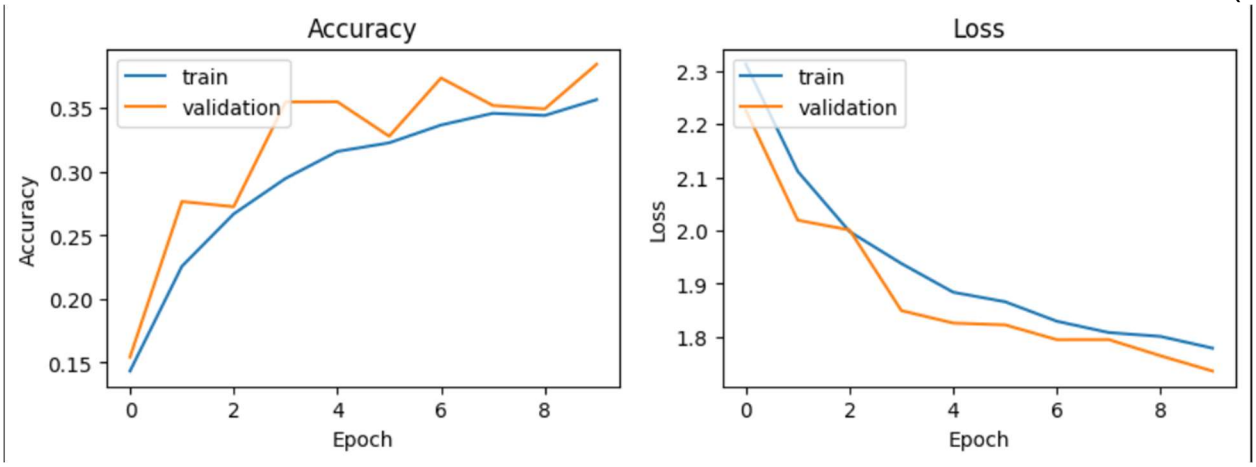
۷. در نهایت، مدل جدید با تابع فعالسازی softmax برای طبقه‌بندی چند دسته‌ای آموزش داده می‌شود.

۸. مدل تعریف شده با استفاده از تابع **()compile**تنظیم می‌شود، همانند مدل‌های قبلی.

۹. مدل با استفاده از تابع **()fit**آموزش داده می‌شود. داده‌های آموزش و ارزیابی به عنوان ورودی تابع داده می‌شوند.

۱۰. تابع **()plot_acc_loss**تعریف شده است که نمودارهای دقت و تابع هزینه را بر اساس تاریخچه آموزش مدل ترسیم می‌کند. این نمودارها نشان می‌دهند که آیا استفاده از انتقال یادگیری بهبودی در عملکرد مدل داشته است یا خیر.

ه)




همان طور که میبینیم با این تکنیک به دقت تقریبا ۴۰٪ رسیده ایم که نسبت به حالت الف و ب دقت کمتری دارد ولی از حالت د دقت بیشتری دارد. (الف:۷۰٪ [simple model] ، ب:۶۵٪ [augmented model]، د:۳۵٪ [transfered model])

یکی از دلایلی که خیلی خوب عمل نکرده است این میتواند این باشد که به تعداد کافی epoch جلو نرفته ایم. دلیل دیگر هم می تواند این باشد که کلاس های دو مسئله ImageNet وCifar10 تفاوت زیادی دارند و همچنین ابعاد آن ها نیز متفاوت است و شاید ResNet50 برای مسئله ImageNet از ویژگی های ظریف تری استفاده می کند چرا که ورودی آن ابعاد بیشتری دارد. همچنین شاید اگر از **fine-tuning** در آخر استفاده میکردیم به نتیجه بهتری می رسیدیم.

علاوه بر این چون مدل دارای لایه های کمتری نسبت به حالت د است فاز training آن نیز نسبت به کمتر طول می کشد.

خط به خط توضیحات کد زیر به شرح زیر است:

۱. در این خط، مدل ResNet50 بدون لایه طبقه‌بندی بالا (top classification layer) بارگذاری می‌شود.

۲. سه بلوک اول مدل ResNet50 یخبندان(freeze) می‌شود تا وزن‌های آموزش دیده شده در این بلوک‌ها در طول آموزش به روز نشوند .همچنین بعد از اینکه به لایه conv3_block4_out رسیدیم آن را در متغیر x ذخیره میکنیم تا فقط از آن استفاده کنیم و از لایه های بعد استفاده نکنیم.

۳. در این قسمت، یک لایه Global Average Pooling اضافه می‌شود. این لایه اطلاعات فضایی از تصاویر را استخراج کرده و آنها را به شکل برداری با ابعاد ثابت تبدیل می‌کند.

۴. یک لایه fully-connected اضافه می‌شود. این لایه دارای ۱۰۲٤ نرون است و از تابع فعالسازی ReLU استفاده می‌کند.

۵. یک لایه logistic با تعداد دسته‌بندی‌ها در CIFAR10 به عنوان خروجی اضافه می‌شود. این لایه از تابع فعالسازی softmax استفاده می‌کند.

۶. با استفاده از **keras.Model**، مدل ایجاد می‌شود که ورودی‌ها و خروجی‌ها بر اساس مدل ResNet50و لایه‌های اضافه شده تعریف می‌شوند.

۷. مدل با استفاده از تابع **()compile**تنظیم می‌شود، همانند مدل‌های قبلی.

۸. مدل با استفاده از تابع **()fit**آموزش داده می‌شود. داده‌های آموزش و ارزیابی به عنوان ورودی تابع داده می‌شوند.

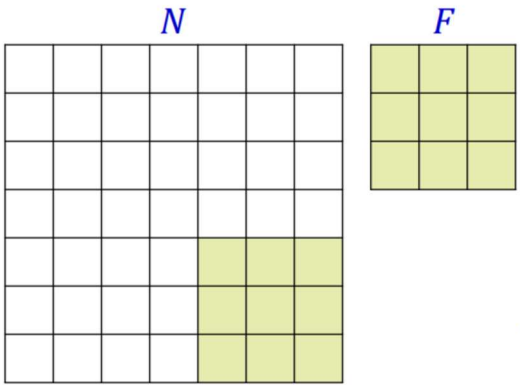
۹. تابع **()plot_acc_loss**تعریف شده است که نمودارهای دقت و تابع هزینه را بر اساس تاریخچه آموزش مدل ترسیم می‌کند.

