**CS 282 Deep Neural Networks** **February 24th, Spring 2023**

# Lecture 10: CNNs and Topics on Computer Vision

*Lecturer: Anant Sahai* *Scribe: James Xu, Joohwan Seo*

## 10.1 ResNet: Continued

Recall that instead of a conventional convolutional neural network (CNN) block, a residual convolution neural network (ResNet) architecture [1] has a feed-forward signal which skips the CNN block and directly affects the outputs. Note that because of this feed-forward structure, the gradients of the weights are affected in the back-propagation step (Fig. 10.1). In particular, the back-propagation signal of ResNet is added by identity. Therefore, even when the gradient flow is small, the gradient flows of ResNet are added by identity, thus allowing continuous training. Without summing identity, the training processes would suffer from a slow learning rate. As mentioned in the previous lecture, the outputs of the ResNet can be considered as some discretization of the ordinary differential equation (ODE), which leads to the rise of neural ODE theory.

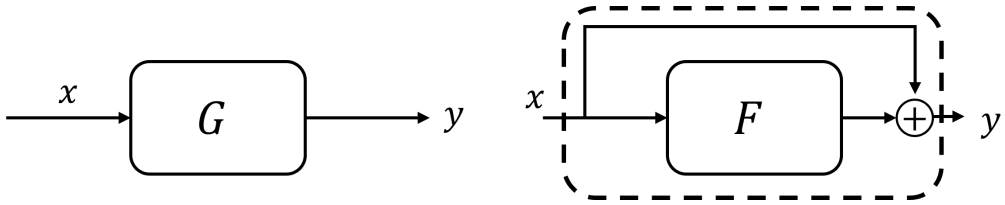


Figure 10.1: Block diagram consisted of basic building blocks. $G$ is the building block of basic CNN (left), and $F$ is the basic building block of ResNet (right). The inputs and outputs are denoted by $x$ and $y$, respectively. (Figure from lecture, drawn by ourselves)

### 10.1.1 The architecture of ResNet building block

We have a natural question - what should we put into the $F$ block? In order to answer this question, we first compare two types of building blocks: Fig. 10.2.

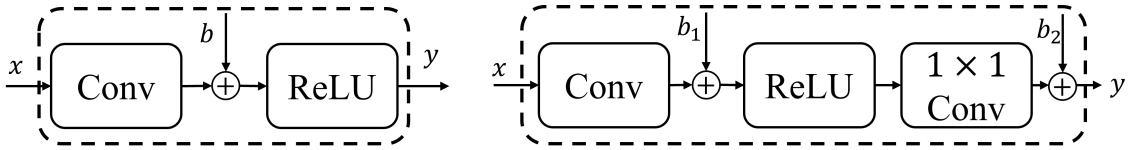


Figure 10.2: Candidate structures for the basic building block $F$. On the left, the building block only consists of the convolution layer and ReLU. On the right, the convolution layer and ReLU, followed by the $1 \times 1$ convolution layer, are presented. Note that the $1 \times 1$ convolution layer can be considered as a generalized version of the $1 \times 1$ linear layer. The inputs and outputs are denoted by $x$ and $y$, respectively, $b$, $b_1$, and $b_2$ denote biases for the convolution layer. (Figure from lecture, drawn by ourselves)

One can easily notice that the one on the right can express the negative outputs, while the first one only can represent positive outputs due to the nature of the ReLU activation function. This gives us a sense

of why the ResNet often has a more complex structure compared to the conventional CNNs. For instance, there is the "skipped over" connection of ResNet as Fig. 10.3, which works equivalently as the second type of structure. In a nutshell, this structure, which has multiple convolution layers (or multiple basic building blocks) within the skipped-over connection, enables more ample representations per each building block. This idea of using sequences of layers as a building block can be generalized in representing other domains, and other structures of neural networks, such as graph neural networks (GNN).

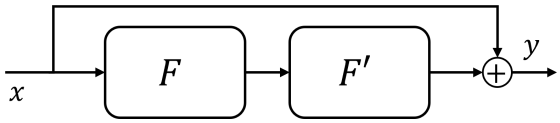


Figure 10.3: "Skipped over" structure of the ResNet. The input signal $x$ is feed-forwarded directly to the output $y$, which skips over the two subsequent building blocks $F$ and $F'$. (Figure from lecture, drawn by ourselves)

### 10.1.2 What's Next?: ConvNeXt

Recently, a more developed CNN-based structure, called ConvNeXt [4], has been proposed. The main building block for the ConvNext, with comparison to ResNet, is presented in Fig. 10.4.

