Convolutional Neural Network

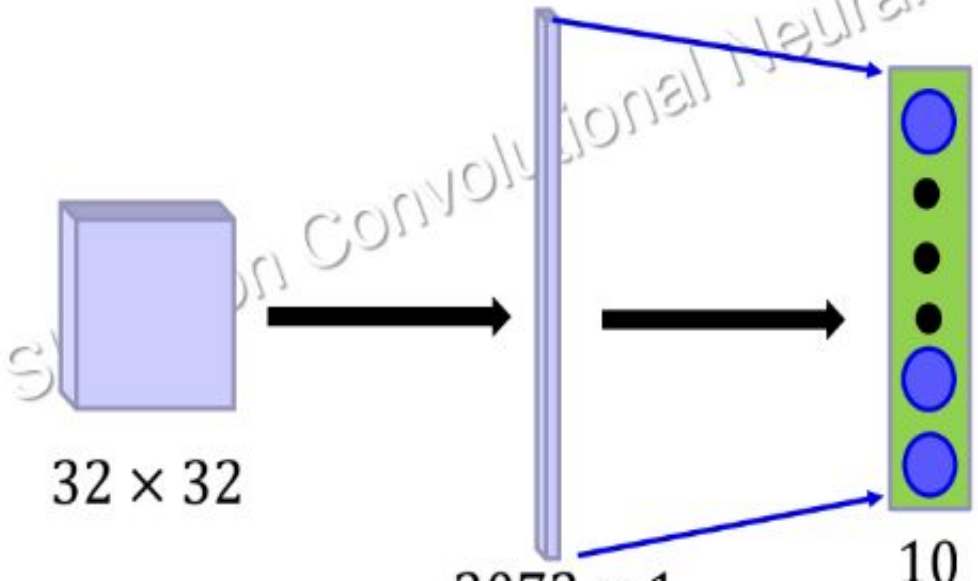
CNN

Neural Network for Image Data

- Suppose the input vector is from an image database.
- Task is to classify if the image contains a bird or not.
- You have to detect a bird in the image.
- Each image is of size say 256X256.
- What will be the size of the input layer here ?
  - 66,536
- If the first hidden layer contains 1024 units, how many weight parameters will be involved ?
  - 66,536 X 1024 = 68,132,864

Convolution vs Fully connected Layer

- Fully connected layer
  - Suppose the input image size is 32×32, we first flatten it (convert to 1-d vector)
  - Dimension of input vector: $32 \times 32 = 3072$.
  - If there are 10 units (neuron) in the first hidden layer of a FC network.
  - Then each unit looks at the full input vector.



$a = f(Z) = f(W^T X + b)$.

Trainable parameters

Weights: 3072×10

Bias: 10

Total trainable parameters : 30730

Convolution vs Fully connected Layer

⚫ Convolution layer

The input image size is 32 × 32.
If there are 10 filters, each of size 3 × 3 in the first Conv layer of the network.
Total trainable parameters ?
100

Convolutional Layer : Consider a convolutional layer with **"l"** feature maps as the input and **"k"** feature maps as output. The filter size is "**n*m**".

Ex. input has ***l=32*** feature maps as inputs, filter size is ***n=3 and m=3*** and ***k=64*** feature maps as outputs.

So, as an output from first conv layer, we learn 64 different **3*3*32** filters .

As well there is a term called bias for each feature map.

So, the total number of parameters are "**(n*m*l+1)*k**".

Number of parameters in a CONV layer would be :

((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.

The same expression can be written as follows: **((m * n * d)+1)* k)**, added 1 because of the bias term for each filter.

