Convolutional Neural Networks

Oliver Zhao

1 Convolutional Neural Networks

1.1 Convolution Layer

A 2D discrete convolution operation is defined as

$$g(x,y) = w * f(x,y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s,t) f(x-s, y-t), \tag{1.1}$$

Where $f(x,y)$ is the original image, $g(x,y)$ is the filtered image, and w is the filter kernel. Different filters may be used to detect for different types of features, as shown in the example below on the right. Additionally, a visual representation is provided of a convolution on the figure below on the left.

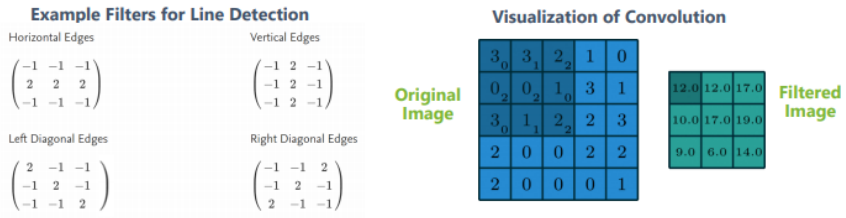


Figure 1: On the left, we see a 3x3 filter being applied to the image. On the right, we see some common filters used for feature detection.

Notice that the filtered output image has smaller dimensions than the original input image. In general, for a image that has $n \times n$ pixels, a filter with $m \times m$ pixels results in an output image that is $(n-m+1) \times (n-m+1)$ pixels. As a result, applying a series of convolutions results in a shrinking image. Additionally, feature information may be lost at the corners and sides of the image. To address this, some models implement "padding". Padding is when the input

image is surrounded with a layer of pixels with a value of zero, such that the output image is the same dimension as the input image. An example is shown below.

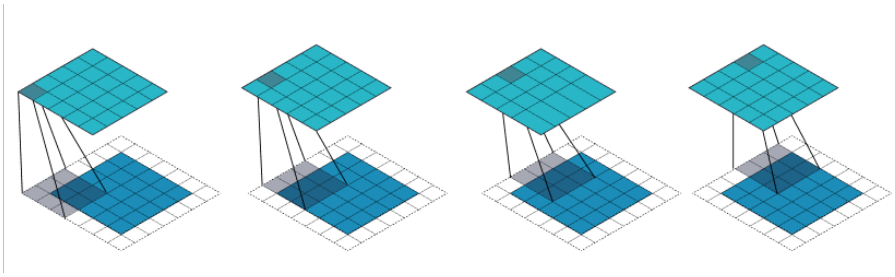


Figure 2: Notice that the padding causes the input image to have the same dimensions as the input image.

Another way to customize how the convolution is applied is by specifying the stride length. The stride length is just the number of pixels that filter shifts each time. Typically, the stride length is just set to 1. Consider the figure below, in which on the left we see a filter with a stride length of 1, and on the right we see a filter with a stride length of 2. Notice that the filter with the longer stride length results in a smaller output image.

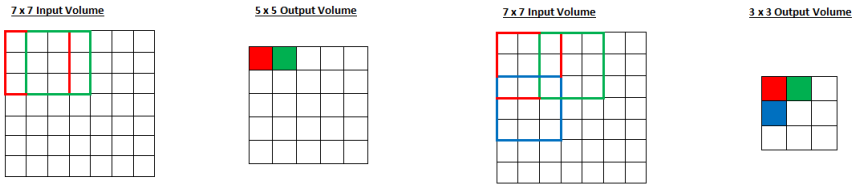


Figure 3: On the left is a filter with a stride length of 1, on the right is a filter with a stride length of 2.

At the conclusion of a convolution operation. Typically a ReLU activation function is applied, in which all negative values in the output are converted to zero and all positive values in the output are the same.

1.2 Pooling Layer

Pooling is used to down sample an image or feature map. It serves two main purposes: (1) reducing image size for faster computation and (2) it also helps reducing overfitting by reducing the complexity of the images. The most common types of pooling at average pooling and max pooling. As the names suggest,

average pooling is when the average value for each patch is extracted and max pooling is when the maximum value of each patch is extracted. The pooling size is typically 2×2 or 4×4. An example of these pooling methods are shown below.