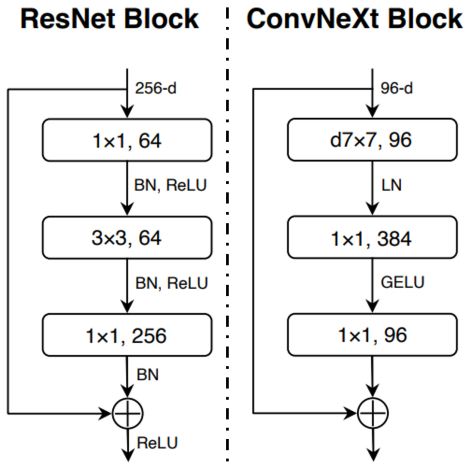


Figure 10.4: Basic building blocks of ResNet and ConvNeXt [4]. The convolution layers are presented with the size of the filters and the number of filters, e.g., $(1 \times 1,\ 64)$ represent 64 filters with its size $1 \times 1$.

The ResNet block and the ConvNeXt block are similar because both networks have a residual signal flow passing over the building blocks. Some key differences between ConvNeXt and ResNet and descriptions are summarized as follows:

1. The first building block, a $7 \times 7$ depth-wise convolution layer, is only performed in each channel in ConvNext, while channel-wise convolution is performed in the ResNet architecture – see Fig. 10.5. Particularly, the first $1 \times 1$ convolution layer acts like a linear combination across different channels. Using a depth-wise convolution layer is much less computationally demanding - see $5^{\text{th}}$ argument for the detail.

2. Unlike ResNet, ConvNext has layer normalization (LN) instead of batch normalization (BN), which acts as a mild nonlinearity.

3. In ResNet, the second layer is composed of $3 \times 3$ convolution layer, with 64 filters. The last layer also shows $1 \times 1$, 256 filters which perform linear combination across the channels.

4. The second layer in ConvNeXt also performs linear combinations across channels. Combined with the first block (and considering that LN has mild nonlinearity), it almost works as a $(7 \times 7,\ 384)$ convolution layer across channels but with much smaller parameters. Therefore, even though ConvNeXt has large convolution filters, and thus can perform a larger convolution, it could maintain a similar number of parameters. In other words, the building block of ConvNeXt is a simpler version of ResNet's building block, or "distillation" - extracting the essence.

5. As can be seen in Fig. 10.6, even though ConvNeXt and ResNet have a similar order of parameters or computational demands, ConvNeXt outperforms ResNet.

6. The ConvNext utilizes Gaussian Error Linear Units (GeLU), which will be elaborated on in the next part.

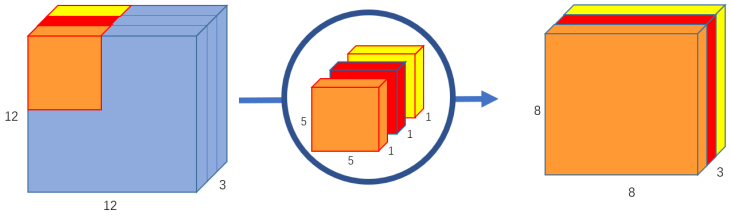


Figure 10.5: Depth-wise convolution is shown. Unlike the typical convolution layer, which performs convolution across the channel, depth-wise (or group-wise) convolution performs convolution within the layer. Image from link.

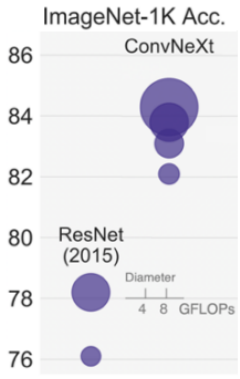


Figure 10.6: Performance comparison between ResNet and ConvNeXt [4]. While ConvNext and ResNet have a comparable number of parameters, ConvNeXt outperforms ResNet.

#### 10.1.2.1 GeLU

In the ConvNext building block, the GeLU [2] has been utilized as the activation function. GeLU behaves similarly to ReLU when the magnitude of the inputs is large but behaves differently around the 0 value – See Fig. 10.7.

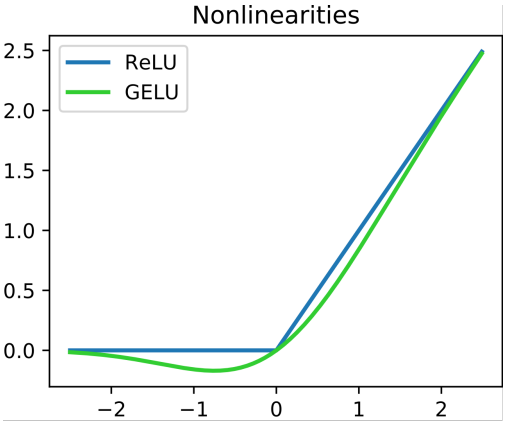


Figure 10.7: GeLU activation function is presented. Figures from wikipedia.