سوال ۴

الف) منبع: https://chat.openai.com

## کانولوشن

- به دلیل کاهش محاسبات می‌توان پنجره را با گام بزرگتر جابجا کرد



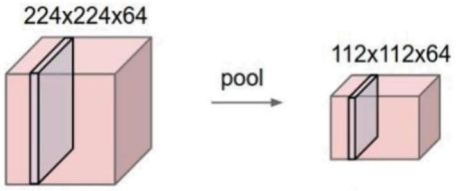

Stride=2

$$Output\ Size = \frac{N - F}{Stride} + 1$$

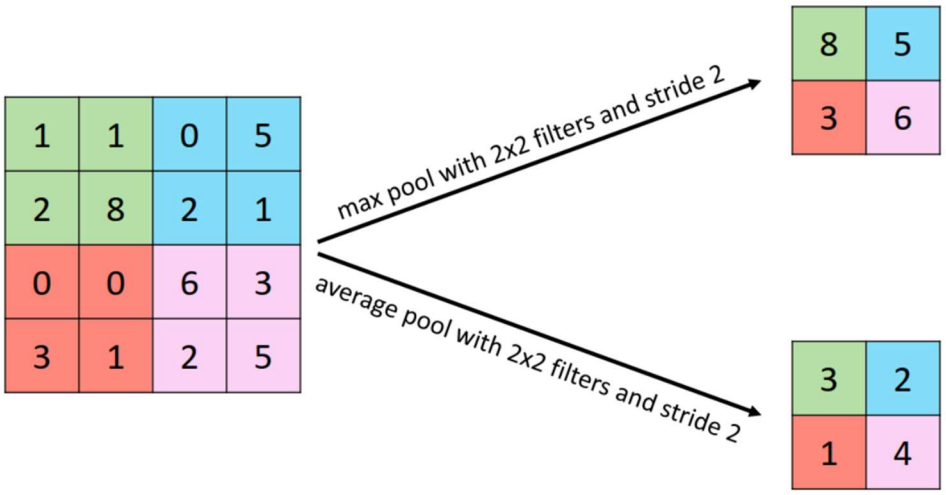
خروجی یک تصویر 3x3 است

## لایه Pooling

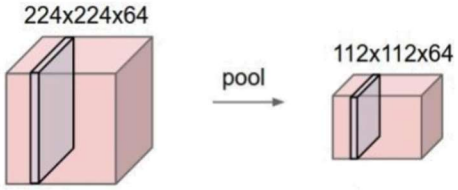
- لایه Pooling در خروجی لایه‌های کانولوشنی قرار می‌گیرد و پیکسل‌های همسایه را با یکدیگر ترکیب می‌کند تا ابعاد نقشه‌های ویژگی کاهش بیابد
- یکی از دستاوردهای اصلی لایه Pooling کاهش ابعاد نورون‌ها و کاهش تعداد پارامترهای شبکه است
- لایه Pooling بر روی هر نقشه فعالیت به صورت جداگانه اعمال می‌شود
- میانگین و ماکزیمم متداول هستند



## لایه Pooling




## لایه Pooling




- ورودی یک حجم با ابعاد $W_1 \times H_1 \times D_1$ است
- ابرپارامترهای لایه Pooling عبارتند از:
  - نحوه تلفیق
  - اندازه فیلترها $F$
  - اندازه گام $S$
  - مقدار گسترش مرزها $P$
- خروجی یک حجم با ابعاد $W_2 \times H_2 \times D_2$ است
- پارمتر ندارد

- $W_2 = (W_1 - F + 2P)/S + 1$
- $H_2 = (H_1 - F + 2P)/S + 1$
- $D_2 = D_1$

Stride یا گام پارامتری از عملیات کانولوشن است که اندازه گام هسته (kernel) را هنگام حرکت در تصویر ورودی مشخص می کند. Stride یا گام ۱ به این معنی است که هسته (kernel) در یک زمان یک پیکسل حرکت می کند، در حالی که گام ۲ به این معنی است که هر بار به اندازه دو پیکسل حرکت می کند. افزایش گام، ابعاد فضایی نقشه ویژگی (spatial dimensions of the feature map) خروجی را کاهش می‌دهد و به طور موثر ورودی را پایین می‌آورد (downsampling the input ). همچنین میدان تاثیر (receptive field) را افزایش می‌دهد.

Pooling (ادغام) یکی دیگر از عملیات مورد استفاده برای کاهش ابعاد فضایی(spatial dimensions) نقشه ویژگی(feature map) است. این کار با تقسیم ورودی به مناطق غیر مشترک(غیر همپوشانی) و اعمال یک تابع تجمع(aggregation function) مانند حداکثر(max) یا میانگین(average) برای هر منطقه کار می کند. ادغام معمولاً بعد از لایه های کانولوشن برای کاهش پیچیدگی محاسباتی (computational complexity) شبکه اعمال می شود. همچنین میدان تاثیر (receptive field) را افزایش می‌دهد.

انتخاب گام(stride) و ادغام(Pooling) می تواند تاثیر قابل توجهی بر عملکرد یک شبکه عصبی داشته باشد. استفاده از یک گام بزرگتر یا ادغام می تواند پیچیدگی محاسباتی شبکه را کاهش دهد، اما همچنین می تواند منجر به از دست رفتن اطلاعات مکانی شود. از سوی دیگر، استفاده از یک گام کوچکتر یا بدون ادغام می تواند اطلاعات فضایی بیشتری را حفظ کند، اما می تواند پیچیدگی محاسباتی شبکه را افزایش دهد. انتخاب بهینه گام و ادغام به مسئله و مجموعه داده خاص ( specific problem and dataset) بستگی دارد.

Stride in convolutional layers refers to the step size with which the kernel (filter) moves across the input image during the convolution operation. It determines the amount of spatial downsampling or subsampling that occurs. Stride specifies how many pixels the kernel moves horizontally and vertically after each convolution operation.

- Stride determines the step size of the kernel while scanning the input feature map.
- A stride of 1 means the kernel moves one position at a time, resulting in overlapping receptive fields.
- A stride of 2 means the kernel skips one position, resulting in non-overlapping receptive fields.
- Strided convolutions reduce the spatial dimensions of the feature map, resulting in a smaller output size compared to the input size.
- It reduces computational complexity by reducing the number of operations and parameters, leading to faster processing.
- Stride can affect the model's receptive field and the amount of spatial information preserved in the output feature map.

