Lecture 8: 09/20 (Tuesday)

Lecturer: Prof. Anant Sahai

Scribes: Daisy Zhang

1 Agenda

Topics covered last time were Key ideas & Convnet manifestation. Today, we continue with Convnets, focusing on input standardization and activation normalization. The lecture included the following topics:

- Respect locality
- Respect "invariance" within data for problem domain
- Support hierarchical structure (Fine $\rightarrow$ Coarse)
- "Room to play"

which are related to the following realization techniques:

- Convolutional Structure
- Weight-sharing
- Data Augmentations
- Down Sampling
- Lifting to more channels as we get coarse
- Diversity to support learning
- Dropout
- More layers

2 Locality

We have the observation that many useful image features are local. To tell if a particular patch of an image contains a feature, It is enough to look at the local patch. We get a different output at each image location using the same filter. We assume locality exists in CNN, where inputs that are more close to each other are more correlated. In the context of image inputs, this assumption is qualitatively valid because of local patches of similar color, texture, or lighting. Figure 8.1 shows an example of locality.

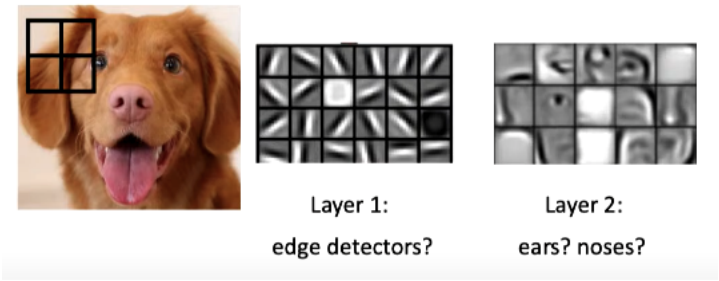

Figure 8.1: This picture of a dog shows that the nature of locality and hierarchy in vision inputs. Locality is exemplified by that most of the local patch in the leftmost picture has similar texture and lighting. Hierarchy is exemplified by that edge features in Layer1, the middle picture, can zoom out and form larger features such as ears and noses in Layer2, the rightmost picture. Image from Lecture8 slides.

3 Weight sharing

Weight sharing in CNN is that a convolutional kernel or filter scans across the whole image input and produces a feature map, so every neighborhood of pixels in the image is processed by the same kernel or filter. That is, the weights in the kernel or filter are shared. We don't assign weight to each individual pixel.

Example: Suppose we have a 5x5 gray-scale picture and a 3x3 kernel (Figure 8.2). We apply the kernel to one pixel of the input image and its surrounding local patch with the same size as the kernel. We get a number from the previous calculation, add the bias, apply non-linearity, and we finally get the new output of one depth of that one pixel of the image. When the image has multiple channels and each channel is convolved with a different kernel, this operation is called *depthwise convolution*.

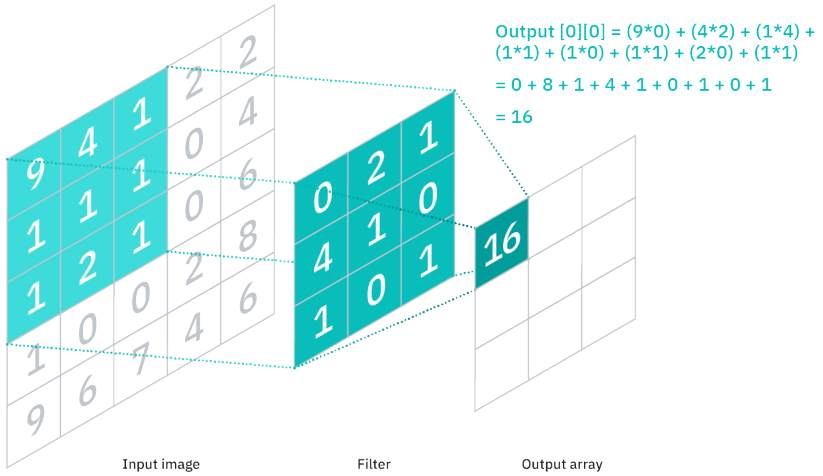

Figure 8.2: Example of a 3x3 convolutional kernel applied to a 5x5 input. The padding size is 0. Image from [2]

The weight-sharing structure provides translational equivalence. The resulting activation map remains the same under the translation of input feature map. Weight sharing also significantly reduces the number of weights of the network, which ensures higher computational speed.