GeLU can be described by

$$\text{GeLU}(x) = x\Phi(x) = \frac{x}{2}\left(1 + \text{erf}\left(\frac{x}{\sqrt{2}}\right)\right),$$

where $\Phi(x)$ is the cumulative density function of the Gaussian distribution. Empirical results show that GeLU works better than ReLU in general, especially when utilized with transformers. GeLU can be interpreted as some random perturbation to the weights or even dropout as well; this is because GeLU is the expected value when we randomly draw $N$ Gaussian distributions and then only pass the input $x$ if its sum is greater than $N$, and otherwise 0. The drawbacks of using GeLU are the increased computational speed and increased non-convexity and non-monotonicity. The first drawback can be reduced by using approximated function. The second "drawback" however, is not a drawback in the context of the neural networks, where we intentionally give nonlinearity to the model – after all, it works better!

#### 10.1.2.2 So, is the ConvNeXt that architecturally different?

Not really. It seems that techniques other than network architecture, such as cosine learning rate schedule, utilizing a learning algorithm named "AdamW", and aggressive data augmentation also help increase the accuracy. We also present some notes and/or topics that might help implement the ResNet.

1. **Stochastic depth regularization** is the counterpart of dropout for the ResNet block. Instead of removing some weights from the kernels, we sometimes zero out the entire building block. Since there is always a residual signal flow, zeroing out the entire block has no harmful effect.

2. **Label Smoothing**: In the classification problem, the classes are normally labeled with one-hot encoding, with a strict 1 value given to its class. However, the final value provided to the cross-entropy function being $\infty$ is needed, which means that the weights also need to be very large. The first problem is that the weight value of $\infty$ is impossible by using a finite number of parameters and memory. In

addition, when the weights have very large values, it could exaggerate a small perturbation of the input (or error in the data), which may eventually lead to false output when there are some noises in the inputs. Therefore, the validation accuracy might be decreased. To mitigate this effect, one can smooth the one-hot encoding to be $\left[\frac{1}{k}, \quad 1-\varepsilon, \quad \frac{1}{k} \quad \cdots \quad \frac{1}{k}\right]$, where $k$ is a tuning parameter and $\varepsilon$ is selected such that the overall summation becomes 1.

## 10.2 Topics on Computer Vision

### 10.2.1 Semantic Segmentation Problem

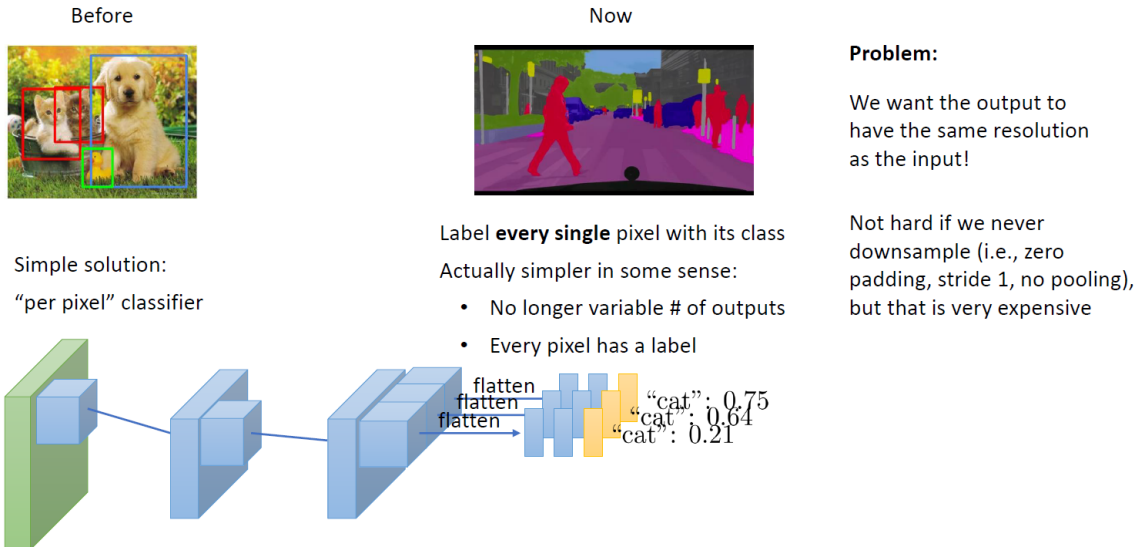


Figure 10.8: Lecture 8, slide 23

Not every image problem is a simple classification problem! There are some problems that we are interested in real life: How do we want our self-driving car to look? Do we want our self-driving car to look and classify things such as people, trees, dogs, etc.? Probably not; we want it to have spatial awareness. For example, you see a scene; you want to recognize on the scene thata there are specific categories of objects in certain places. You'd want to be worried about the person-type object close by. It's very different for the person-type object to be away from the car vs. in front of the car! We can see several classic image problems in Figure 10.9. We also see an example of a simple architecture to classify pixels in Figure 10.8.