Pooling, on the other hand, is a separate operation that reduces the spatial dimensions of the feature map by downsampling. It divides the input into non-overlapping regions and aggregates the information within each region, such as taking the maximum or average value.

- Pooling is a separate operation performed after convolution, typically to downsample the feature map and extract important features.
- Pooling regions (e.g., max pooling or average pooling) are non-overlapping, meaning they do not share common elements.
- Pooling reduces the spatial dimensions of the feature map, similar to strided convolutions.
- The pooling operation summarizes the local information within each pooling region, reducing the amount of spatial information.
- Pooling helps create spatial invariance, making the network less sensitive to translations and increasing robustness to small spatial shifts.

The concept of stride in convolutional layers and pooling are related to the spatial downsampling of feature maps in convolutional neural networks (CNNs), but they serve different purposes and have distinct effects on network performance.

1. Stride in Convolutional Layers:
    - Stride refers to the step size by which the convolutional kernel moves across the input feature map.
    - When performing a convolution operation with stride greater than 1, the kernel skips over some input positions, resulting in a reduced output size.
    - Strided convolutions can be used to control the spatial resolution of feature maps and adjust the level of detail captured by the network.
    - Larger stride values lead to greater downsampling and reduced spatial resolution, potentially discarding fine-grained details but increasing computational efficiency.

- Strided convolutions primarily aim to extract high-level features and capture larger spatial contexts.
- Larger stride reduces the spatial dimensions of the output feature map, leading to spatial downscaling and information loss.
- Smaller stride allows more overlap between receptive fields, capturing more local information.

2. Pooling:
    - Pooling is a separate operation performed after convolutions, where local regions of the feature map are summarized into a single value.
    - Pooling regions, such as max pooling or average pooling, non-linearly downsample the feature map.
    - Pooling operates on non-overlapping regions and reduces the spatial dimensions of the feature map, resulting in spatial invariance and robustness to translations.
    - Pooling helps in reducing the spatial resolution, extracting dominant features, and providing a form of regularization by preventing overfitting.
    - It reduces the computational complexity and the number of parameters in subsequent layers.

Effects on Network Performance:

- Strided convolutions and pooling both contribute to spatial downsampling, reducing the spatial resolution of feature maps and capturing more abstract representations.
- Strided convolutions may sacrifice fine-grained details, but they can help capture larger context and global information, which can be beneficial for tasks where global understanding is important.
- Pooling aids in capturing robust and invariant features, making the network more tolerant to small spatial shifts and variations, improving generalization performance.

- Both stride and pooling help manage computational complexity by reducing the number of operations and parameters in subsequent layers.
- The choice of stride and pooling size depends on the specific task, dataset, and network architecture, and striking a balance between preserving spatial information and reducing computational cost is crucial.

In summary, stride in convolutional layers determines the step size during convolution, affecting the downsampling and spatial resolution, while pooling summarizes local regions to downsample the feature map. They play complementary roles in capturing different levels of detail and context, managing computational complexity, and promoting spatial invariance and robustness in CNNs. The appropriate choices for stride and pooling size depend on the specific requirements and trade-offs of the given task.

ب)
منابع:

https://www.analyticsvidhya.com/blog/2020/01/fundamentals-deep-learning-activation-functions-when-to-use-them/
https://www.analyticsvidhya.com/blog/2021/03/binary-cross-entropy-log-loss-for-binary-classification/

برای مسئله داده شده طبقه بندی تصاویر محصولات معیوب از محصولات سالم:

۱. برای لایه‌های میانی شبکه، استفاده از تابع فعال‌سازی ReLU (Rectified Linear Unit) را پیشنهاد می‌کنم. ReLU یک تابع فعال سازی متداول در شبکه های عصبی کانولوشن است زیرا از نظر محاسباتی کارآمد است و به کاهش مشکل ناپدید شدن گرادیان( vanishing gradient problem) کمک می کند. همچنین باعث غیر خطی بودن(non-linearity) نیز می شود. برای آخرین لایه شبکه، چون مسئله طبقه بندی باینری (معیوب در مقابل سالم) است (binary classification problem)، میتوانیم از تابع فعال سازی سیگموئید(sigmoid) استفاده کنیم. یا اگر بیش از دو کلاس وجود داشت، از تابع فعال سازی softmax هم می توانیم استفاده کنیم. (چون در مسائل طبقه بندی(multiclass classification problem) تابع فعال سازی softmax کارآمد است.)

۲. برای این مسئله طبقه بندی باینری(binary classification problem)، یک تابع ضرر مناسب(loss function)، آنتروپی متقاطع باینری(binary cross-entropy) است زیرا یک کار(task) طبقه بندی باینری (معیوب یا سالم) است. این تابع ضرر عدم تشابه بین احتمالات پیش‌بینی‌شده و برچسب‌های کلاس واقعی را اندازه‌گیری می‌کند و معمولاً برای مسائل طبقه‌بندی باینری استفاده می‌شود. ( This loss function measures the dissimilarity between the predicted probabilities and the true class labels, and is commonly used for binary classification problems.)

۳. recall کلاس معیوب (recall of the defective class)

فرضیات:

TP = true positives (defective products classified as defective)

FP = false positives (healthy products classified as defective)

FN = false negatives (defective products classified as healthy)

در این سناریو، مهم است که تعداد محصولات معیوب که به عنوان سالم طبقه بندی می شوند (false negatives) و به دست مشتری می رسند، به حداقل برسد. این بدان معنی است که ما می خواهیم recall کلاس معیوب را به حداکثر برسانیم (به حداقل رساندن false negatives)، که به عنوان نسبت محصولات معیوب مثبت واقعی (TP) به همه محصولات معیوب(TP + FN) تعریف می شود.

(recall = ratio of true positive defective products to all defective products)

recall بزرگ(high recall) به این معنی است که اکثر محصولات معیوب به درستی شناسایی می شوند و تعداد محصولات معیوب که به دست مشتری می رسد کاهش می یابد.