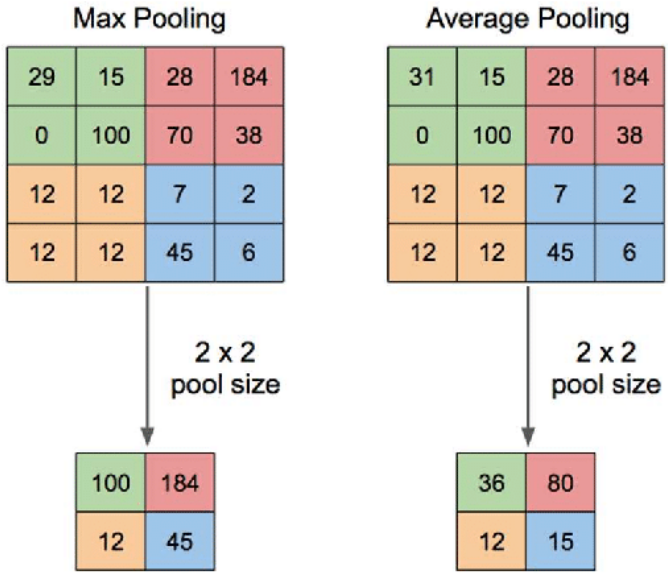


Figure 4: Max pooling and average pooling result in different outputs.

2 Classification

A classification-based CNN architecture is usually made of up several convolution layers appended to a few dense layers. These dense layers feed into an output layer in which the number of nodes is equal to the number of classes.

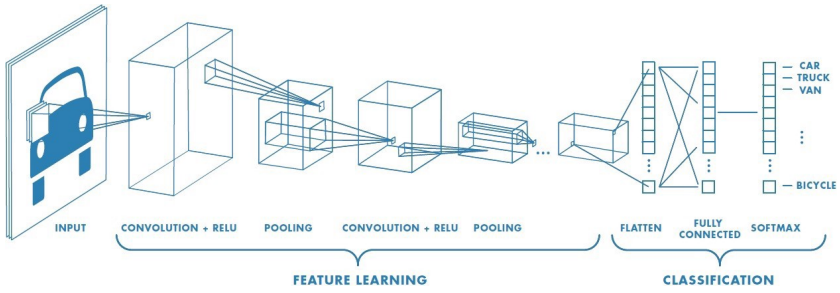


Figure 5: Example CNN architecture for classification-based tasks.

In the final layer of the classification-based CNN, the softmax function is used to ensure that the sum of the outputs of every node in the final layer is equal to 1. The softmax function is denoted as

$$f(s)_i = \frac{e^{s_i}}{\sum_j^C e^{s_j}}, \tag{2.1}$$

Where s_j is the score inferred by the neural network for the jth class and C is the number of classes. Now that we have out final outputs for the classification-based CNN, we must use a loss function that can reasonably describe the performance of the model. It is not useful to describe the outputs as simply "right" and "wrong". For example, consider an image that belongs to Class A. A model that classifies the image to be 90% Class A is better than a model that classifies the image to be 65% Class A, even though both ultimately lead to the "correct" classification. To take these into account, the cross-entropy loss function typically is used for multi-class classification problems, where

$$\mathcal{L} = -\sum_i^C t_i \log(s_i), \tag{2.2}$$

Where t_i is the ground truth and s_i is the model-predicted class for each class i in the set of classes C. Note that if $s_i = t_i = 1$ (perfect classification), the loss is zero.

3 Object Detection

3.1 You Only Look Once (YOLO)

3.1.1 Workflow

The YOLO object detection model splits an image into $S \times S$ cells, where if an object's center falls into a cell, that cell is "responsible" for detecting the object. Each cell predicts the location of B bounding boxes, a confidence score, and a probability of an object class conditioned on the existence of an object in the bounding box.