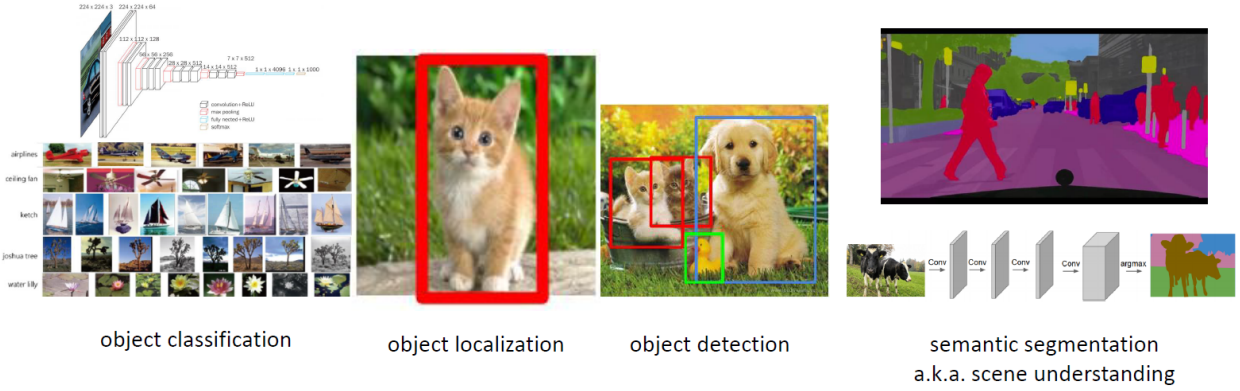


Figure 10.9: Lecture 8, slide 30, [3]

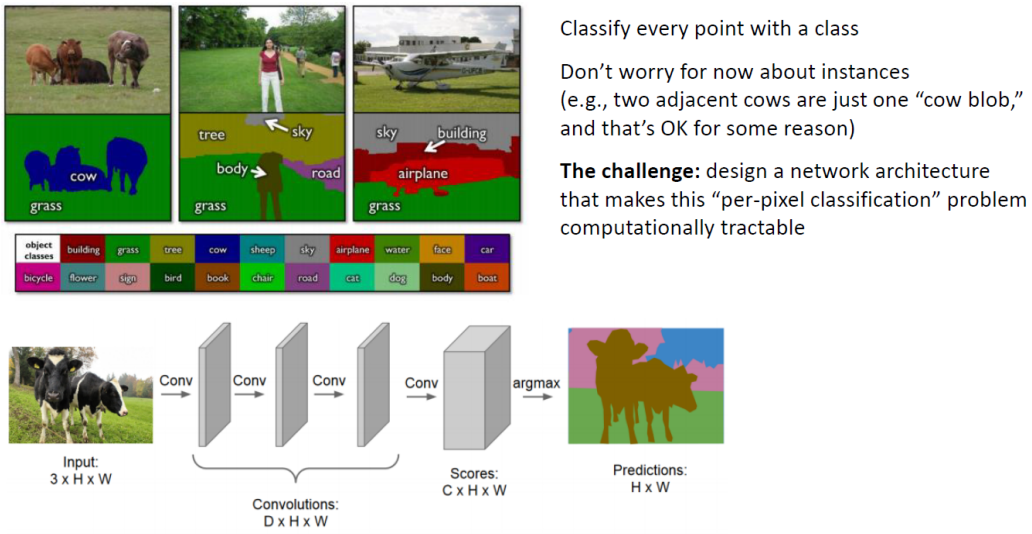


Figure 10.10: Lecture 8, slide 24, [3]

This problem is the semantic segmentation problem, where we are given a scene, categories, and different types of objects. We want to label the scene, with every pixel with what kind of object it contains. How can we adjust what we know to solve this problem? Examining the cows shown in Figure 10.10, note that there are many cows, but our final output is just a big blob of cows. In some cases, we care about classifying different objects differently i.e. here's one person, another person, etc. In the self-driving car case, you might not care! You don't want to hit anyone!