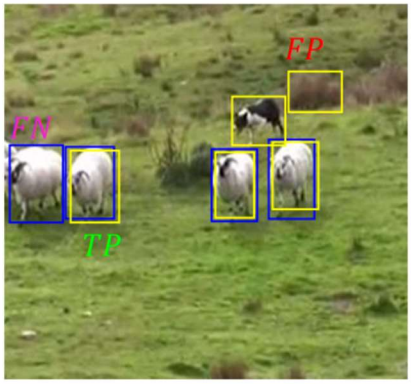
با این حال، در نظر گرفتن دقت(precision) نیز مهم است، که به عنوان نسبت محصولات معیوب مثبت واقعی به تمام محصولات طبقه بندی شده به عنوان معیوب تعریف می شود.

(precision = ratio of true positive defective products to all products classified as defective)

دقت بالا(high precision) به این معنی است که اکثر محصولات طبقه بندی شده به عنوان معیوب واقعا معیوب هستند و تعداد محصولات سالمی که به اشتباه به عنوان معیوب طبقه بندی شده(false positives) و دور ریخته شده اند را کاهش می دهد. تعادل بهینه بین recall و percision به مسئله خاص و هزینه های مرتبط با مثبت کاذب(FP) و منفی کاذب(FN) بستگی دارد.

## دقت متوسط (AP)

- در یک تصویر تشخیص‌های متفاوتی داریم که طبق شکل زیر تعریف می‌شوند:




$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

$$F_1 = 2\frac{Precision \times Recall}{Precision + Recall}$$

- حد آستانه برای پذیرش یک تشخیص را چند قرار دهیم؟

ج) شبکه های عصبی کانولوشنی (CNN) برای کار با داده های شبکه مانند مانند تصاویر (grid-like data such as images)، که در آن روابط فضایی(spatial relationships) بین پیکسل ها مهم است، طراحی شده اند. CNN ها احتمالا در کاربرد های زیر موفق نیستند و خیلی خوب عمل نمی کنند.

۱. طبقه بندی موضوع متن(Classification of the topic of the text):
داده های متنی شبکه ای(grid-like) نیستند و روابط فضایی(spatial relationships) بین کلمات یا کاراکترها وجود ندارد. در عوض، ترتیب و بافت(the order and context) کلمات برای درک معنای متن مهم است. به همین دلیل، انواع دیگر شبکه های عصبی مانند شبکه های عصبی بازگشتی (Recurrent Neural Networks یا RNN) یا ترانسفورماتورها (Transformers) ممکن است برای کارهای طبقه بندی متن مناسب تر باشند. و CNN ها احتمالا عملکرد خیلی خوبی نخواهند داشت.

۲. تشخیص گوینده از روی صوت (Identifying the speaker from the sound):
در حالی که داده های صدا را می توان به عنوان یک طیف نگار(spectrogram) نشان داد، که نمایش شبکه ای از فرکانس های موجود در صدا در طول زمان (grid-like representation of the frequencies present in the sound over time) است، شناسایی گوینده از روی صدا شامل درک الگوهای زمانی پیچیده و روابط بین فرکانس های مختلف است. به همین دلیل، سایر انواع شبکه های عصبی مانند RNN یا CNN های ۱ بعدی(1D CNNs) ممکن است برای تسک های شناسایی گوینده مناسب تر باشند.

۳. تحلیل جدول مربوط به مشتریان یک فروشگاه برای پیش بینی رفتار بعدی هر مشتری
( Analyzing the table related to the customers of a store to predict the next behavior of each customer ):
داده های جدولی شبکه ای نیستند و روابط فضایی بین ویژگی ها ندارند( Tabular data is not grid-like and does not have spatial relationships between features). در عوض، هر ویژگی جنبه متفاوتی از داده ها را نشان می دهد و ممکن است مقیاس و توزیع متفاوتی(different scale and distribution) داشته باشد. به همین دلیل، انواع دیگر مدل‌های یادگیری ماشین مانند درخت‌های تصمیم(decision trees)، جنگل‌های تصادفی( random forests) یا ماشین‌های تقویت گرادیان(gradient boosting machines) ممکن است برای تجزیه و تحلیل داده‌های جدولی مناسب‌تر باشند. همچنین شاید مدل‌های یادگیری عمیق مانند شبکه‌های عصبی پیش‌خور(feedforward neural networks) یا شبکه‌های عصبی مکرر (RNN)، که می‌توانند وابستگی‌های زمانی(temporal dependencies) را بهتر دریافت کنند مناسب باشند. و CNN ها احتمالا عملکرد بهتری نسبت به موارد ذکر شده نخواهند داشت.

د)

Convolutional Neural Networks (CNNs) are powerful models for image classification and other grid-like data, but they can have several problems:

- Large computational and memory requirements: CNNs can have millions of trainable parameters, which can make them computationally expensive to train and require large amounts of memory. This can be a problem when working with large datasets or high-resolution images.
  In fact, CNNs have Computational complexity problem. CNNs, especially deeper architectures, can be computationally intensive and require substantial resources for training and inference. Training on large datasets can also be time-consuming. Hardware accelerators like GPUs and TPUs are often used to speed up training and inference.
- Overfitting: CNNs are prone to overfitting, especially when the number of training examples is small relative to the number of parameters in the model. Overfitting occurs when the model learns to memorize the training data instead of generalizing to new examples, resulting in poor performance on unseen data.
  in other words, CNNs can have a large number of parameters, and with limited training data, they may overfit the training set and fail to generalize well to unseen data. Techniques like regularization, dropout, and data augmentation can help mitigate overfitting.
- Difficulty in understanding and interpreting the model: The internal workings of a CNN can be difficult to understand and interpret, making it challenging to explain how the model makes its predictions. This can be a problem in applications where interpretability is important.
- Sensitivity to changes in the input: CNNs can be sensitive to changes in the input such as rotation, scaling, or translation. This means that small changes in the input image can result in large changes in the output of the model. Data augmentation techniques can be used to mitigate this problem by artificially increasing the size of the training dataset with transformed versions of the original images.
- Limited ability to handle non-grid-like data: CNNs are designed to work with grid-like data such as images, where spatial relationships between pixels are important. They may not perform well on non-grid-like data such as text or tabular data, where other types of neural networks or machine learning models may be more suitable.
- Vanishing or exploding gradients: Deep CNNs with many layers can suffer from the vanishing gradient problem, where the gradients become extremely small as they propagate backward, leading to slow or no learning. On the other hand, gradients can also explode, causing unstable training. Techniques like gradient clipping, batch normalization, or skip connections (e.g., ResNet) help alleviate these issues.
- Sensitivity to hyperparameters: CNNs have several hyperparameters, such as learning rate, batch size, network architecture, and regularization parameters. Choosing

appropriate values for these hyperparameters can significantly impact the performance of the network, and finding the optimal values often requires experimentation and tuning.