The model identifies the coordinates of the bounding box with four values: x, y, w, h. x and y identify the center x-coordinate and y-coordinate, while w and h indicate the width and height, respectively. These four values are normalized by the image width and height, such that $x, y, w, h \, \epsilon \, (0, 1]$.

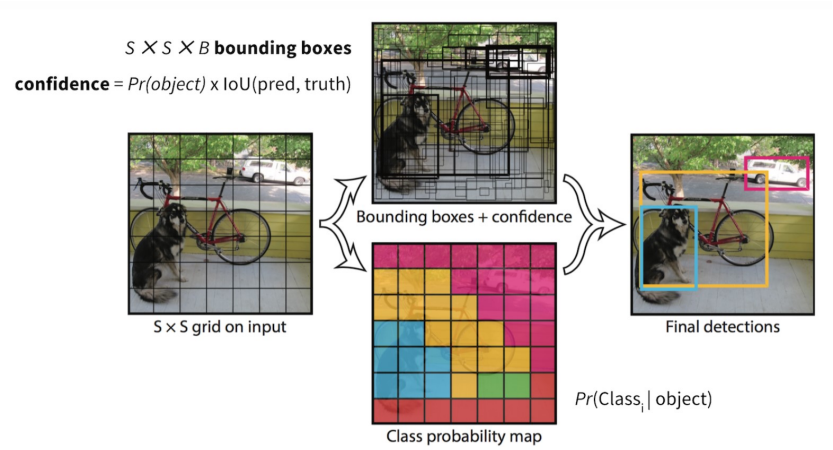


Figure 6: YOLO workflow.

A confidence score indicates the likelihood that the cell contains an object, or Pr(contains object)×IoU(pred,truth). If the cell contains an object, it predicts one set of probabilities of this object belonging to every class $C_i,\ i = 1, \ldots, K$.

Thus, one image contains $S \times S \times B$ bounding boxes, with each box containing 4 location predictions, 1 confidence score, and K conditional probabilities for object classification. The total prediction values for one image thus have a tensor of dimensions $S \times S \times (5B + K)$.

3.1.2 Architecture

The YOLO architecture consists of a series of convolutional layers based off the DarkNet architecture, before feeding into several fully connected layers. In short, the general architecture is similar to that of classification-based CNNs with the exception that the output has a final prediction shape of $S \times S \times (5B + K)$.

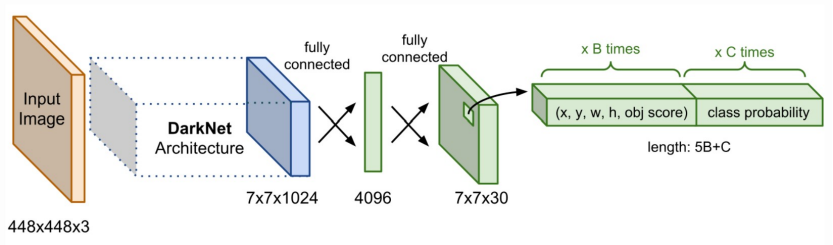


Figure 7: YOLO architecture.

3.1.3 Loss Function

The loss function for the YOLO architecture is

$$\mathcal{L}_{\text{tot}} = \mathcal{L}_{\text{loc}} + \mathcal{L}_{\text{cls}}, \tag{3.1}$$

Where

$$\begin{aligned}
\mathcal{L}_{\text{loc}} &= \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{\text{obj}} [(x_i - \hat{x_i})^2 + (y_i - \hat{y}_i)^2 + (\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2] \\
\mathcal{L}_{\text{cls}} &= \sum_{i=0}^{S^2} \sum_{j=0}^{B} (1_{ij}^{\text{obj}} + \lambda_{\text{noobj}}(1 - 1_{ij}^{\text{obj}}))(C_{ij} - \hat{C}_{ij})^2 + \sum_{i=0}^{S^2} \sum_{c \epsilon \mathcal{C}} 1_i^{\text{obj}} (p_i(c) - \hat{p}_i(c))^2
\end{aligned} \tag{3.2}$$