Solution

l=1 = one feature map

n*m=3*3

k=10

((3*3*1) +1)*10

=(9+1)*10

=10*10

=100

Convolution vs Fully connected Layer

⚫ Convolution layer

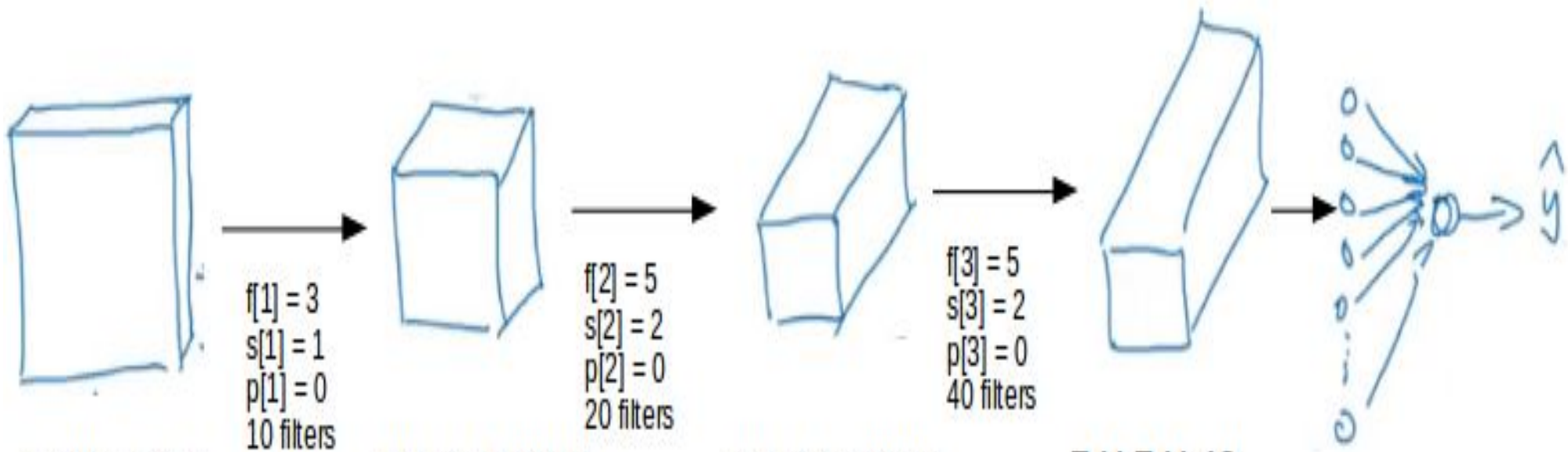
The input image size is 32 × 32.

If there are 6 filters, each of size 5× 5 in the first Conv layer of the network.

Total trainable parameters ?
Ans=

If we see the number of parameters in case of a convolutional layer, it will be = (5*5*1 + 1) * 6 (if there are 6 filters), which is equal to 156.

Convolutional layers reduce the number of parameters and speed up the training of the model significantly.



nh[0] = nw[0] = 39
nc[0] = 3

nh[1] = nw[1] = 37
nc[1] = 10

nh[2] = nw[2] = 17
nc[1] = 20

$nh[1] = nw[1] = (n + 2p - f) / s + 1$
$nh[1] = nw[1] = (39 + 0 - 3) / 1 + 1$
$nh[1] = nw[1] = 37$

There are a number of hyperparameters that we can tweak while building a convolutional network.

These include
the number of filters,
size of filters,
stride to be used,
padding, etc

ANN

```
def build_model_using_sequential():
  model = Sequential([
    Dense(hidden_units1, kernel_initializer='normal',
activation='relu'),
    Dropout(0.2),
    Dense(hidden_units2, kernel_initializer='normal',
activation='relu'),
    Dropout(0.2),
    Dense(hidden_units3, kernel_initializer='normal',
activation='relu'),
    Dense(1, kernel_initializer='normal', activation='linear')
  ])
  return model
# build the model
model = build_model_using_sequential()
```

```
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10))
model.summary()
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])

history = model.fit(train_images, train_labels, epochs=10,
                    validation_data=(test_images, test_labels))
```

Ref: https://www.tensorflow.org/tutorials/images/cnn

Introduction

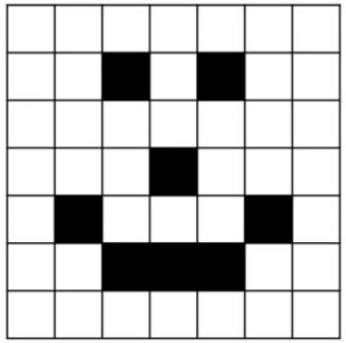
- Machine Learning, a computer science branch that studies the design of algorithms that can learn. Deep learning is a sub-field of machine learning that is inspired by artificial neural networks, which in turn are inspired by biological neural networks.
- Convolutional Neural Networks are very similar to ordinary Neural Networks that have learn-able weights and biases.
- In neural networks, Convolutional neural network owns major applications in image recognition, image classification, detection of objects, recognizing faces, etc.

- How our brain classifies an image!

Whenever we see an image, our brain looks for the features in the image to classify that image. We categorize things by recognizing the features. We need clear features in the image to classify it.

In simple words, Neural Networks works exactly like a human mind.

How the Computer sees an image?



0	0	0	0	0	0	0
0	1	0	0	0	1	0
0	0	0	0	0	0	0
0	0	0	1	0	0	0
0	1	0	0	0	1	0
0	0	1	1	1	0	0
0	0	0	0	0	0	0

What We See

What Computers See

An image is a matrix of pixels. If there is a black and white image, then we will get a 2D array. While, if we passed a colored image, then we will get a 3D array, which means it has an extra parameter for depth which is RGB channel as shown below. The pixel value lies between 0 and 255, and images are stored in bytes (0 to 255).

