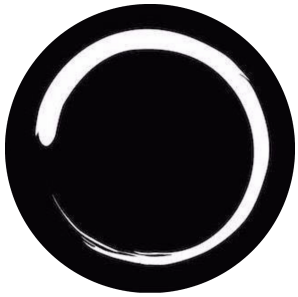
inzva DLSG

Convolutional Neural Networks

Week5 & Week6



Contents

1 Introduction to Convolutional Neural Networks

Convolution Neural Networks, also known as CNNs, are type of neural networks generally used for processing data that has 2D grid such as images. However, they can be also used with 1D grid like audio and time series. They are called CNNs because they employ the convolution operation, which we will look into in the following sections.

Up to this point, we have utilized Fully Connected Networks (FCNs) to do some tasks, but it is worth exploring why CNNs are preferred for image data instead of FCNs. Firstly, images are high dimensional. Hidden layers in FCNs generally larger than the input size and this leads to significant increase in the number of weights. This would be a problem in terms of memory and computation. Secondly, CNNs have the concept of local connectivity and parameter sharing. Local connectivity implies that each neuron is connected to small chunk of image aka receptive field. This is in contrast to FCNs, where all the neurons are fully connected. Parameter sharing allows the network to efficiently learn and detect patterns in images by using the same set of parameters across different regions. We will delve deeper into this concept in the following sections.

In this chapter, we will describe the fundamentals of CNNs, such as the convolution operation, parameter sharing, strided convolution, padding, pooling and upsampling.

1.1 Convolution Operation

Convolution operation is a mathematical operation that combines two function to produce the third function. In CNNs, the first argument to the convolution is referred to as the **input**, and the second argument is known as the **convolutional kernel**

or **filter**. The kernel is used to extract features from the images. The output of this convolutional operation is referred to as the **feature map**. See Figure 1 for an example of convolution operation in 2D grid.

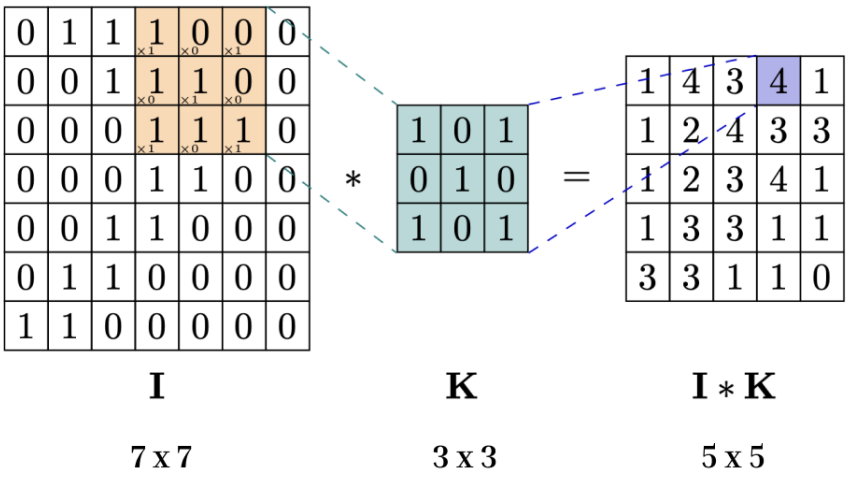


Figure 1: Convolution Operation [11]

A 3×3 kernel is applied to the image. It calculates weighted sum for every 3×3 square area. The weighted sum is nothing but multiplying each element of the kernel with the corresponding element of the 3×3 square area of the input image and then summing up these products. The kernel moves horizontally and vertically across the entire image to create output at each position. A general rule states that when a matrix of size $n \times n$ is convolved with a kernel of size $f \times f$, it gives us matrix of size $(n - f + 1) \times (n - f + 1)$. However, we will modify this rule a bit in the following sections. As illustrated in Figure 1, when we apply a convolutional kernel of size 3×3 to a 7×7 image, we obtain an output size of 5×5, which is calculated as follows: $7 - 3 + 1 = 5$.

1.2 Parameter Sharing

In traditional neural networks such as FCNs, there exists interaction between every input unit and every output unit. However, CNNs have sparse interactions thanks to utilizing smaller kernels compared to the input. This means that computing the output requires less memory.

Parameter sharing refers to using the same set of parameters across all the input. To express it a bit more clearly, in a CNN, each element of the kernel is used at every position of the input. The utilization of parameter sharing in the convolution operation means that rather than learning a separate set of parameters for every location, we learn only one set. During image processing, it is useful to detect edges in the early layers since similar edges exist throughout the image. Therefore, it is good to share parameters across the image.

If we look at the Figure 2, we applied convolution operations on the left image by using just a convolution kernel. This kernel is used to extract vertically oriented edges within the input image.

Figure 2: Edge Detection

1.3 Strided Convolution

Stride is a hyperparameter of CNNs that determines the extent of movement over the image. If we look at the Convolution Operation section, the example illustrates that the kernel shifts one by one. It means that stride is 1. With the usage of stride, we change our rule a bit. When a matrix of size $n \times n$ is convolved with a kernel of size $f \times f$ and stride s, it gives us matrix of size $((n-f)/s)+1 \times ((n-f)/s)+1$.

Figure 3 illustrates the convolution with stride of 2. Also, the diagram shows how the output size changes when we adjust the stride value.