ترجمه فارسی:
شبکه‌های عصبی پیچشی (CNN) مدل‌های قدرتمندی برای دسته‌بندی تصاویر و دیگر داده‌های شبکه‌ای( image classification and other grid-like data) هستند، اما ممکن است با چنین مشکلاتی مواجه شوند:

• نیاز به محاسبات بزرگ و حافظه (Large computational and memory requirements): شبکه‌های عصبی پیچشی می‌توانند دارای میلیون‌ها پارامتر قابل آموزش باشند که ممکن است آموزش آنها زمان‌بر و نیاز به حافظه بزرگی داشته باشد. این مشکل می‌تواند زمانی بروز یابد که با مجموعه‌داده‌ها یا تصاویر با رزولوشن بالا کار می‌کنید.
در واقع، مشکل پیچیدگی محاسباتی(Computational complexity) در شبکه‌های عصبی پیچشی وجود دارد. به خصوص در معماری‌های عمیق‌تر، آنها ممکن است نیازمند محاسبات گسترده و منابع قابل توجهی برای آموزش و پیش‌بینی باشند. به منظور سرعت‌بخشی در آموزش و پیش‌بینی، شتاب‌دهنده‌های سخت‌افزاری مانندGPU و TPU اغلب استفاده می‌شوند.
• بیش‌برازش (Overfitting): شبکه‌های عصبی پیچشی به بیش‌برازش حساس هستند، به خصوص زمانی که تعداد نمونه‌های آموزش کمتر از تعداد پارامترها در مدل باشد. بیش‌برازش هنگامی رخ می‌دهد که مدل یاد می‌گیرد تا داده‌های آموزش را به جای تعمیم به نمونه‌های جدید حفظ کند، که باعث کاهش عملکرد در داده‌های ناشناخته می‌شود. به عبارت دیگر، شبکه‌های عصبی پیچشی می‌توانند دارای تعداد زیادی پارامتر باشند و با داده‌های آموزش محدود، ممکن است به مجموعه آموزش بیش‌برازش کنند و توانایی تعمیم به خوبی به داده‌های ناشناخته را نداشته باشند. روش‌هایی مانند رگولاریزاسیون، قطره‌ریزی و افزایش داده می‌توانند به کاهش بیش‌برازش کمک کنند.
• دشواری در فهم و تفسیر مدل (Difficulty in understanding and interpreting the model): عملکرد داخلی یک شبکه عصبی پیچشی ممکن است دشوار در فهم و تفسیر باشد و بررسی اینکه مدل چگونه پیش‌بینی می‌کند مشکلاتی را به وجود آورد. این مسئله در برنامه‌هایی که قابلیت تفسیر پیش‌بینی‌ها مهم است، مشکل ساز می‌شود.
• حساسیت به تغییرات در ورودی (Sensitivity to changes in the input): شبکه‌های عصبی پیچشی حساس به تغییرات در ورودی مانند چرخش، مقیاس‌بندی یا ترجمه می‌باشند. این به این معنی است که تغییرات کوچک در تصویر ورودی ممکن است منجر به تغییرات بزرگ در خروجی مدل شود. تکنیک‌های افزایش داده می‌توانند به کاهش این مشکل با افزایش مصنوعی اندازه مجموعه آموزش با نسخه‌های تبدیل شده از تصاویر اصلی کمک کنند.
• قدرت محدود در کار با داده‌های غیرشبکه‌ای (Limited ability to handle non-grid-like data): شبکه‌های عصبی پیچشی برای کار با داده‌های شبکه‌ای مانند تصاویر طراحی شده‌اند، جایی که روابط فضایی بین پیکسل‌ها مهم است. آنها ممکن است در کار با داده‌های غیرشبکه‌ای مانند متن یا داده‌های جدولی به خوبی عمل نکنند و در این موارد، شبکه‌های عصبی یا مدل‌های یادگیری ماشین دیگری می‌تواند مناسب‌تر باشد.
• ناپدید شدن گرادیان یا افجار گرادیان (Vanishing or exploding gradients): شبکه‌های عصبی پیچشی عمیق با تعداد لایه‌های زیاد ممکن است با مشکل گرادیان ناپدید روبه‌رو شوند که در آن گرادیان‌ها هنگام پیشروی به صورت بسیار کوچک می‌شوند و عملکرد آموزش کند یا صفر می‌شود. به علاوه، گرادیان‌ها ممکن است به صورت ناگهانی بزرگ شوند که منجر به آموزش ناپایدار می‌شود. تکنیک‌هایی مانند کلیپ گرادیان، نرمال‌سازی دسته‌ای یا اتصالات پرش) مانند شبکه (ResNetبه کاهش این مشکلات کمک می‌کنند.
• حساسیت به هایپرپارامترها (Sensitivity to hyperparameters): شبکه‌های عصبی پیچشی دارای چندین هایپرپارامتر هستند، مانند نرخ یادگیری، اندازه دسته، معماری شبکه و پارامترهای رگولاریزاسیون. انتخاب مقادیر مناسب برای این هایپرپارامترها می‌تواند به طرز چشمگیری بر عملکرد شبکه تأثیر بگذارد و یافتن مقادیر بهینه اغلب نیازمند آزمون و تنظیم است.

## سوال ۵

تا جایی که میشد داخل کد کامنت گذاشتیم و توضیح دادیم با این حال توضیحات کد به شرح زیر است:

بخش اول (preprocessing):

این کد یک سری کارهای مختلف را انجام می‌دهد. البته برخی از مواردی که در نظر گرفته شده است بر اساس توضیحات در کامنت‌ها بوده و با استفاده از برخی منابع خارجی مانند لینک‌ها، می‌توان توضیحات دقیق‌تری ارائه داد:

۱. بخش import و نصب بسته‌ها:

- بسته‌های مورد نیاز برای اجرای برنامه، از جملهnumpy ، pandas، shutil، tensorflow، zipfile، keras.backendو ...، وارد می‌شوند و نصب می‌شوند (دستوراتی که با علامت تعجب (!) آغاز شده‌اند دستورات لینوکسی هستند).
- ماژول‌ها و کلاس‌های مورد نیاز برای اجرای کد و استفاده از توابع خاص، با استفاده از دستورimport ، وارد برنامه می‌شوند.