When a computer sees an image (takes an image as input), it will see an array of pixel values. Depending on the resolution and size of the image, it will see a 32 x 32 x 3 array of numbers (The 3 refers to RGB values)

“Convolutional neural network”

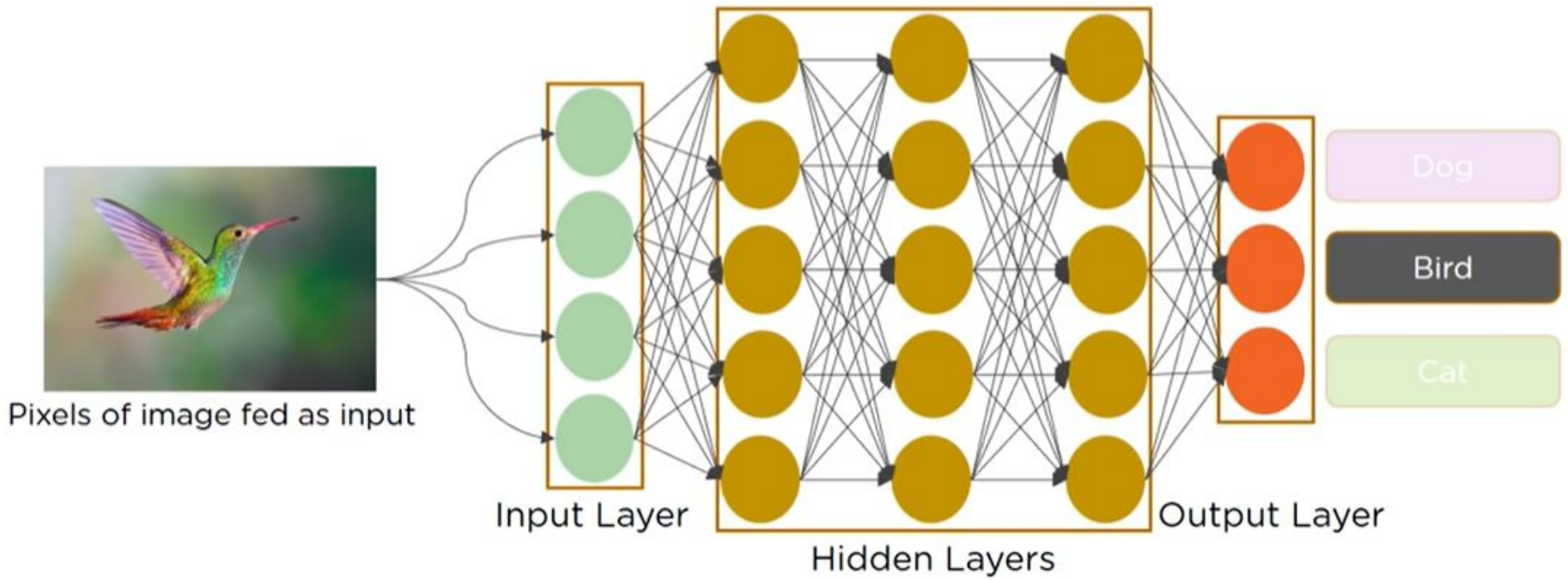
- A CNN is a class of deep, feed-forward (not recurrent) artificial neural networks.
- The name “convolutional neural network” indicates that the network employs a mathematical operation called convolution.
- Convolution is a specialized kind of linear operation.
- Convolutional networks are simply neural networks that use convolution in place of general matrix multiplication in at least one of their layers.
- Basic Structure

- A more detailed overview of what CNNs do would be that you take the image, pass it through a series of convolutional, pooling (downsampling), and fully connected layers, and get an output. The output can be a single class or a probability of classes that best describes the image.

How image recognition works?

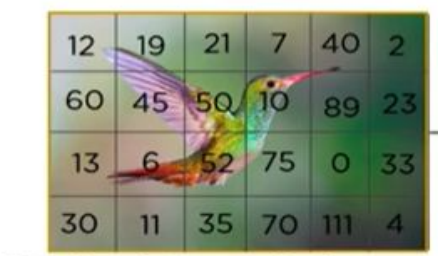
Do you know how Deep Learning recognizes the objects in an image?

It does it using a Convolution Neural Network



How image recognition works?