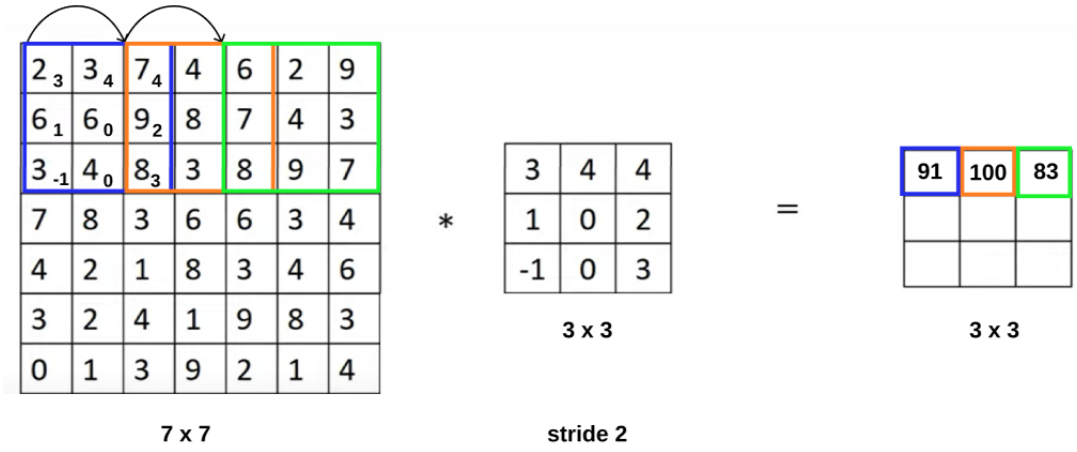


Figure 3: Strided Convolution

1.4 Padding

As you will be able to see in the previous sections, when we apply the kernel to the image, the size of the image decreases. We sometimes need to avoid this because we do not want to lose information. Additionally, pixels at the edges receive fewer convolutions compared to those at the center.

Padding ensures that the size of the input remains same across the CNN layers. The most commonly used padding techniques are zero padding and valid padding. Zero padding, a.k.a. same padding, adds zeros outside of the image. On the other hand, valid padding means that no additional padding is added when employing the

convolution operation. With the usage of padding, we modify our rule slightly. When a matrix of size $n \times n$ is convolved with a kernel of size $f \times f$ and using stride s and padding p, it gives us matrix of size $((n + 2p - f)/s) + 1 \times ((n + 2p - f)/s) + 1$.

You can look at the Figure 4 for the same padding operation.

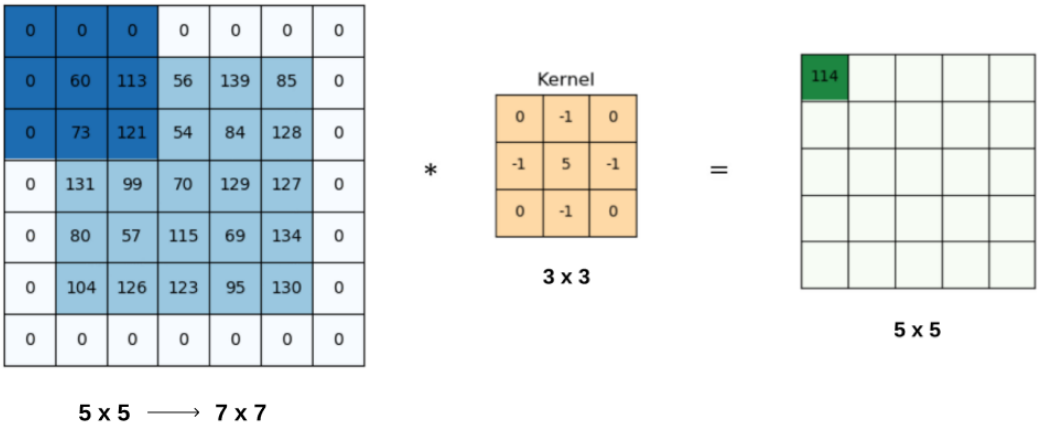


Figure 4: Padding [3]

1.5 Pooling

Pooling is a technique in CNNs for reducing size while preserving important information. The pooling process summarizes the information over a whole feature map. The crucial part is that the size of the pooling filter must be smaller than the size of the feature map. There are some common pooling techniques in CNNs such as max pooling and average pooling. While max pooling operation retrieves the biggest number in the filter area, average pooling calculates the average of the numbers within the filter area. You can look at the Figure 5 for max pooling and average pooling.

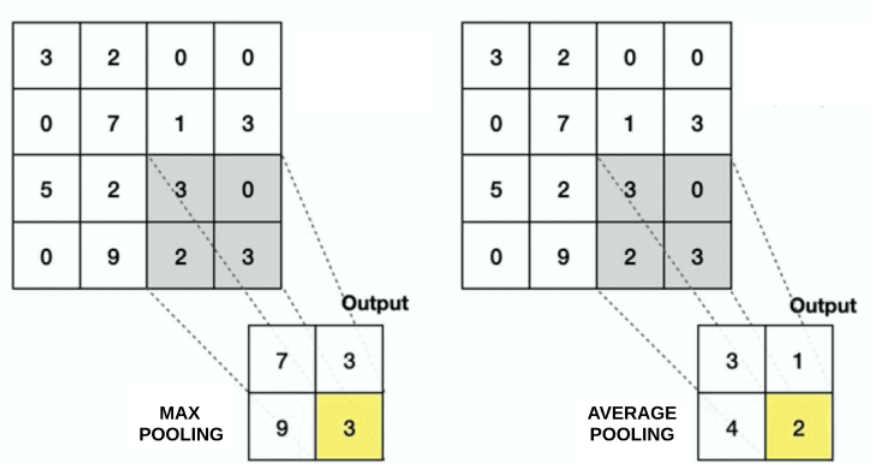


Figure 5: Max and Average Pooling [9]

1.6 Upsampling

In certain tasks such as semantic segmentation or image generation, we want the output to be same size of the input. It can be achieved using upsampling techniques. There are lots of ways to scale up the network.