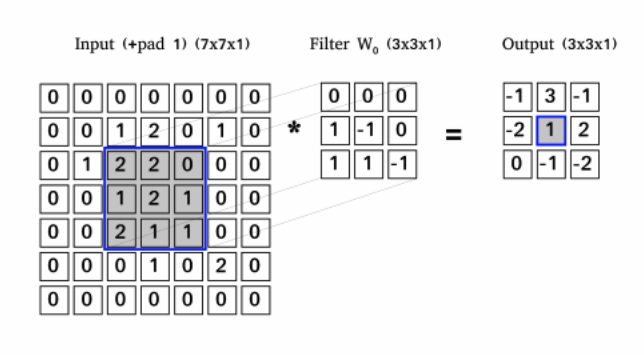

Figure 8.3: A 5x5 input with padding size 1 is applied with a 3x3 kernel, then generates a 3x3 output. Image from [1]

Sometimes, the *padding* operation is introduced. Padding is the addition of crafted pixels on the sides of the image so that the pixels on the borders are not lost from the output of convolution. Padding size is usually one less than kernel size. If the crafted pixels are all zeros, this operation is called *zero-padding*. If the crafted pixels are mirrored values from the original input, this operation is called *mirror-padding*. Deep learning application typically uses zero-padding for practice.

Figure 8.3 shows an example of convolution operation with padding.

Note: *depth* of one input/output layer in the CNN is a term for the number of channels of the layer. For example, the depth of an RGB image is 3.

4 Support hierarchy

Receptive Field is a term borrowed from biological vision. A particular pixel of the output generated by the neural network has a dependence on some, but not all pixels in the input. So this term defines what pixels in the original image this output depends on. Figure 8.4 is an example of receptive field.

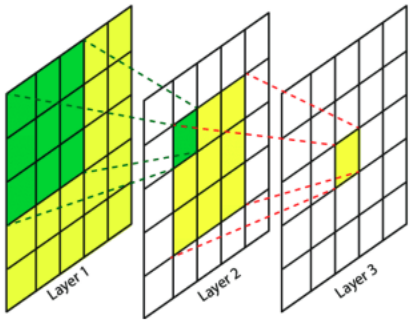

Figure 8.4: The yellow pixel in layer 3 has its receptive fields colored in yellow in layer 2 and layer 3. The green pixel in layer 2 has its receptive field in layer 1 colored in green

If we keep adding convolutions layers on the images, the receptive field will grow linearly. However, linear growth is slower than desired. Faster growth is desired because the neural network can easily learn the full hierarchy.

Therefore, we need a way to do dimensionality reduction on feature maps, i.e., *down sampling*.

For example, if we want to transform a 2x2x4 region of the output into a 1x1x4, we need to represent 2x2 elements with 1 pixel. This is also known as *pooling*.

There are several ways for pooling, including

- average
- max
- pick-one, i.e., pick the pixel at a certain location of the computed local patch, and discard the rest
- weighted average

Pick-one pooling seems a waste of computation because it discards most of the computed pixels. This method of pooling is implemented by *stride* instead, which only computes the pixel at the desired location to save time and computation.

The weighted average can be viewed as a convolutional layer and a stride.

Pooing layers make the receptive field grow exponentially. In addition to dimensionality reduction, pooling also provides local translational invariance, allowing the CNN to be more robust to features varying in their locations.

5 Structure summary

1. Convolutional layers
   (a) A way to avoid needing millions of parameters with images
   (b) Each layer is "local"
   (c) Each layer produces an "image" with roughly the same width and height, and number of channels = number of filters
2. Pooling: moving from fine to coarse but more abstract
   (a) If we ever want to get down to a single output, we must reduce resolution as we go
   (b) Max pooling: downsample the "image" at each layer, taking the max in each region. Max is differentiable. It stops gradient descent to the smaller value which it discards, so it ensures the gradient descent to the important feature.
   (c) This makes it robust to small translation changes.
3. Finishing it up
   (a) At the end, we get something small enough that we can "flatten" it (turn it into a vector), and feed it into a standard fully connected layer.
   (b) Suppose the input size is W, the kernel size K, the stride S, and the padding P. Input and kernel are both squared.
   Then the output size is
   $$\frac{W - K + 2P}{S} + 1$$

6 Data Augmentation

Regularity is needed for the neural network, which is the invariance to some insignificant changes to our knowledge. For example, it is expected that a pattern can still be recognized when it shifts horizontally by 2inches. However, these insignificant changes are not present in the training set. Data augmentation can apply these changes to the input images and feed the newly generated data to the neural network to train for robustness.