Let's see how CNN identifies the image of a bird



Pixels of image fed as input

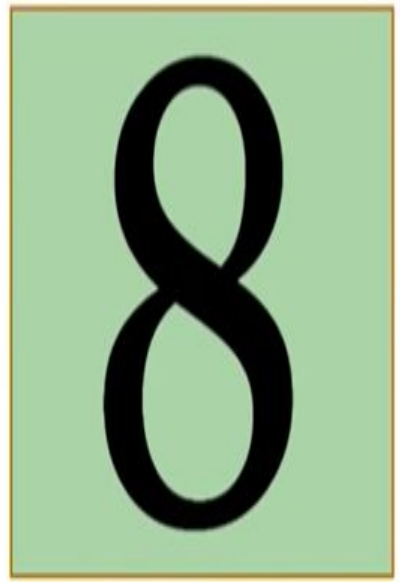
Input Layer

Input layer accepts the pixels of the image as input in the form of arrays

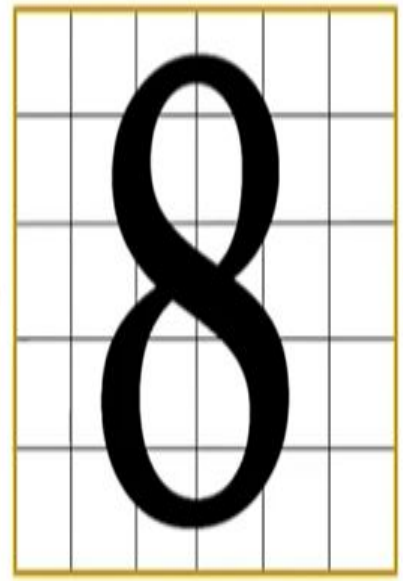
CNN is a feed forward Neural network

Convolution operation forms the basis of any Convolution Neural Network

In CNN, every image is represented in the form of arrays of pixel values



Real Image of the digit 8

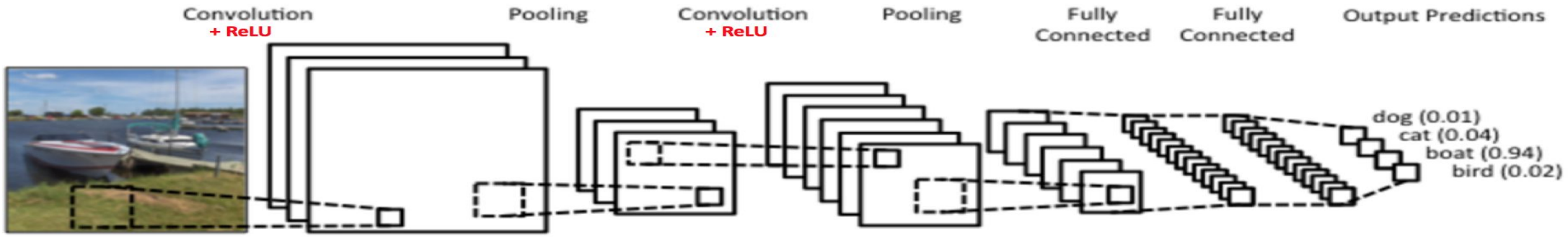


Represented in the form of an array

0	0	1	1	0	0
0	1	0	0	1	0
0	0	1	1	0	0
0	1	0	0	1	0
0	0	1	1	0	0

Digit 8 represented in the form of pixels of 0's and 1's

Classic CNN architecture



There are four main operations in the ConvNet shown in above Figure

1. Convolution
2. Non Linearity (ReLU)
3. Pooling or Sub Sampling
4. Classification (Fully Connected Layer)

A classic CNN architecture would look like this.

Input -> Conv -> ReLU -> Conv -> ReLU -> Pool -> ReLU -> Conv -> ReLU -> Pool ->Fully Connected

Basic Architecture of Convolutional Neural Network(CNN)

Convolutional network terminology

Convolution

Filter/Kernel

Padding

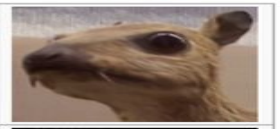
Striding

Feature map/activation map (Output of convolution)

Pooling

Different values of the **filter** matrix will produce different Feature Maps for the same input image.
As an example, consider the following input image:

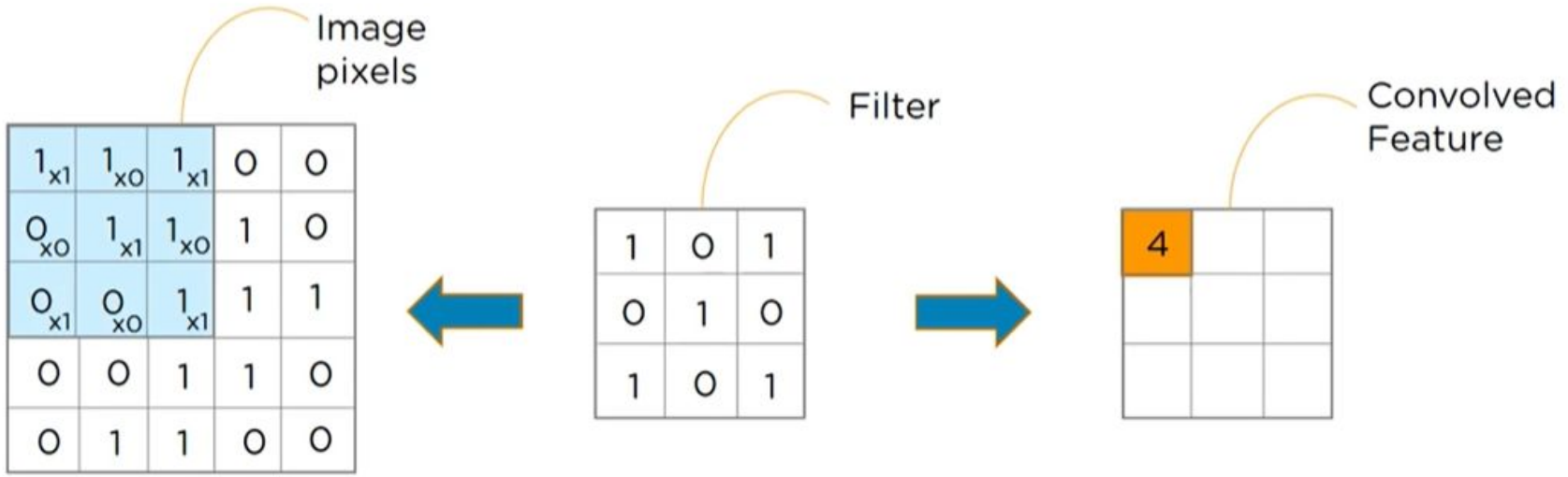
this means that different filters can detect different features from an image, for example edges, curves etc.