The first way to scale up the network is to duplicate the each input four times, known as nearest neighbor technique. You can examine this technique on the left of Figure xxx. The second way is max-unpooling that redistribute the maximum value(s) back to their original position(s) and fills the rest with zeros. On the right of Figure 6, you can see the max-unpooling technique.

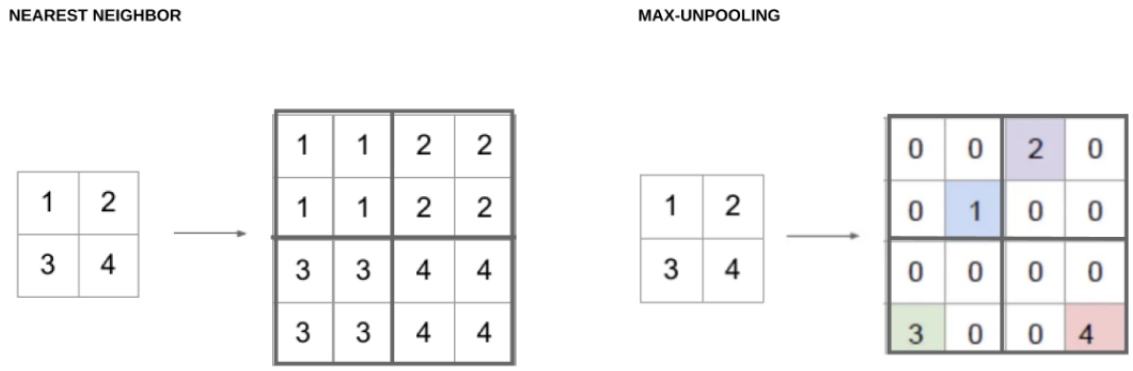


Figure 6: Nearest Neighbour and Max-unpooling [7]

Another technique is transpose convolution. In this technique, we utilize convolutional kernel for upsampling. Suppose that we are given a $n \times n$ input and a $f \times f$ kernel. Every value in the input is multiplied with the kernel. Additionally, the summation occurs where the output overlaps. When we look into the Figure 7, we have 2×2 input and 2×2 kernel. The areas where output overlaps are summed.

2 Image Classification Using CNNs

Image classification stands out as one of the most renowned task among computer vision tasks. It is the task of assigning a class or label for an entire image. While the task may seem straightforward, it can be challenging if there are large number of classes.

In this chapter, we will delve into numerous CNN models designed for image classification.

2.1 LeNet

LeNet is one of the earliest CNN architecture designed by Yann LeCun for classifying 28×28 grayscale images of handwritten digits, introduced in 1998. Nowadays, the dataset known as MNIST.

The overall architecture of LeNet is relatively straightforward. LeNet consists of 7 layers, including 2 convolutional layers, 2 pooling layers, and 3 fully connected layers. In convolutional layers, the size of the convolutional kernels is 5×5. Despite using 5×5 convolutional kernels in the first convolutional layer, the size of the feature map remains unchanged due to using padding of 2. On the other hand, the size of the

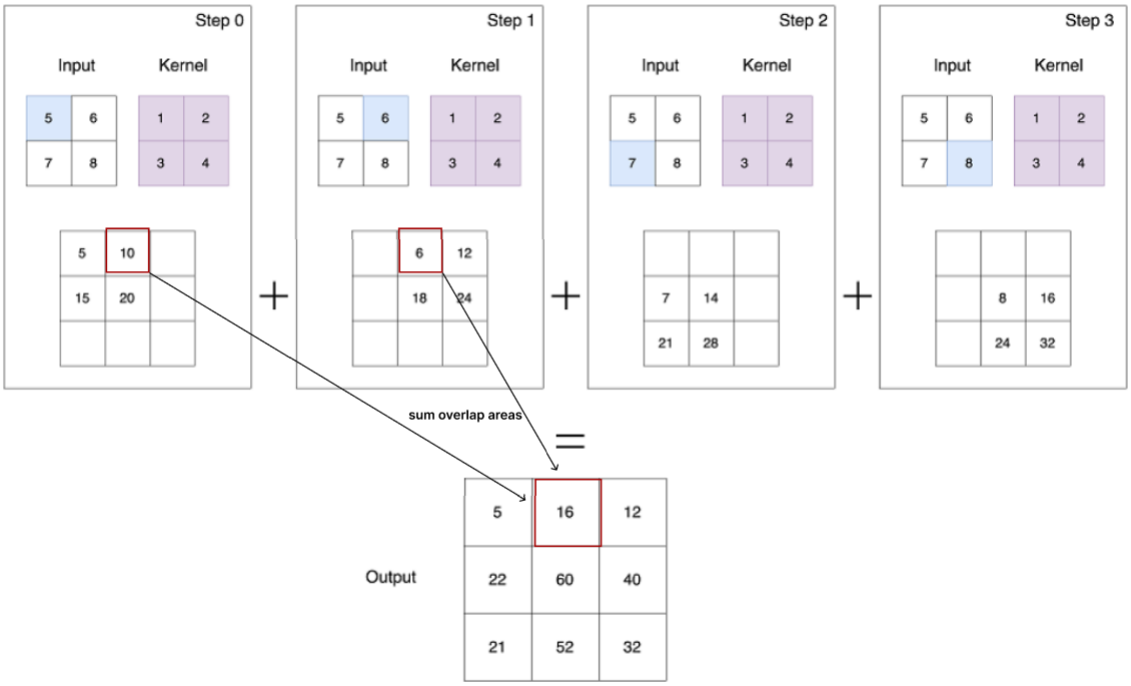


Figure 7: Transpose Convolution [4]

feature map in the second convolutional layer decreases from 14×14 to 10×10 because no padding is applied. It is used Tanh as an activation function. The architecture can be examined in the Figure 8.