Where λ_{coord} and λ_{noobj} is used to control how much we want to weigh the loss from the bounding box coordinate predictions and confidence score predictions, respectively. Meanwhile,

- 1_i^{obj}: indicator function of whether cell i contains an object.
- 1_{ij}^{obj}: indicates whether the jth bounding box of cell i is responsible for the object prediction.
- C_{ij}: confidence score of cell i [Pr(contains object)×IoU(pred,truth)].
- $\hat{C}_{ij}$: predicted confidence score.
- $\mathcal{C}$: set of all classes.
- $p_i(c)$: conditional probability of whether cell i contains an object of class $c \epsilon \mathcal{C}$.
- $\hat{p}_i(c)$: predicted conditional class probability.

Note that the loss function shows that classification error is only penalized if the object is present in the grid cell, or $1_i^{\text{obj}} = 1$. Meanwhile, box coordinate error is only penalized if the predictor is "responsible" (i.e. has the highest IoU) for the ground truth box, or $1_{ij}^{\text{obj}} = 1$.

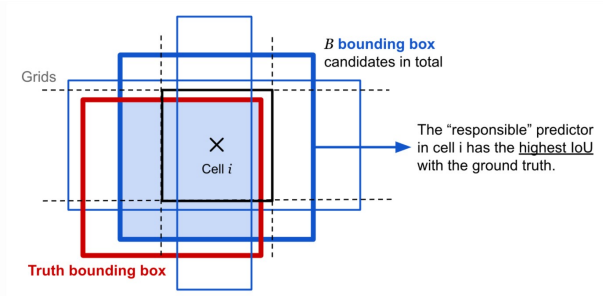
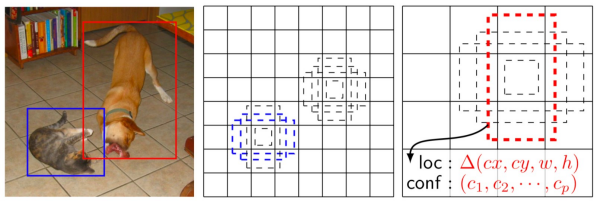


Figure 8: Example of bounding box "responsible" for predicting the object in cell i.

3.2 Single-Shot Detection (SSD)

3.2.1 Workflow

In contrast to YOLO, SSD does not split the image into grids of arbitrary sizes. Instead, it predicts offset of predefined anchor boxes for every location of the feature map. Each box has a fixed size and positive relative to its corresponding cell. Feature maps are different levels have different receptive field sizes. The anchor boxes on different levels are rescaled so that one feature map is only responsible for objects at one particular scale. For example, in the figure below, the cat is captured by the 8×8 feature map while the dog is captured by the 4×4 feature map.



(a) Image with GT boxes (b) 8×8 feature map (c) 4×4 feature map

Figure 9: SSD anchor boxes are different feature levels.

Like the YOLO workflow, the width, height, and center coordinates of the anchor boxes are all normalized to $[0, 1)$. At a location (i, j) of the ℓth feature layer of size $m \times n$, $i = 1, \ldots, n$ and $j = 1, \ldots, m$, we have a unique linear scale proportional to the layer level and five different box aspect ratios, in addition to a special scale when the aspect ratio is 1. This gives us 6 anchor boxes per feature cell.

$$
\begin{aligned}
\text{level index: } & \ell = 1, \ldots, L \\
\text{scale of boxes: } & s_\ell = s_{\min} + \frac{s_{\max} - s_{\min}}{L-1}(\ell - 1) \\
\text{aspect ratios: } & r \,\epsilon\, \{1, 2, 3, \frac{1}{2}, \frac{1}{3}\} \\
\text{additional scale: } & s'_\ell = \sqrt{s_\ell s_{\ell+1}} \text{ when } r = 1 \\
\text{width: } & w_l^r = s_\ell \sqrt{r} \\
\text{height: } & h_l^r = s_\ell / \sqrt{r} \\
\text{center coordinates: } & (x_\ell^i, y_\ell^i) = \left(\frac{i + 0.5}{m}, \frac{j + 0.5}{n} \right)
\end{aligned}
\tag{3.3}
$$

At every location, the model outputs 4 offsets and c class probabilities by applying a $3 \times 3 \times p$ convolution filter (p = number of channels in feature map) for every one of the k anchor boxes. Thus, given a feature map of size $m \times n$, we get $kmn(c + 4)$ prediction filters.