۲. بخش تنظیمات اولیه:

- دستوراتی برای تغییر مسیر کاری (cd) به پوشه /content/ صادر می‌شوند.
- یک فایل با استفاده از شناسه gdown دانلود می‌شود و سپس با استفاده از دستورunzip ، فایل فشرده را در مسیر فعلی استخراج می‌کند.

۳. تابع:dataframe_creation

- تابعی است که با گرفتن مسیر پوشه تصاویر و نام، مسیر کامل تصاویر را در یک دیتافریم (DataFrame) ذخیره می‌کند.
- تابع به پوشه‌های مختلف راه می‌یابد و نام تصاویر را در هر پوشه استخراج می‌کند.
- سپس مسیرهای کامل تصاویر را در یک لیست ذخیره کرده و نام تصاویر را بدون مسیر و پسوند آن استخراج می‌کند.
- در نهایت، یک دیتافریم ایجاد می‌کند و مسیر کامل تصاویر را در آن قرار می‌دهد و ستون مربوط به مسیرها را با نام داده شده به تابع نامگذاری می‌کند. ستون شناسه تصاویر نیز به عنوان شناسه (index) دیتافریم تعیین می‌شود.

۴. تابع:display

- تابعی است که لیستی از تصاویر را به عنوان ورودی دریافت می‌کند و آن‌ها را به صورت ماتریس تصاویر نمایش می‌دهد.

- برای نمایش هر تصویر، از تابع **tf.keras.preprocessing.image.array_to_img**استفاده می‌کند.

۵. ایجاد پوشه‌ها:

- دستوراتی برای ایجاد دو پوشه به نام‌های train و train_masks در مسیر فعلی اجرا می‌شوند.

۶. ذخیره تصاویر:

- ابتدا لیستی خالی برای تصاویر و برچسب‌ها ایجاد می‌شود.
- سپس برای هر فایل در ساختار پوشه‌های موجود در مسیر داده شده، اگر فایلی با پسوند **label.bmp_**بود، آن را به لیست برچسب‌ها اضافه می‌کند و در غیر این صورت به لیست تصاویر اضافه می‌شود.
- سپس هر تصویر و برچسب با استفاده از کتابخانه PIL بازخوانی و به اندازه 256x256 تغییر اندازه می‌دهد و در قالب فایل PNG ذخیره می‌کند.
- در نهایت، تعداد تصاویر و برچسب‌های ذخیره شده را در پوشه‌های train و train_masksچاپ می‌کند.

۷. تابع dataframe_creation مجدد:

- تابع dataframe_creation را برای مسیرهای ذخیره شده در قسمت قبلی با نام‌های متفاوت فراخوانی می‌کند تا دیتافریم‌های متفاوتی برای تصاویر و برچسب‌ها ایجاد شود.
- سپس ستون مسیر برچسب‌ها را به دیتافریم تصاویر اضافه می‌کند.

۸. مرحله آماده‌سازی داده:

- تابعی به نام data_augmentation تعریف شده است که تصاویر را به صورت تصادفی افقی (چپ به راست) برگردانده و ماسک‌ها را نیز به همین صورت تغییر می‌دهد.
- تابعی به نام preprocessing نیز تعریف شده است که تصاویر را با استفاده از کتابخانه tf.ioبه صورت تنسور می‌خواند و سپس به اندازه ۲۵۶ x256تغییر اندازه می‌دهد و به نوع داده float32 و بین ۰ تا ۱ تبدیل می‌کند. ماسک‌ها نیز به همین صورت خوانده و تغییر اندازه می‌دهد.
- تابع create_dataset برای ساخت دیتاست از تصاویر و ماسک‌ها استفاده می‌شود. ابتدا از تابع tf.data.Dataset.from_tensor_slices برای ایجاد یک دیتاست از تنسورهای مسیر تصاویر و مسیر ماسک‌ها استفاده می‌شود. سپس با استفاده از تابع

mapدو تابع preprocessing و data_augmentation روی این دیتاست اعمال می‌شوند.

- در نهایت، داده‌ها به صورت بچ‌ها با BATCH_SIZE جمع‌آوری شده و دیتاست نهایی با استفاده از توابعcache ، shuffle، batchو prefetch آماده می‌شود. تعداد تکرارها نیز برابر با تعداد تصاویر در دیتافریم آموزش تعیین می‌شود.

۹. نمایش تصاویر:

- برای تصاویر آموزشی در دیتاست آموزش، تعدادی تصویر و ماسک گرفته می‌شود و با استفاده از تابع display به صورت تصویری نمایش داده می‌شوند.

بخش دوم(پیاده سازی مدل و آموزش آن):

در این بخش یک مدل U-Net را برای تشخیص ماسک در تصاویر آموزش می‌دهیم. دستورات زیر به طور کامل توضیح داده شده‌اند:

۱. ابتدا، وزن‌های قبلی آموزش داده شده برای مدل MobileNetV2 بارگذاری می‌شوند. این وزن‌ها برای انتقال یادگیری (Transfer Learning) استفاده می‌شوند. سپس شکل ورودی مدل به **img_size**تنظیم می‌شود و بخش بالای مدل (top) حذف می‌شود.

۲. به دلیل داشتن اتصالات از رده‌های پایین به رده‌های بالا در معماریU-Net ، لازم است از لایه‌های زیر به عنوان خروجی‌های مدل اصلی استفاده شود:

- block_1_expand_relu # 64x64
- block_3_expand_relu # 32x32
- block_6_expand_relu # 16x16
- block_13_expand_relu # 8x8
- block_16_project

۳. در این قسمت، لایه‌های مورد نیاز از مدل MobileNetV2 استخراج می‌شوند و در یک لیست ذخیره می‌شوند.

۴. مدل **down_stack**با ورودی مدل **base_model.input**و خروجی‌های مستخرج شده ساخته می‌شود. همچنین لازم است این بخش از مدل قابل آموزش نباشد.