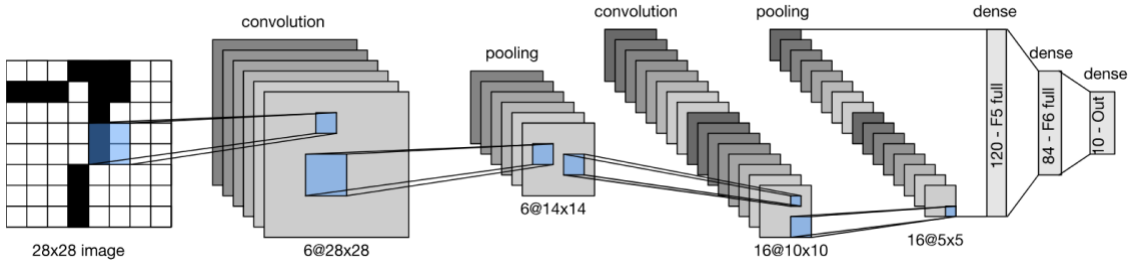


Figure 8: LeNet Architecture [14]

2.2 AlexNet

AlexNet is the architecture that won ImageNet Large Scale Visual Recognition Challenge (ILSVRC), designed by Alex Krizhevsky in 2012. ImageNet is a large-scale database containing over 14 million images, which has been a benchmark for evaluating the performance of architectures in the field of image classification and object detection. As you can notice in the Figure 9, there are some innovations in the AlexNet architecture. AlexNet consists of 11 layers, including five convolutional layers, three pooling layers, two fully-connected layers, and a softmax classifier output layer. If you look into AlexNet paper, you see that the input size is $224 \times 224 \times 3$, which is a mistake. The input size should be $227 \times 227 \times 3$. ReLU is used as an activation function. Additionally, in the paper, the authors stated that they used the data augmentation techniques and dropout layers to reduce the overfitting.

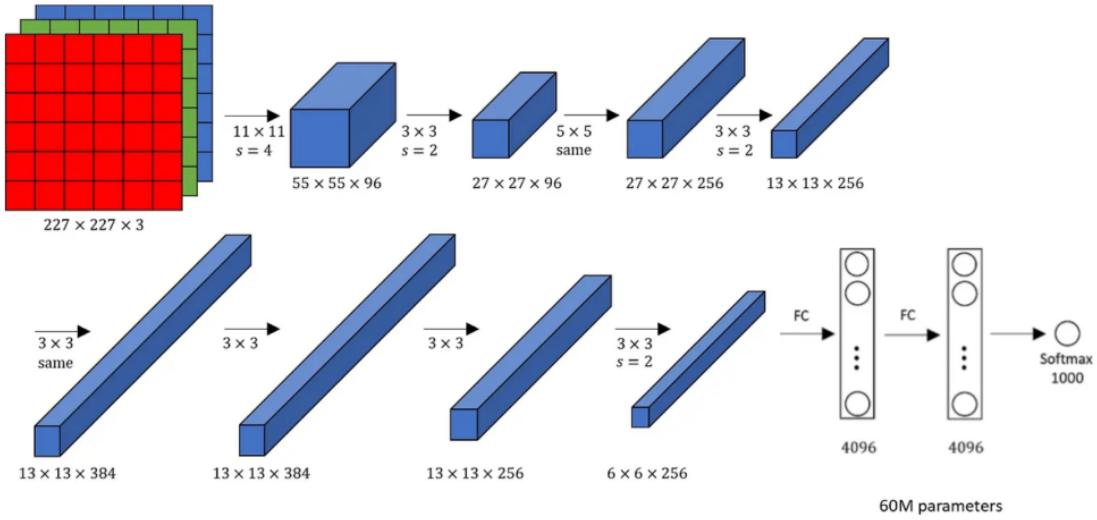


Figure 9: AlexNet Architecture [1]

2.3 VGGNet

VGGNet is another CNN architecture that consists of 13 convolutional layers and 5 max pooling layers. These are followed by 3 fully connected layers. You can look at the Figure 10 for the architecture. The authors released a series of VGG models with different layer lengths, from 11 to 19. Each VGG model shares common characteristics within its VGG block. The same kernel size 3×3 is applied to all convolutional layers, while a pooling size of 2×2 is applied across all pooling layers. This VGG model achieved 92.7% classification performance in the ILSVRC-2014 competition.

Figure 10: VGGNet Architecture [13]

2.4 ResNet

ResNet which stands for Residual Network is the first convolutional network to utilize residual blocks. As we transition from LeNet to VGGNet, we observe that networks are becoming deeper. In theory, adding more layers should lead to higher accuracy. However, it is observed that deeper networks leads to degradation problem due to vanishing/exploding gradients. This problem results in accuracy degradation and higher training error. To address this issue, the authors introduce a residual block. A residual block is the main component of the ResNet. There exist a concept called

shortcut connection or skip connection in a residual block. Shortcut connections skips some of the layers in the neural network. As you see in the Figure 11, this shortcut connection performs identity mapping by skipping some layers. The formulation of $F(x) + x$ refers to add the outputs of the stacked layers to the identity mapping. ResNet architecture consists of these residual blocks.