Operation	Filter	Convolved Image
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
Gaussian blur (approximation)	$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	

Convolution Layer

A Convolution Layer has a number of filters that perform convolution operation

Every image is considered as a matrix of pixel values.
Consider the following 5*5 image whose pixel values are only 0 and 1



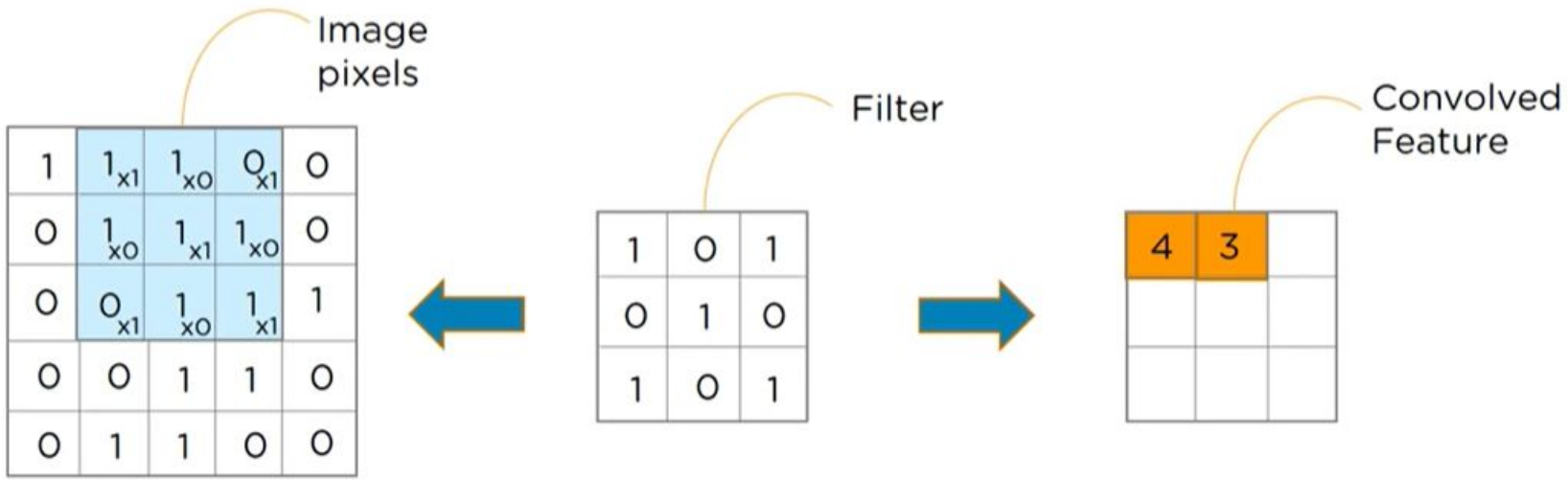
Sliding the filter matrix over the image and computing the dot product to detect patterns

Parameter sharing

Convolution Layer

A Convolution Layer has a number of filters that perform convolution operation

Every image is considered as a matrix of pixel values.
Consider the following 5*5 image whose pixel values are only 0 and 1