۵. تابع **upsample**تعریف می‌شود که وظیفه آن انجام بخش دیکدر (Decoder) مدل U-Net است. این بخش برای هر بخش کاهش ابعاد در بخش انکودر یک بخش افزایش ابعاد متناظر در بخش دیکدر دارد. این لایه از تابع **Conv2DTranspose**استفاده می‌کند تا ابعاد فضایی تنسور

ورودی را دو برابر کند. سپس با استفاده از لایه **BatchNormalization**نرمال‌سازی انجام می‌شود.

۶. لیست **up_stack**شامل توابع **upsample**با پارامترهای مختلف است که برای ساخت بخش دیکدر مدل U-Net استفاده می‌شود.

۷. تابع **unet_model**مدل U-Net را با استفاده از بخش انکودر و دیکدر تعریف می‌کند. ابتدا یک لایه ورودی با ابعاد مناسب تعریف می‌شود. سپس تصاویر ورودی از طریق بخش انکودر می‌گذرند و خروجی‌های مربوطه استخراج می‌شوند. آخرین عنصر این لیست برای ورودی بخش دیکدر استفاده می‌شود و بقیه عناصر به ترتیب معکوس در لیست **skips**ذخیره می‌شوند. سپس با استفاده از لیست **up_stack**و **skips**، بخش دیکدر مدل ساخته می‌شود. در نهایت، با استفاده از لایه **Conv2DTranspose**نقشه نهایی با استفاده از ابعاد خروجی مدل تولید می‌شود.

۸. مدل با تابع **unet_model**و تعداد کانال‌های خروجی **OUTPUT_CHANNELS**تعریف می‌شود و با بهینه‌ساز **adam**و تابع خطای **dice_loss**کامپایل می‌شود. همچنین دو معیار **binary_accuracy**و **dice_coef**به عنوان معیارهای متریک در طول آموزش ثبت می‌شوند.

۹. سپس برای یک نمونه از داده‌های آموزشی، تصاویر و ماسک متناظر نمایش داده می‌شوند.

۱۰. توابع **visualize** و **show_predictions** برای نمایش تصاویر استفاده می‌شوند. این توابع تصاویر واقعی، ماسک و ماسک پیش‌بینی شده را نشان می‌دهند.

۱۱. سپس خروجی مدل و اطلاعات مربوط به آن نمایش داده می‌شوند.

۱۲. یک بازخوانی زودهنگام (Early Stopping) تعریف می‌شود تا از بیش‌برازش جلوگیری کند.

۱۳. در این بخش، یک کالبک (Callback) به نام **DisplayCallback**تعریف شده است که در طول آموزش، بازه‌های مشخصی که به تعداد اپوک بخصوصی می‌رسند، تصاویر و ماسک‌ها را نشان می‌دهد.

۱۴. مدل با تعداد اپوک‌های مشخص شده و با استفاده از داده‌های آموزشی و اعتبارسنجی آموزش داده می‌شود.

۱۵. سپس برای چند نمونه از داده‌های اعتبارسنجی، تصاویر و ماسک‌های متناظر نمایش داده می‌شوند.

۱۶. یک نمونه تصویر از داده‌های اعتبارسنجی گرفته می‌شود و ماسک پیش‌بینی شده برای آن محاسبه می‌شود.

۱۷. تابع **Liou**برای محاسبه ضریب تداخل متقاطع (Intersection over Union) تعریف شده است.

۱۸. ضریب تداخل متقاطع بین ماسک واقعی و ماسک پیش‌بینی شده محاسبه می‌شود و نتیجه چاپ می‌شود.

الف) کد زیر را برای این قسمت پیاده سازی کردم استفاده کردم:

```
def Liou(y_true, y_pred):
    # calculate the IoU loss with the following formula
    # Liou = 1 - ((∑H r=1 ∑W c=1 S(r, c)G(r, c)) / (∑H r=1 ∑W c=1[S(r, c) +
G(r, c) - S(r, c)G(r, c)]))
    y_true = tf.reshape(y_true, [-1])
    y_pred = tf.reshape(y_pred, [-1])
    intersection = tf.reduce_sum(y_true * y_pred)
    union = tf.reduce_sum(y_true) + tf.reduce_sum(y_pred) - intersection
    return 1 - (intersection / union)


# get one sample image
for image, mask in valid.take(1):
    sample_image, sample_mask = image, mask

# get the prediction
pred_mask = model.predict(sample_image[tf.newaxis, ...])
# pred_mask = pred_mask.reshape(img_size[0],img_size[1],1)
pred_mask = pred_mask.reshape(img_size[0],img_size[1],3)

# calculate the Loss IoU
Liou_sample = Liou(sample_mask, pred_mask)
# print("Liou: ", Liou_sample)*
print("Liou: ", Liou_sample.numpy())
```

توضیح کد:

تابع **Liou**تعریف شده در کد برای محاسبه ضریب تداخل متقاطع (Intersection over Union) با فرمول مشخص شده در مستند تمرین استفاده می‌شود.

در خط اول تابع، مقادیر واقعی **y_true**و پیش‌بینی شده **y_pred**را با استفاده از تابع **tf.reshape**به یک بعد تغییر شکل می‌دهیم. این کار به ما کمک می‌کند تا بتوانیم مقادیر ماسک‌ها را به شکل مناسب برای محاسبه استفاده کنیم.

در خط بعدی، مقدار تداخل بین دو ماسک را با ضرب نقطه‌ای **(*)** و سپس با استفاده از **tf.reduce_sum** محاسبه می‌کنیم. مقدار تداخل نشان می‌دهد که در چه میزان پیکسل‌های متناظر دو ماسک با یکدیگر همپوشانی دارند.

در خط بعدی، مقدار اجتماع بین ماسک واقعی و ماسک پیش‌بینی شده را محاسبه می‌کنیم. ابتدا مقادیر ماسک واقعی و پیش‌بینی شده را با استفاده از **tf.reduce_sum** جمع می‌کنیم، سپس از مقدار تداخل کاسته می‌شود تا مقدار اجتماع به دست آید.

در نهایت، با استفاده از فرمول **(intersection / union) - 1**، مقدار ضریب تداخل متقاطع محاسبه شده و به عنوان خروجی تابع بازگشت داده می‌شود.