#### 10.2.1.1 Possible Solutions to the Semantic Segmenation Problem

**How can we solve this?** We can look at the architecture shown in Figure 10.10 since we're familiar with it. We know how to make a classifier, and now want to make a per-pixel classifier. You can replicate everything for every pixel. This requires a separate neural net for every pixel, which is not a good way. We can reduce

the size, keep doing same-size convolutions, and, at the end, we'll have a bunch of 1 by 1 convolutions for every pixel, and we'll take the arg max and get the predictions. However, this is still painful for two reasons:

1. There is a significant amount of computation
2. The possibility of this not learning what we want it to learn! Why would this be? It could be that the original network, had the size get smaller. Like pooling, you are inducing an inductive bias for creating hierarchical structures. By eliminating inductive bias, you might not be learning the right thing. You've changed the architecture and the inductive bias.

But what if we just train it to recognize images by themselves for a single label and then just translate that architecture here? But this STILL has the problem, which is that you are not actually training on this problem but a different one where some objects may be very attractive, so you have the edges between labels being blurred in. The receptive field still sees the object slightly off the side and classifies it as that, which causes issues.

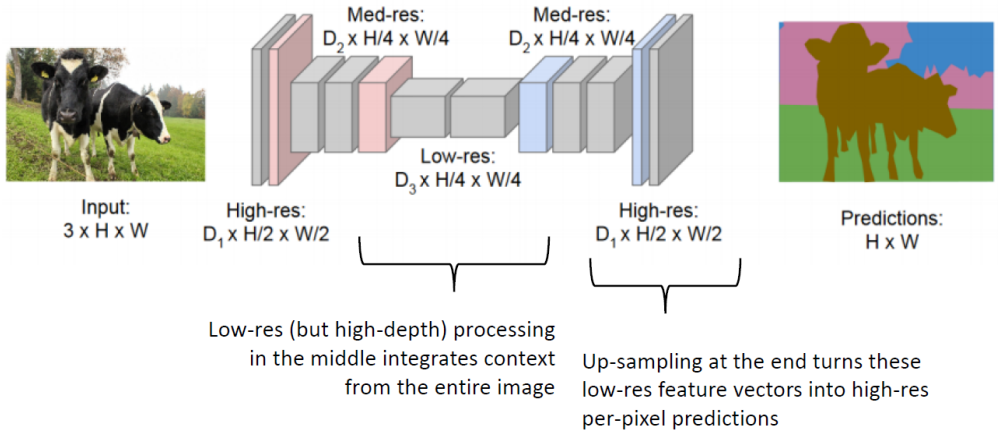


Figure 10.11: Lecture 8, slide 25, [3]

How can we achieve the goal and make it work tractably? One approach is a fully convolutional approach shown in Figure 10.11. We want to have the inductive bias that comes from having high resolution to low resolution, so we build it in. But we still have to build it back out to get something of the same size as the input image. In spirit, we want to learn some kind of feature building to help distill out what's essential to understand what's going on; we want to have something that is a low-resolution representation with many more channels that is capturing contextual information from across the entire image. We then need to do some sort of up-sampling in which we can interpolate these low-resolution things into high-resolution predictions. In everything we've talked about so far, we've talked about down-sampling and not up-sampling.

#### 10.2.1.2 Dimensionality Reduction Perspective

PCA tells us we have a high dimensional space of inputs, but we just look at the directions of the greatest variation. Those give us a sub-space in which we think the interesting action is. Instead of running a learning

algorithm on the original high dimensional space, we run it on the low dimensional space where we expect to have a more reasonable connection between the number of samples we have and the size of our data (classical ML perspective). Also, because of the Eckart-Young theorem, you can think about PCA as solving a problem of finding the best low-rank approximation for the input data. This can be viewed as an end-to-end problem of taking the input and reconstructing the exact input at the output but forcing yourself to downsample and upsample again. Let's simply use a neural net so that we can do nonlinear dimensionality reduction by training end-to-end for this self-reconstruction problem that is called auto-encoders (Will be discussed later in the course). This has a similar structure: start big, go small, and get back to something big, where the goal is to (without labels) generate good intermediate representations that capture the directions that seem to matter the most.

#### 10.2.1.3 Application to the Semantic Segmentation Problem

The current problem is **different** in that while it maintains the same structure, it's fully supervised end-to-end because we have these semantic segmentations already labeled for us, so we're not trying to construct the same thing but something different. You can think of this as canonical correlation analysis, where we have different vectors, and we want to find the appropriate low-dim alignment between the different things.

We want to go "back out" or go from something small to something big. To understand this, we have to understand the two operations that we did in a ConvNet that can make something smaller.