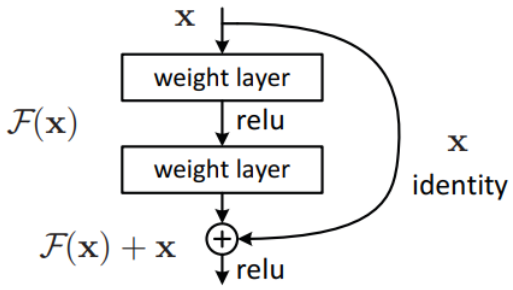


Figure 11: Residual Block [6]

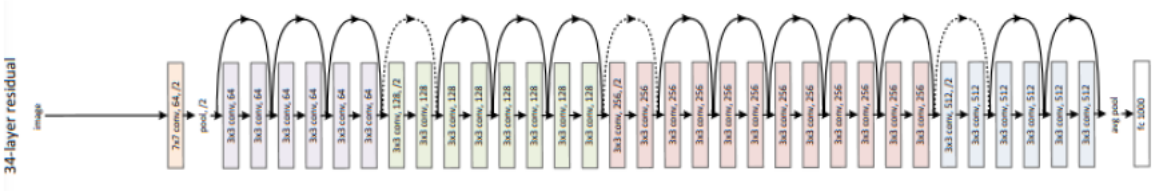


Figure 12: 34 Layer ResNet [6]

If you look into the Figure 12, you will notice that some shortcuts are represented by dotted lines while others are solid lines. In residual blocks, we can add the identity x to the stacked result if only the dimensions match. The dotted lines only come up at some parts where the depths do not match. It is performed a linear projection in the dotted lines to match the dimensions. The linear projection can be explained by performing 1×1 convolution to match the channel sizes while preserving the spatial dimensions.

2.5 Inception

So far, we have examined some CNN architectures. These networks stack CNN layers to delve deeper. We choose a single kernel size for a convolution operation. However, determining the right kernel size can be challenging. That is why kernels with multiple sizes are employed at the same level. If you look at the inception module shown in Figure 13, there are multiple kernels with multiple sizes, and their outputs are concatenated. This inception module output sent to another inception module input. Inception network consists of lots of inception modules. One big problem with the modules is that using 5×5 convolution expensive with a large number of filters. If you notice the 1×1 convolutions preceding the 3x3 and 5x5 convolutions, they serve to handle this problem. These 1x1 convolutions reduce computation by performing dimensionality reduction. There are various inception networks, and one of the most well-known is GoogLeNet.

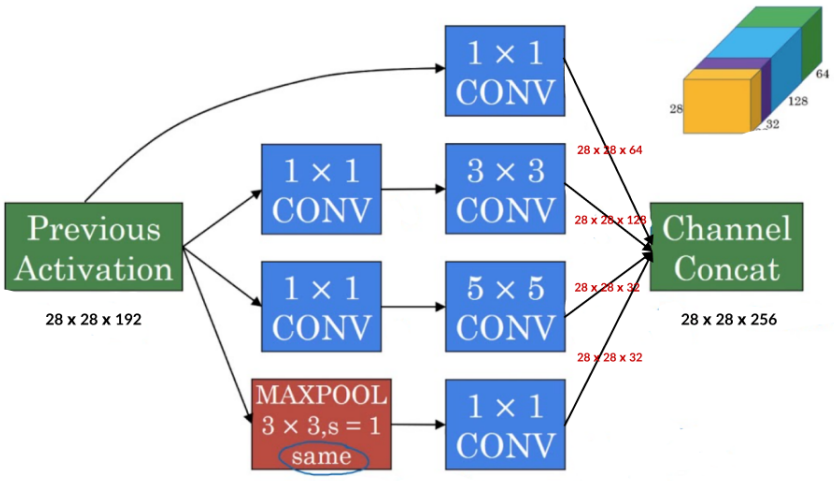


Figure 13: Inception Module [8]

3 Image Segmentation Using CNNs

Image segmentation is another computer vision task that categorize each pixel in an image. While image classification assigns a single label to an entire image, image segmentation assigns a label to every pixel within the image. Image segmentation task has multiple types such as semantic segmentation, instance segmentation and panoptic segmentation. Semantic segmentation assigns a same unique label to each of the object. For example, if we look at the Figure 14 b, there are 4 cars and all of them are represented with same color. Instance segmentation assigns a unique id for each individual object instance. As you can see in the Figure 14 c, car is a object but every car has a different color. Lastly, panoptic segmentation combines both semantic segmentation and instance segmentation.

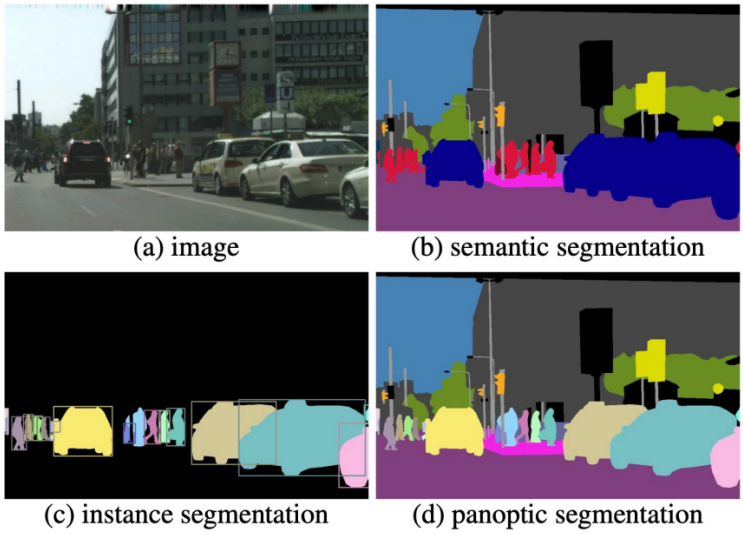


Figure 14: Segmentation types

There are numerous architectures available for the segmentation task. In this chapter, we will explore U-Net, one of the most popular architecture for the semantic segmentation task.