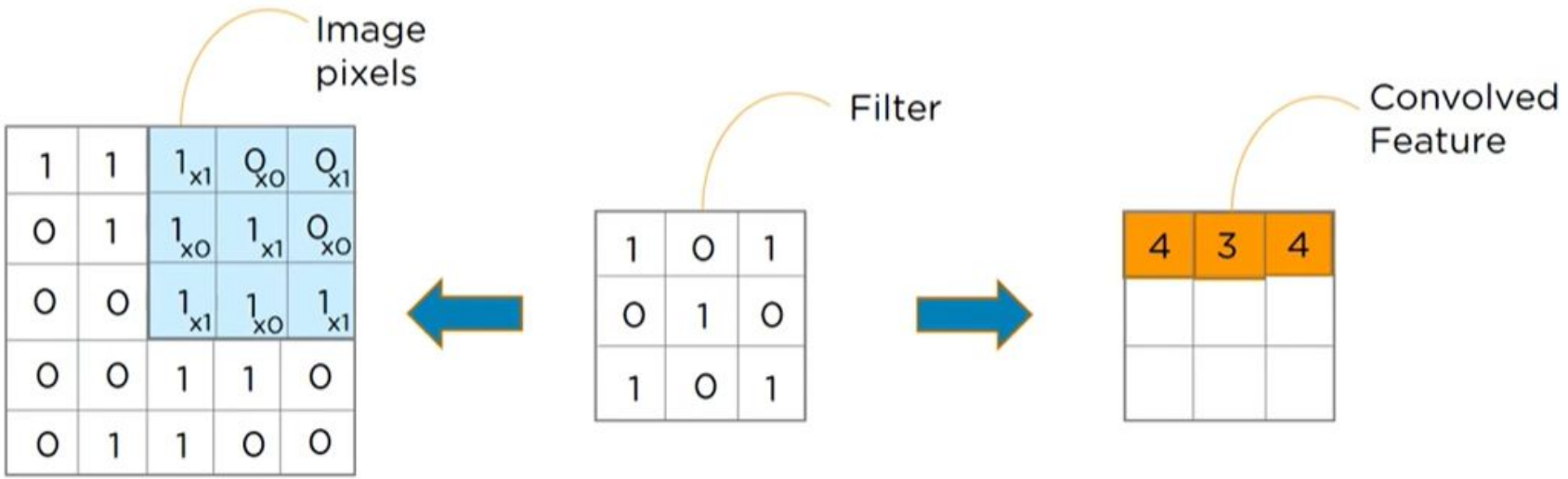


Sliding the filter matrix over the image and computing the dot product to detect patterns

Convolution Layer

A Convolution Layer has a number of filters that perform convolution operation

Every image is considered as a matrix of pixel values.
Consider the following 5*5 image whose pixel values are only 0 and 1

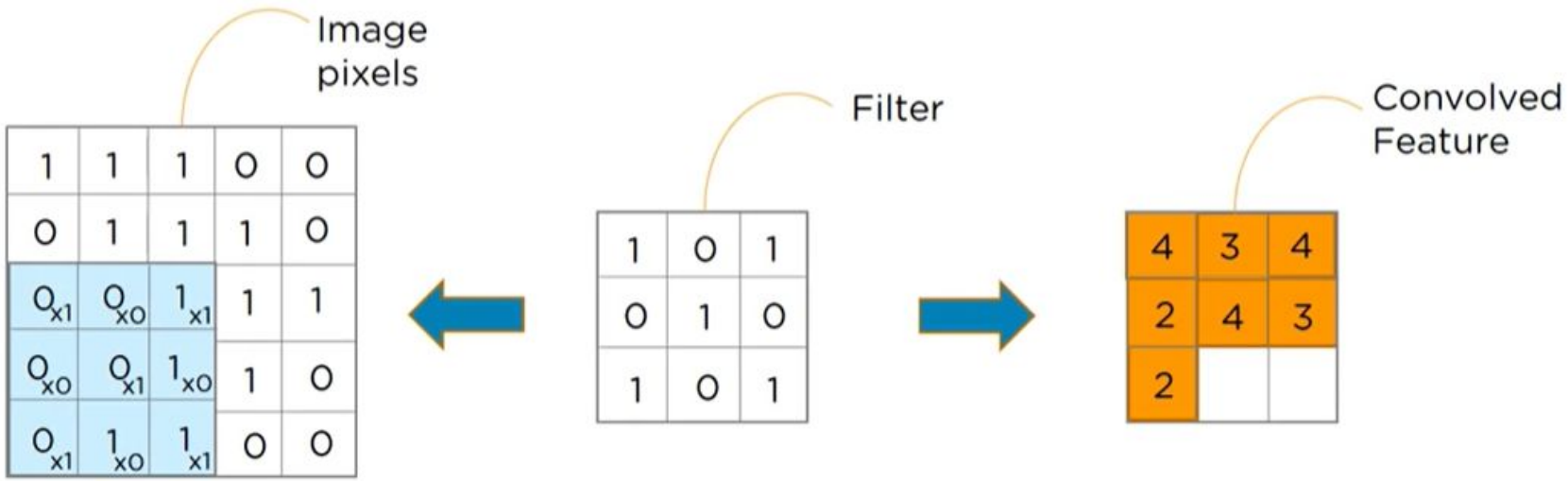


Sliding the filter matrix over the image and computing the dot product to detect patterns