1. A convolution with stride size bigger than 2
2. Max pooling

We need to understand these two counterparts that make things get big. In this particular setting, we take the signal processing (SP) idea that we do classically and replicate it for our setting. Up-sampling and down-sampling are classical SP operations. So we ask ourselves what is the counterpart of those and figure out what we want to make learnable. The general rule is to make the filter weights learnable.

### 10.2.2 Up Sampling

#### 10.2.2.1 Signal Processing Up Sampling

In signal processing, we have a discrete-time signal with values, as shown in Figure 10.12. We perform a convolution and take a filter response such as a piece-wise constant. This is classical signal processing reconstruction and how we interpolate things. It's a convolution operation because it's a filtering operation. When we do this in deep learning, it's called a **transpose convolution**.

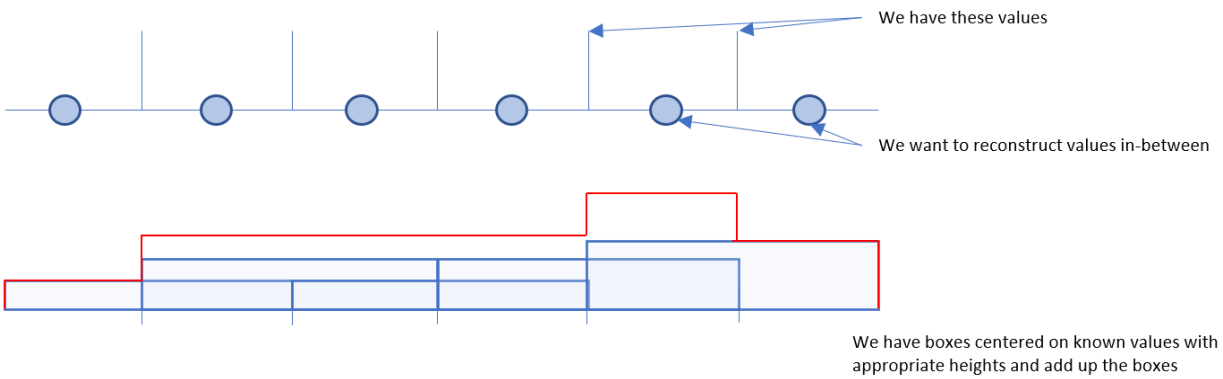


Figure 10.12: Example of Up Sampling, the red line represents the sum, [5]

#### 10.2.2.2 Transpose Convolution

When we perform a stride of $\frac{1}{2}$, we go from a $2 \times 2$ to a $5 \times 5$ image. For normal convolutions, we take a weighted average to get one value. Now, we take one value, use a filter, and copy it into the up-sampled image (Note: When there's an overlap, we take the average; in traditional SP, we add). Normal convolutions and transpose convolutions are shown in Figure 10.13.

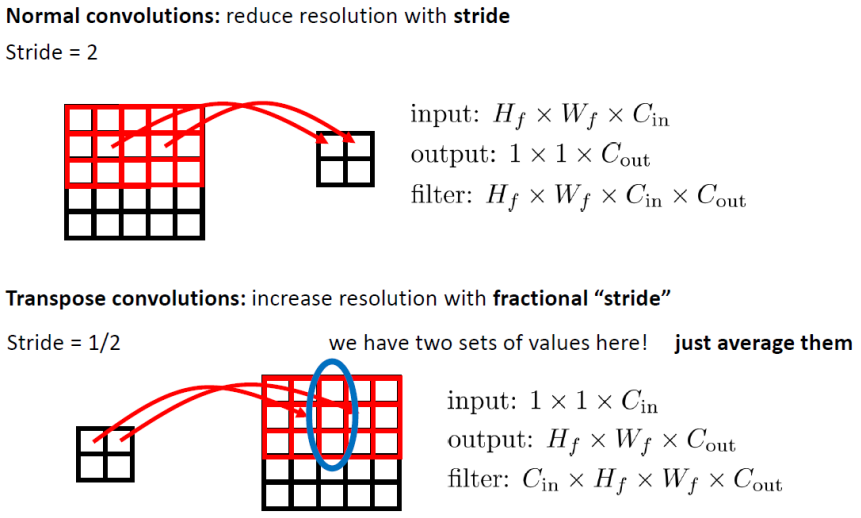


Figure 10.13: Lecture 8, Slide 26, [3]

**Question**: We know in traditional SP if we use simple boxes and use them to interpolate, we see that there are visual artifacts from these boundaries, so the low-resolution result will still have stripes and bars. What's the traditional solution?

**Solution**: We apply a filter afterward in traditional SP. We do the same after a transpose convolution. We use a filtering the operation to remove artifacts (Note: We are learning the filter).