In practice, in order to save space, the newly generated images are not stored along with the original huge dataset. These images with data augmentation applied are generated on the fly. Modern deep learning applications typically never see the same image twice, because all data have been augmented during the training. During data augmentation, the label along with the input image normally does not change.

Data Augmentation represents some of the easiest ways that people can transfer the domain knowledge into the training process. We humans understand what are insignificant features by our evolved vision system. This knowledge can be encoded into the practice of data augmentation and drop it into the training. This practice can help the neural network know what is important based on our knowledge.

Basic augmentations include autocontrast, rotation, translation, posterization, etc. Some basic augmentations are exemplified in Figure 8.5. More aggressive augmentations include cutout, Mixup, CutMix, and PixMix [3].

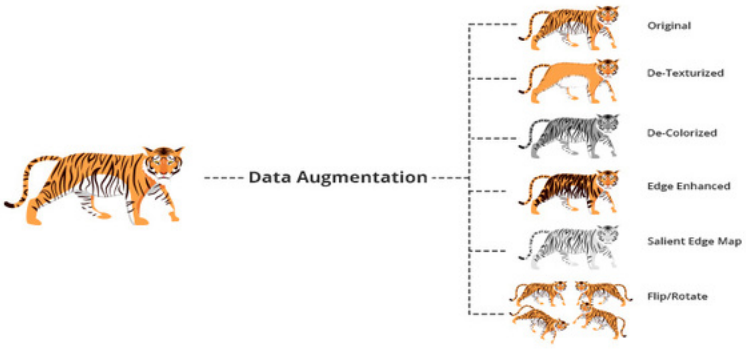

Figure 8.5: Examples of basic augmentations [4]

Figure 8.6: Examples of more aggressive augmentations [3]

7 Standardization and Normalization

Standardization is advised in almost all learning practices. Why do we need standardization? The reason behind it needs to be known. To understand it, it's necessary to know the answer to this question: what tempts a neural network to learn?

Suppose we want to learn $x * w$ and w is the parameter to learn. Then $\frac{d}{dw}xw = x$. Thus, larger $\|x\|$ moves w more in gradient descent. We want to move w more only when we are confident that it's in the right direction. The largeness of $\|x\|$ needs to relate to the essence of data. For example, if x is a weight measurement, then its pure value is larger when its unit is mg than when its unit is kg. However, in this case, we don't want to update more on w, the parameter to learn, because the largeness of data doesn't reflect that it is importance.

Accordingly, when the input size (magnitude) does not carry confident and important information, such as weight data in kg and in mg, we need to convert it to data with zero mean and unit variance.

The design choice with zero mean and unit variance is good because $0 + 0 = 0$ and $1 * 1 = 1$. We want to make sure that the neural network is learning with the greatest sensitivity when the dataset is essentially changing. For example, the elbow point of a ReLU network can capture where the predicted values vary. It will be hard for a ReLU network to learn if the elbow points are not aligned with the output action space. Standardization is a way to align inputs to where the function can learn more easily.

The expressive power of the network is not lost with standardization and normalization. The expressive power is embedded in the bias and the weight terms, which can shift the mean and variance.

We perform standardization on the training set by subtracting the mean and scaling the variance computed from the training set. We can use the same mean and variance from the training set to normalize the validation set.

References

[1] E. S. Agency. Machine learning group: About machine learning: Convolutional neural networks introduction.

[2] I. C. Education. Ibm cloud learn hub: What are convolutional neural networks?, 2020.

[3] D. Hendrycks, A. Zou, M. Mazeika, L. Tang, B. Li, D. Song, and J. Steinhardt. Pixmix: Dreamlike pictures comprehensively improve safety measures. *CVPR*, 2022.

[4] A. Oberoi. What is data augmentation in deep learning?, 2022.