3.1 U-Net

U-Net is a well-known deep learning architecture used for semantic segmentation, introduced by Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-Net originally introduced for biomedical image segmentation task but has been widely used in other image segmentation tasks. Prior to the development of U-Net, the sliding window approach was used for image segmentation tasks. This approach takes patches from image and processes each patch individually using a CNN architecture. However, this technique has a significant drawback. CNN must be executed seperately for each patch anf this results in inefficient processing times. On the other hand, U-Net is much faster than sliding window approach.

As you can see in the Figure 15, U-Net consists of two main components: an encoder and a decoder. In the paper, the authors referred to the encoder part as the "contracting path" and the decoder part as the "expanding path".

The encoder part is a typical convolutional network architecture. It consists of two 3×3 convolutions, followed by ReLU and a 2×2 max pooling with stride of 2. At each downsampling step the number of channels is doubled. In the decoder part, every step consists of 2×2 up-convolution and two 3×3 convolutions followed by ReLU. At the final layer a 1x1 convolution is used to map the feature vector to the desired number of classes. In the paper, data augmentation technique is used.

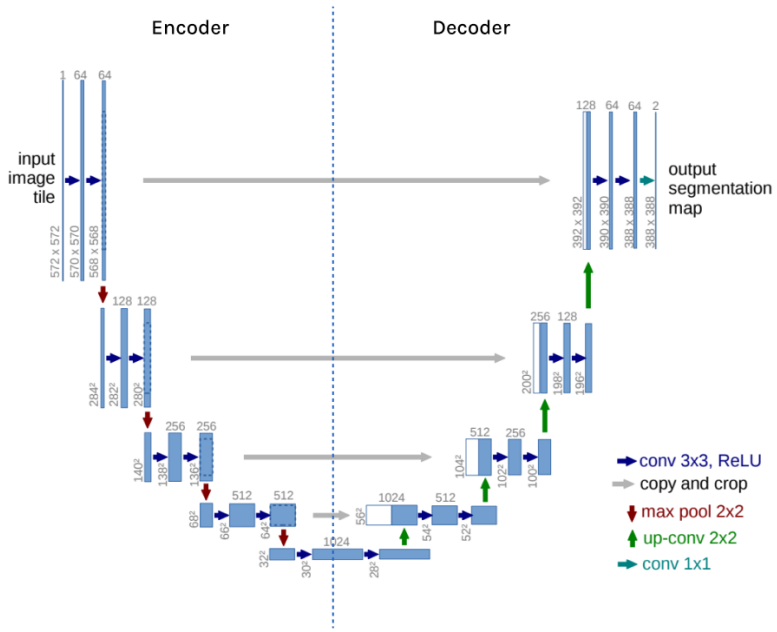


Figure 15: U-Net architecture [12]

4 Object Detection Using CNNs

Object detection is a term that detecting objects in a given image or video. It combines two different tasks: image classification and object localization. We've already examined what image classification is. On the other hand, object localization refers to finding the location of the object and draw a bounding box around them. The goal of object detection is to identify and localize the objects within an image. Generally, object detection algorithms can be divided into two main categories: two-shot detection and single-shot detection. Two-shot detection algorithms have two stages:

region proposal and classification. The first determines the potential regions of interest and, the second stage classifies these regions. On the other hand, single-shot detection algorithms handle this process in a single pass. However, there are advantages and disadvantages between these algorithms. While two-shot object detection algorithms such as R-CNN (Region-Based Convolutional Neural Networks) or Fast R-CNN (Fast Region-Based Convolutional Neural Networks) are known for more accurate predictions on small objects, single-shot object detection algorithms such as YOLO (You Only Look Once) are known for their speed. Consequently, single-shot object detection is preferred for real-time problems. In this chapter, we will explore YOLO, one of the most prominent single-shot object detection algorithms.

4.1 YOLO (You Only Look Once)

YOLO is a state-of-the-art algorithm that first introduced in 2015 with the "You Only Look Once: Unified, Real-Time Object Detection" paper [10].

4.1.1 Evaluation Metrics

Before we delve into the YOLO, it is important to first look into some main evaluation metrics. One of the most popular evaluation metrics is IoU (Intersection over Union), also known as Jaccard index. It measures how well the predicted bounding box align with the ground truth box. It is calculated by dividing intersection of the boxes into union of the boxes. You can take a look at the Figure 16. The IoU value ranges between 0 and 1, where a value closer to the 1 means higher alignment.

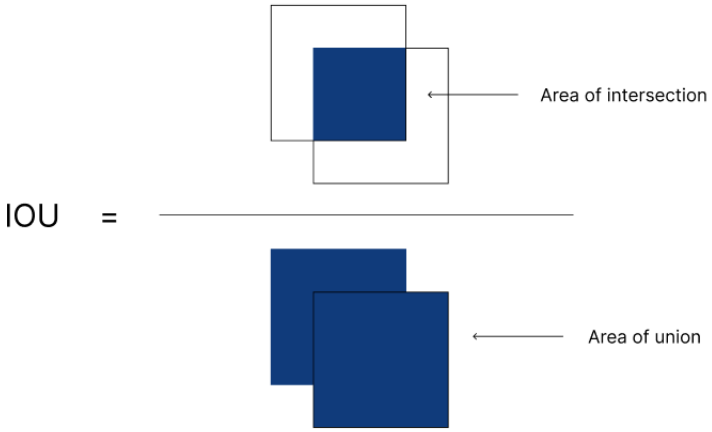


Figure 16: Intersection over Union

The other popular evaluation metric is mAP (mean Average Precision). However, to understand the mAP, we first need to understand confusion matrix, recall, precision and AP (Average Precision) terms.