#### 10.2.2.3 Un-Pooling

For **average pooling**, we can just think of un-pooling as filling in the same number and letting the learned filter make the necessary adjustments.

For **max pooling**, we can remember which element was the max. This is demonstrated in Figure 10.14. If we think about down-sampling something and then bringing it back up again, we're allowed to remember! If we remember that we started out with something (input) and went down to the output, we can remember which element was the max and stick it in the same place. It's exactly the same flavor as pooling, except just reversed. Afterward, we get something with 0's in a bunch of places, which is exactly how, in traditional SP terms, we have an intermediate step where we have 0's. We have the rate, but 0's in all the positions where we don't know. Then we use a filter to get those 0's to fill in properly. Same thing here. Remember, every time you have a pool in a ConvNet, it is always preceded by some convolution. So now, we will have an un-pool and a filter. So this is like having a mirror to return to the same size!

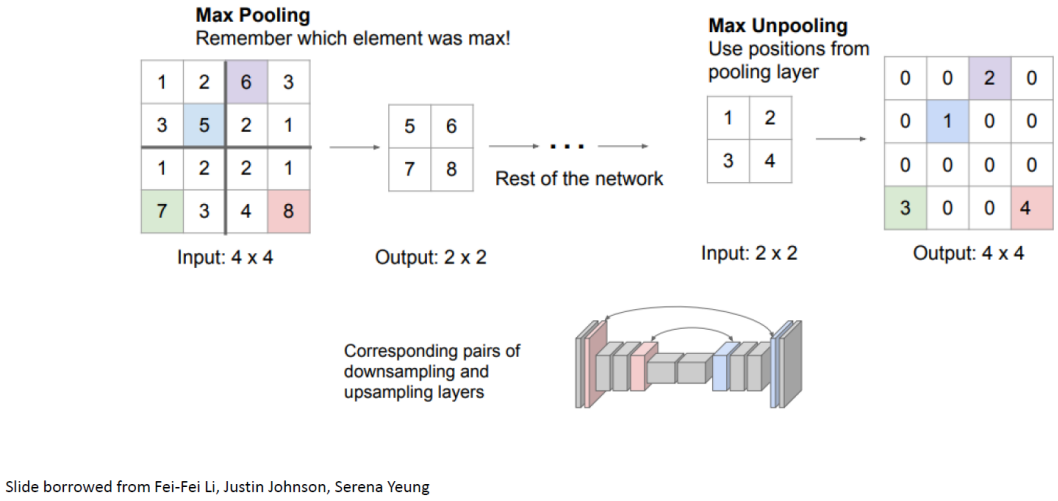


Figure 10.14: Lecture 8, Slide 27, [3]

This works reasonably well. However, it misses something! When we think about natural images or things in the world having a fine to coarse structure, we tend to think of the coarse as defining the essential things that might be happening and the fine structures as details. **But the details can matter!** There may be details that you are throwing away as you move from fine to coarse that might still be relevant to reconstruct at a fine scale. For example, in Kolmogorov complexity, there is a decomposition that people think of. We have an idea of a cat. There is also the level of detail, such as its pose or where the fur is located. However, depending on what you want to do during the reconstruction, different aspects of those details might matter, so if your final reconstruction is 'zoomed in' and is a high-resolution image of the cat, and you want to label the individual hairs, that's different if you just want to label the cat and its shape. This is different than simply saying that the cat is somewhere in the image. The details of the pose don't matter for the cat, but **do** matter for labeling the individual pixels of the final image as being a cat or not a cat! There's a challenge in that information can be legitimately thrown away as you are moving down, but it is needed to get the fine alignments right when going up. For this, there is a classic trick called the **U-net architecture** to deal with this.

#### 10.2.2.4 U-Net Architecture

As shown in Figure 10.15, the finest things are on top, and coarser things are down low. So the evolution of the network is finest, getting coarser, to the coarsest, and then going back up in resolution. Recall the transpose convolution or the un-pooling, and we always have a filter afterward! **The trick of U-Net is to say that we will have a filter afterward; so we just give the filter access to information at the same level that was in the previous thing?** So we just copy the information and give the filters access to both sets of information: the appropriately distilled and up-sampled information and the raw info at the end of the last level.