Convolution Layer

A Convolution Layer has a number of filters that perform convolution operation

Every image is considered as a matrix of pixel values.
Consider the following 5*5 image whose pixel values are only 0 and 1

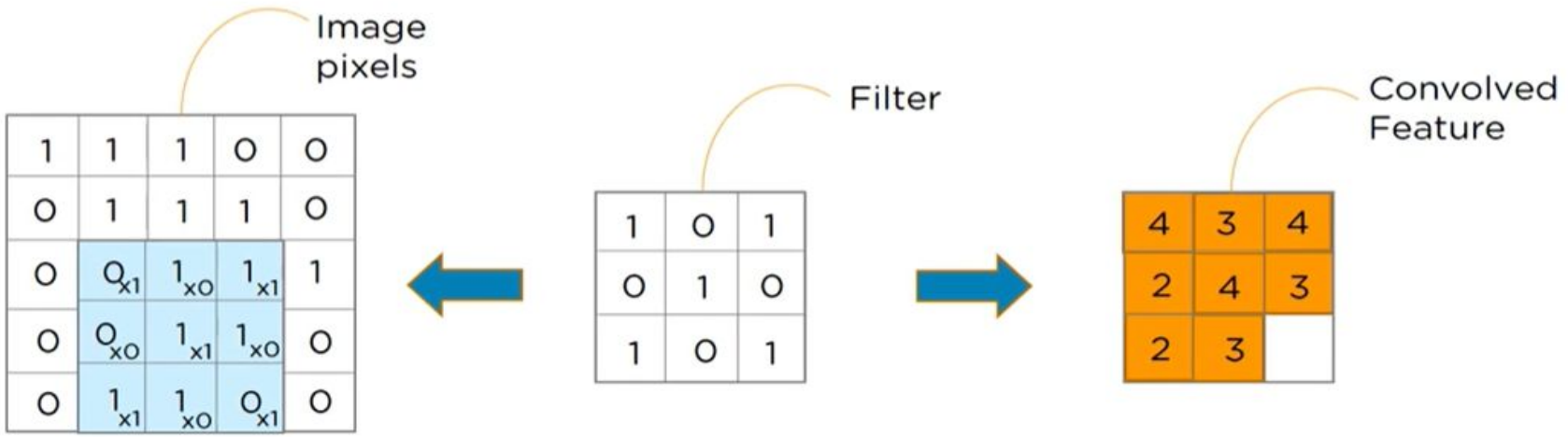


Sliding the filter matrix over the image and computing the dot product to detect patterns

Convolution Layer

A Convolution Layer has a number of filters that perform convolution operation

Every image is considered as a matrix of pixel values.
Consider the following 5*5 image whose pixel values are only 0 and 1

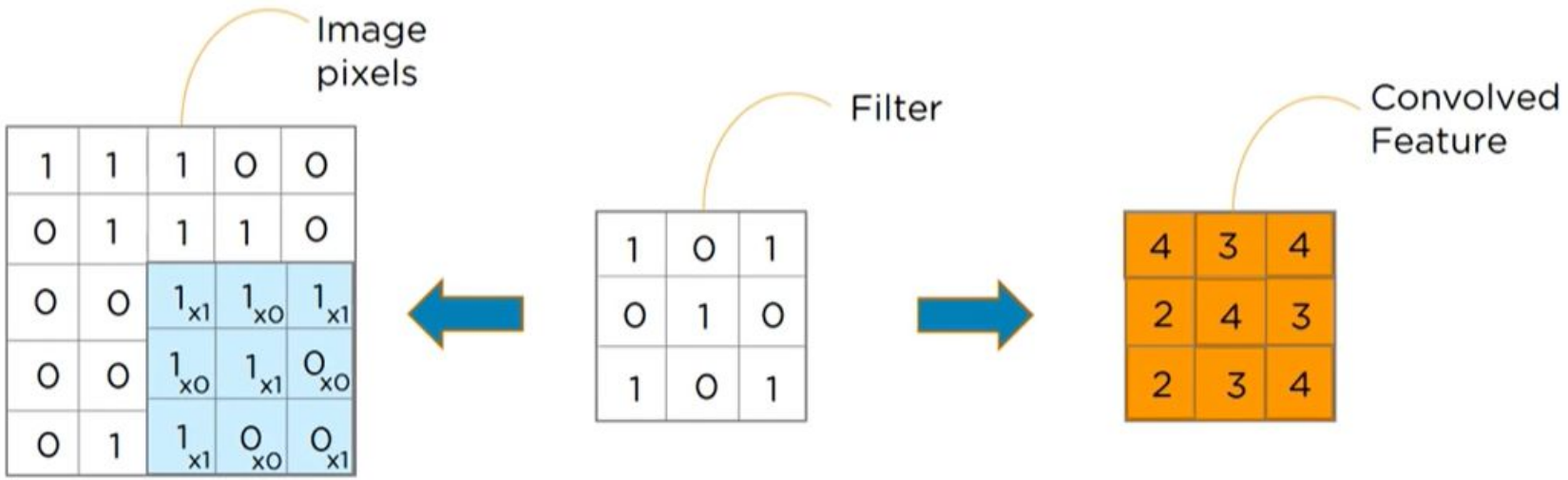


Sliding the filter matrix over the image and computing the dot product to detect patterns