Confusion matrix is a table that gives some information about a model performance. If you look at the Figure 17, you can see some terms in the table. True positive means that IoU score between the predicted bounding box and ground truth bounding box passes the threshold value. It represents the correct predictions. False positive refers that your predicted bounding box is incorrect. False negative means that there is a ground truth bounding box but it is missed (no predicted bounding box). True negative represents the background and this value is not used for evaluation. On the other hand, the precision value tells how many of the model's predictions

are correct, while the recall tells out of all the ground truth values how many of them the model predicted correctly. Precision and recall values can be calculated as shown in the figure.

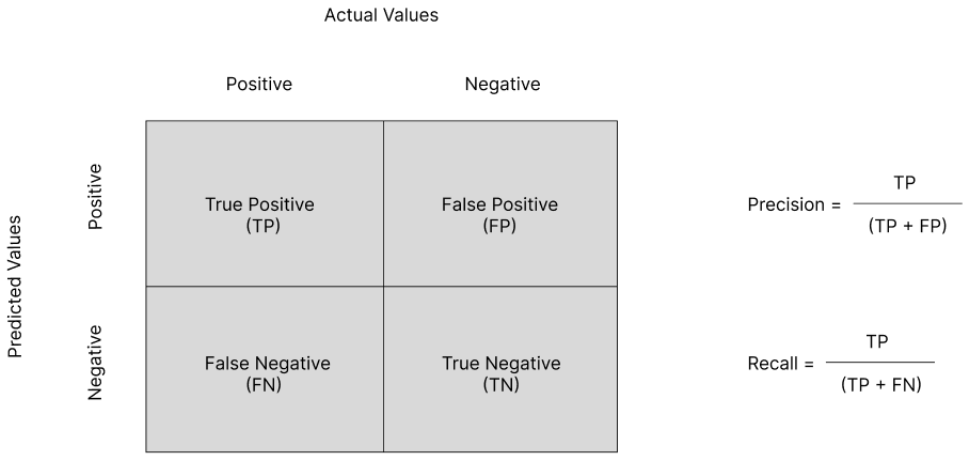


Figure 17: Confusion Matrix

When these precision and recall values was calculated, it can be drawn precision-recall curve. Under the area of this precision-recall curve (AUC) refers to the AP. The AP is calculated for each class and, mAP basically refers the mean of the AP of all classes. You can look at the Figure 18 for the formula.

$$mAP = \frac{1}{n}\sum_{k=1}^{k=n} AP_k$$
$$AP_k = \textit{ the AP of class k}$$
$$n = \textit{ the number of classes}$$

Figure 18: Mean average precision [5]

4.1.2 YOLO Architecture

YOLO uses a convolutional neural network to detect objects in a given image. The end-to-end single network processes images and predicts bounding boxes and class probabilities in a single evaluation.

YOLO takes the input image and resize it to 448×448. Afterward, it divides the input image into $S \times S$ grid cells and S set to 7 in the paper. Each cell is responsible for predicting one object. If the center of an object falls into a grid cell, that grid cell is responsible for predicting the object. Each grid cell predicts B bounding boxes and the a confidence score for each box. The B value is 2 in the paper. Each bounding box comprises of 5 predictions: x, y, w, h and confidence score. The x and y values are calculated with respect to the top-left corner of each individual grid cell. The w (width) and h (height) values are calculated the proportion of the image dimensions. All x, y, w, and h values are between 0 and 1. You can see all of these in the Figure 19. In spite of the fact that we have B bounding boxes, we only predicts one set of class probabilities (let's say we have C class) for each grid cell. This means that we have B bounding boxes for each grid but only one set of class probabilities for that

grid. These all predictions encoded to a $S \times S \times (B \cdot 5 + \mathrm{C})$ tensor. In the paper, the authors use S=7, B = 2 and C = 20, which results in a $7 \times 7 \times 30$ tensor.

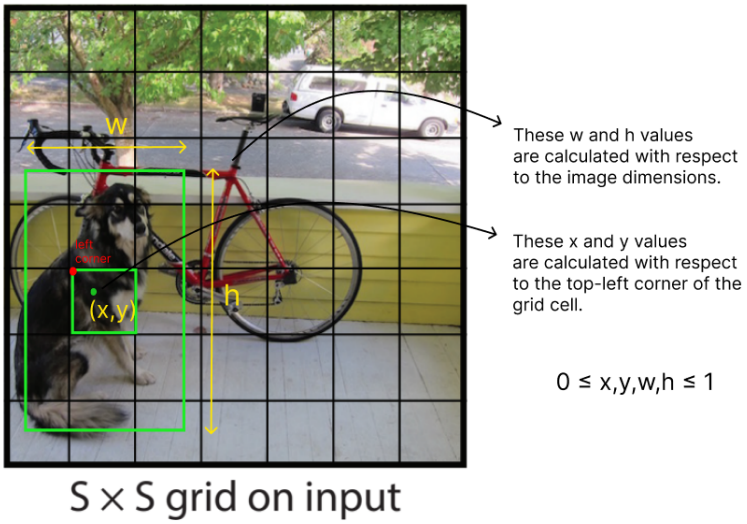


Figure 19: $S \times S$ grid [10]

To go from the input image to these outputs, a CNN is used. We have already mentioned that the overall structure is a CNN architecture, as shown in the Figure 20. The final layer predicts both class probabilities and bounding box coordinates.