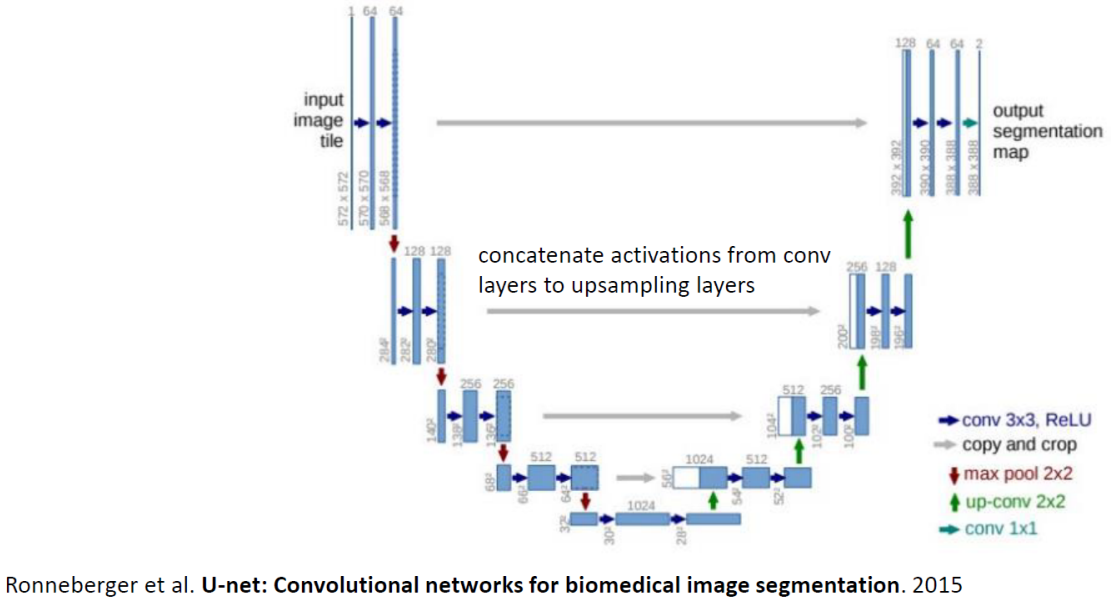


Figure 10.15: Lecture 8, Slide 29, [3]

When we perform up-sampling, (some learned transpose convolution, un-pooling), we have to filter it after. Now, instead of throwing out the raw information, we also concatenate the raw information from this level before down-sampling. Concatenating here means we are adding the information as channels. Now this filter has more information to work with and can use both sets of information to do what it needs to do (able to produce high resolution). The core concept of the U-Net architecture is shown in Figure 10.16.

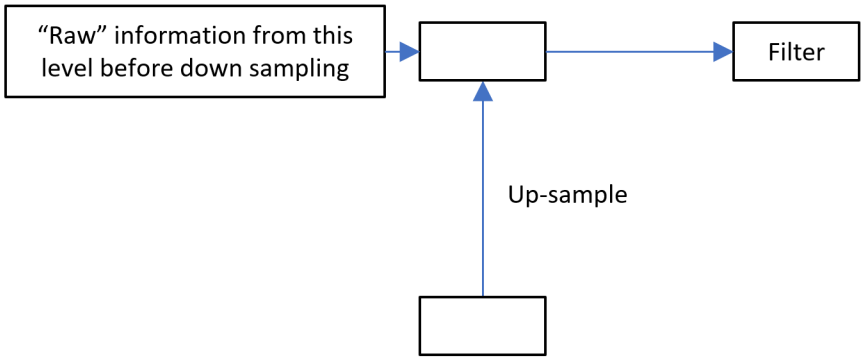


Figure 10.16: Key concept of the U-Net Architecture, [5]

The U-net architecture was the go-to standard way of doing this style of problem. It's also very important for generative models (GANs and so on), and is a standard trick. If we want to maintain information at multiple levels of varying fineness and coarseness. We train the U-net end-to-end, and we hope that the gradients will force the right information to be distilled at the coarse level and that the gradient flowing back at the coarse level does not lose/give up too much. This is because of the information we need to reconstruct properly at this level. Hopefully, the gradient will tell us to keep that. Essentially, the gradients make the down-sampling function by putting the right information that's distilled so that it is pushed down to the next level, and keeps other relevant information in the other positions so that it comes out. Ordinarily, in other architectures, there is no force to do this in the gradient. **There is no inductive bias!**

Today, it's still used in some places, but transformer architectures are sub-planting U-Nets. However, it's still important to know as a case study and it can help us to engineer things that are specific to our problem.

## References

[1] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[2] D. Hendrycks and K. Gimpel. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016.

[3] S. Levine. Designing, visualizing and understanding deep neural networks, February 2021.

[4] Z. Liu, H. Mao, C.-Y. Wu, C. Feichtenhofer, T. Darrell, and S. Xie. A convnet for the 2020s. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11976–11986, 2022.

[5] A. Sahai. Designing, visualizing and understanding deep neural networks lecture notes, February 2023.