3.2.2 Architecture

In contrast to YOLO, the SSD does not rely on successive convolutional layers to feed into the dense layers. Instead, it relies on a concept called pyramid representation, in which features are extracted from each successive convolution layer and compiled directly to the dense layer.

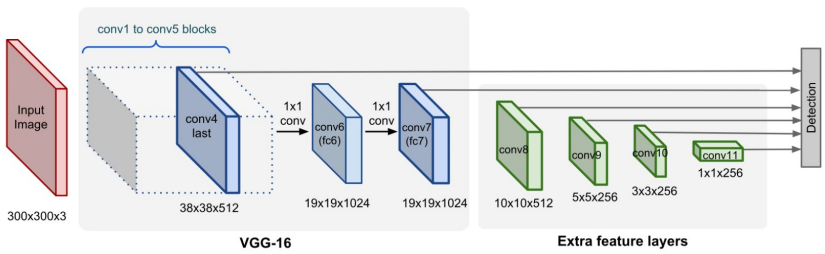


Figure 10: SSD architecture.

3.2.3 Loss Function

Similar to YOLO, the loss function is some combination of a localization loss and classification loss, where

$$\mathcal{L}_{\text{tot}} = \frac{1}{N}(\mathcal{L}_{\text{cls}} + \alpha\mathcal{L}_{\text{loc}}), \tag{3.4}$$

Where N is the number of matched bounding boxes and α balances the weights between two losses, picked by cross-validation. The localization loss is a smooth L1 loss between the predicted bounding box correction and the true values, or

$$\mathscr{L}_{\text{loc}} = \sum_{i,j} \sum_{m\epsilon\{x,y,w,h\}} 1_{ij}^{\text{match}} L_1^{\text{smooth}}(d_m^i - t_m^j)^2 \tag{3.5}$$

Where the values t_m^j are defined as

$$\begin{aligned} t_x^j &= (g_x^j - p_x^i)/p_w^i \\ t_y^j &= (g_y^j - p_y^i)/p_h^i \\ t_w^j &= \log(g_w^j/p_w^i) \\ t_h^j &= \log(g_h^j/p_h^i) \end{aligned} \tag{3.6}$$

And the smooth L1 loss is defined as

$$L_1^{\text{smooth}}(x) = \begin{cases} 0.5x^2 & \text{if } |x| < 1 \\ |x| - 0.5 & \text{otherwise} \end{cases} \tag{3.7}$$

1_{ij}^{match} is an indicator function showing whether the ith bounding box with coordinates $(p_x^i, p_y^i, p_w^i, p_h^i)$ is matched to the jth ground truth box with coordinates $(g_x^j, g_y^j, g_w^j, g_h^j)$ for any object. $d_m^i, m\ \epsilon\ \{x, y, w, h\}$ are the predicted correction terms.

Meanwhile, the classification loss is a softmax loss over multiple classes, where

$$\mathcal{L}_{\text{cls}} = -\sum_{i\epsilon\text{pos}} 1_{ij}^k \log(\hat{c}_i^k) - \sum_{i\epsilon\text{neg}} \log(\hat{c}_i^0), \text{ where } \hat{c}_i^k = \text{softmax}(c_i^k), \tag{3.8}$$

Where 1_{ij}^k indicates whether the ith bounding box and jth ground truth box are matched for an object in class k. "pos" is the set of matched bounding boxes (N items in total) and "neg— is the set of negative examples. Once all anchor boxes are sorted by objectiveness confidence score, the model picks the top candidates for training so that neg:pos is at most 3:1.