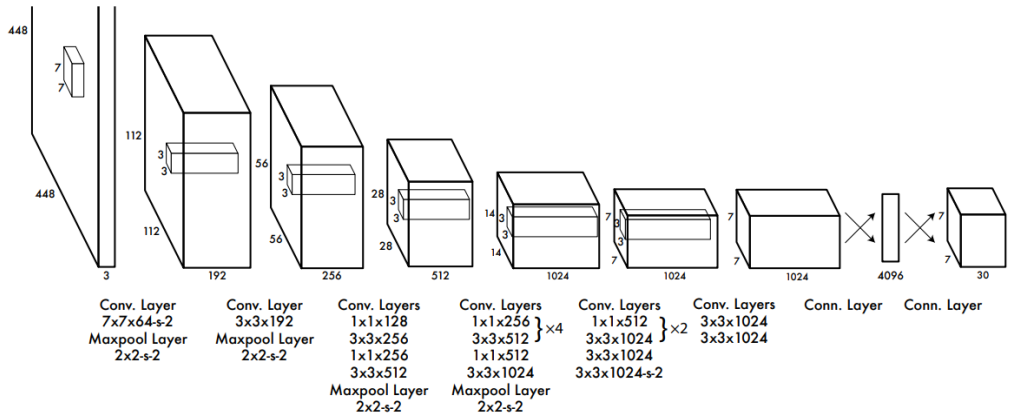


Figure 20: YOLO architecture [10]

During training process, each grid cell can produce multiple bounding boxes. However, one of the bounding boxes needs to be chosen to calculate the loss. This choice is made according to the IoU scores between the each bounding box and the ground truth. The bounding box with the highest IoU score is chosen. This process is known as non-max suppression. When we comes to the loss function, it mainly consists of three losses: bounding box loss, confidence loss and class probability loss. You can see the overall loss in Figure 21. It can seem a bit confusing but we will explain the loss.

As you can see, there are two parameters in the loss: λ_{coord} and λ_{noobj}. We already know that the majority of the grid cells do not contain an object. This means that confidence scores of these cells will be near 0. If we give equal weights to all the grid cells, the loss of no-object grid cells dominates the others. To address this issue, we

$$
\begin{aligned}
&\lambda_{\textbf{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right] \\
&+ \lambda_{\textbf{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right] \\
&+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left(C_i - \hat{C}_i \right)^2 \\
&+ \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} \left(C_i - \hat{C}_i \right)^2 \\
&+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{\text{obj}} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2
\end{aligned}
$$

Figure 21: YOLO architecture [10]

use the parameters. In the paper, authors set $\lambda_{coord} = 5$ and $\lambda_{noobj} = 0.5$, increasing the weight of detection and decreasing the importance of no-object loss.

The first two terms of the total loss accounts for the bounding box loss. It is the sum of the squared error between the predicted bounding box and the ground truth bounding box. The next two terms refers the confidence loss and the last term is for the object class loss.

We examined the first version of YOLO here, which was released in 2015. However, many different versions of YOLO have been released over the years. You can look at the Figure 22 to see the versions and their improvements.

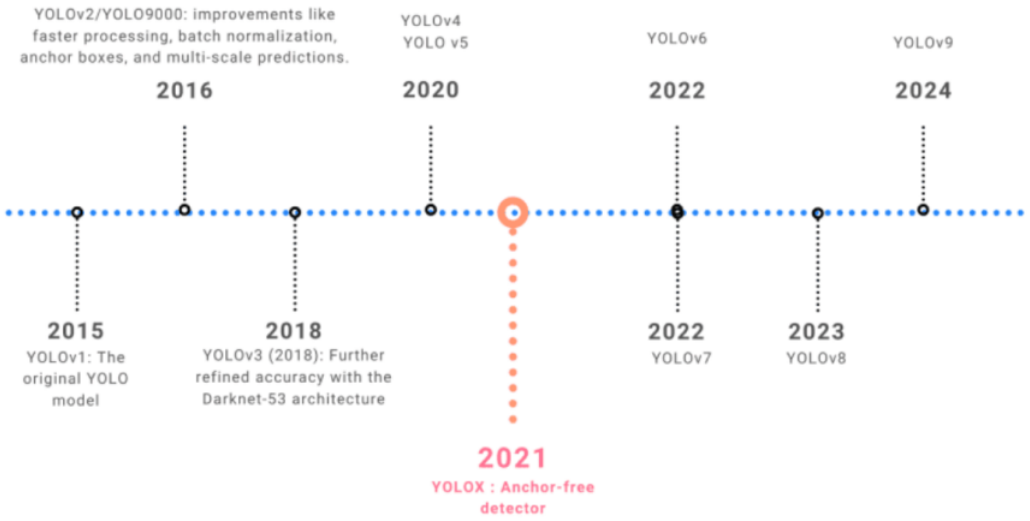


Figure 22: YOLO architecture [2]

References

[1] Bangar, S. Alexnet architecture explained, 2022. Accessed: 2024-05-22.

[2] Boesch, G. Yolox explained: Features, architecture and applications.

[3] Dharmaraj. Zero-padding in convolutional neural networks, 2021. Accessed: 2024-05-18.

[4] et al., K. C. Math and architectures of deep learning, 2024. Accessed: 2024-05-19.

[5] Gad, A. F. Evaluating object detection models using mean average precision (map).

[6] He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2015), 770–778.

[7] Jun. Unsampling: Unpooling and transpose convolution, 2020. Accessed: 2024-05-19.

[8] NG, A. C4w2l07 inception network.

[9] Qayyum, R. Introduction to pooling layers in cnns, 2022. Accessed: 2024-05-17.

[10] Redmon, J., Divvala, S. K., Girshick, R. B., and Farhadi, A. You only look once: Unified, real-time object detection. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2015), 779–788.

[11] Riebesell, J. Convolution operator, 2021. Accessed: 2024-05-15.

[12] Ronneberger, O., Fischer, P., and Brox, T. U-net: Convolutional networks for biomedical image segmentation. *ArXiv abs/1505.04597* (2015).

[13] Varshney, P. Vggnet-16 architecture: A complete guide, 2020. Accessed: 2024-06-06.

[14] Zhang, A., Lipton, Z. C., Li, M., and Smola, A. J. Dive into deep learning, 2023.