در قسمت بعدی از کد، یک تصویر و ماسک نمونه از مجموعه داده **valid** گرفته می‌شود و در متغیرهای **sample_image** و **sample_mask** ذخیره می‌شود.

در خط بعد، با استفاده از مدل **model**، ماسک پیش‌بینی شده برای تصویر نمونه محاسبه می‌شود و در متغیر **pred_mask** ذخیره می‌شود. ابتدا متغیر **sample_image** را با **tf.newaxis** به یک بعد اضافه می‌کنیم تا شکل ورودی مدل را مناسب کنیم، سپس پیش‌بینی را با استفاده از **model.predict** انجام می‌دهیم. سپس با استفاده از **reshape**، شکل ماسک پیش‌بینی شده را به ابعاد مورد نظر تغییر می‌دهیم.

در خط بعد، با فراخوانی تابع **Liou**، ضریب تداخل متقاطع بین ماسک واقعی و ماسک پیش‌بینی شده محاسبه می‌شود و در متغیر **Liou_sample** ذخیره می‌شود.

در نهایت، مقدار ضریب تداخل متقاطع (**Liou_sample**) با استفاده از **()numpy** چاپ می‌شود.

ب)

Sources:
https://www.scitepress.org/Papers/2019/73475/73475.pdf
https://towardsdatascience.com/how-accurate-is-image-segmentation-dd448f896388
https://medium.com/analytics-vidhya/different-iou-losses-for-faster-and-accurate-object-detection-3345781e0bf

BCE loss (Binary Cross-Entropy) و IoU loss (Intersection over Union) دو تابع مختلف ضرر(loss function) هستند که در یادگیری ماشین استفاده می شوند.

BCE loss یک تابع ضرر است که برای مسائل طبقه بندی باینری( binary classification problems) استفاده می شود. عدم تشابه بین احتمالات پیش بینی شده و برچسب های کلاس واقعی را اندازه گیری می کند( measures the dissimilarity between the predicted probabilities and the true class labels). BCE loss به عنوان احتمال ورود منفی برچسب های کلاس واقعی با توجه به احتمالات پیش بینی شده تعریف می شود( negative log-likelihood of the true class labels given the predicted probabilities). معمولاً در ترکیب با یک تابع فعال سازی سیگموئید(sigmoid) در لایه خروجی یک مدل طبقه بندی باینری(binary classification model) استفاده می شود.

از سوی دیگر، IoU loss یک تابع ضرر است که برای مسائل تشخیص و ناحیه بندی اشیا( object detection and segmentation problems) استفاده می شود. همپوشانی بین دو جعبه مرزی یا ماسک ناحیه بندی(overlap between two bounding boxes or segmentation masks) را اندازه‌گیری می‌کند. IoU به عنوان نسبت سطح اشتراک به منطقه اجتماع دو جعبه یا ماسک مرزی ( ratio of the intersection area to the union area of the two bounding boxes or masks) تعریف می شود. IoU loss معمولاً برای آموزش مدل‌های تشخیص یا ناحیه بندی اشیا (object detection or segmentation models) برای بومی‌سازی دقیق اشیاء در یک تصویر (accurately localize objects) استفاده می‌شود.

به طور خلاصه، BCE loss و IoU loss برای انواع مختلف مسائل استفاده می شود و جنبه های مختلف عملکرد یک مدل را اندازه گیری می کند. BCE loss برای مسائل طبقه‌بندی باینری استفاده می‌شود و عدم تشابه بین احتمالات پیش‌بینی‌شده و برچسب‌های کلاس واقعی را اندازه‌گیری می‌کند، در حالی که از IoU loss برای مشکلات تشخیص و ناحیه بندی اشیا استفاده می‌شود و همپوشانی بین جعبه‌ها یا ماسک‌های مرزی پیش‌بینی‌شده و واقعی را اندازه‌گیری می‌کند.

به بیان دیگر:

Binary Cross Entropy (BCE) loss and Intersection over Union (IoU) loss are two different loss functions that can be used in binary image segmentation.

BCE loss is a common loss function used for binary classification problems. It measures the dissimilarity between the predicted probability distribution and the true distribution. In the context of binary image segmentation, BCE loss measures the pixel-wise error between the predicted segmentation mask and the ground truth mask.

On the other hand, IoU loss is a measure of overlap between two masks. It is calculated as the ratio of the intersection area to the union area of the predicted mask and the ground truth mask. IoU loss is commonly used in object detection and segmentation tasks to measure how well the predicted bounding box or segmentation mask overlaps with the ground truth.

A comparison between IoU loss and BCE loss has been made by testing two deep neural network models on multiple datasets and data splits. The results show that training directly on IoU significantly increases performance for both models compared to training on conventional BCE loss[1].

In summary, BCE loss measures pixel-wise error while IoU loss measures overlap between masks. Both can be used in binary image segmentation tasks, but IoU loss has been shown to perform better in some cases.

---

BCE (Binary Cross-Entropy) loss and IoU (Intersection over Union) loss are two different loss functions commonly used in different types of tasks.

1. Binary Cross-Entropy (BCE) Loss:
    - BCE loss is primarily used for binary classification problems.
    - It measures the dissimilarity between the predicted probabilities and the true binary labels.

- BCE loss is calculated using the formula: -[y * log(y_pred) + (1-y) * log(1-y_pred)], where y is the true binary label and y_pred is the predicted probability.

2. Intersection over Union (IoU) Loss:
    - IoU loss is often used for tasks related to object detection or semantic segmentation.
    - It measures the similarity or overlap between the predicted bounding box or segmentation mask and the ground truth.
    - IoU is calculated by dividing the intersection area between the predicted and ground truth regions by the union area.
    - IoU loss is computed as 1 - IoU.

The main difference between BCE loss and IoU loss lies in their applications and the types of tasks they are designed for:

- BCE loss is suitable for binary classification tasks where the goal is to predict the probability of a binary event.
- IoU loss, on the other hand, is commonly used in tasks like object detection or semantic segmentation, where the focus is on measuring the overlap between regions or masks.

It's important to note that while BCE loss can be used in some cases for tasks related to object detection or segmentation, it doesn't directly capture the concept of overlap or similarity between regions. IoU loss, specifically designed for such tasks, provides a more direct optimization objective by encouraging predictions that closely align with the ground truth regions.

پایان