Convolution Layer

A Convolution Layer has a number of filters that perform convolution operation

Every image is considered as a matrix of pixel values.
Consider the following 5*5 image whose pixel values are only 0 and 1

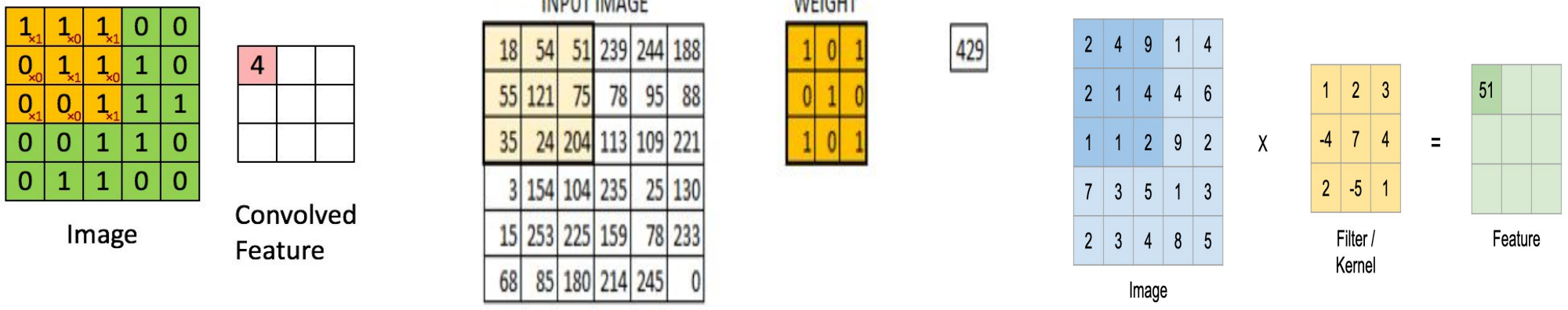


Sliding the filter matrix over the image and computing the dot product to detect patterns

Convolutional Layer (The Conv layer is the core building block of a Convolutional Network that does most of the computational heavy lifting)

Convolution:
Let us suppose this in the input matrix of 5×5 and a filter of matrix 3X3(a filter is a set of weights in a matrix applied on an image or a matrix to obtain the required features), output image after convolution 3X3

We always take the sum or average of all the values while doing a convolution.

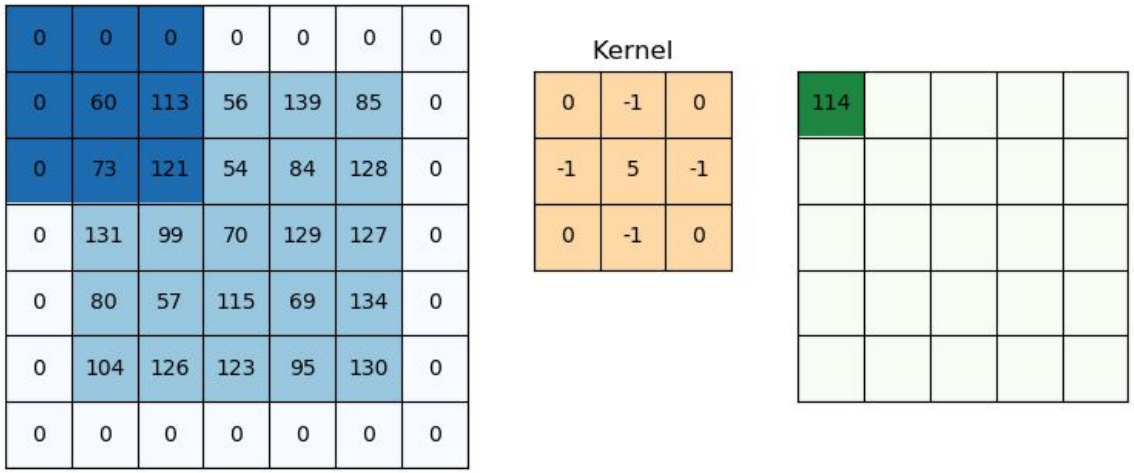


Now t**his filter is also an array of numbers (the numbers are called weights or parameters).**
A very important note is that the depth of this filter has to be the same as the depth of the input (this makes sure that the math works out), so the dimensions of this filter(ex if filter size 5x5) is 5 x 5 x 3 when the input(image size 32x32) is a 32 x 32 x 3 array of pixel values.

Padding Concept:

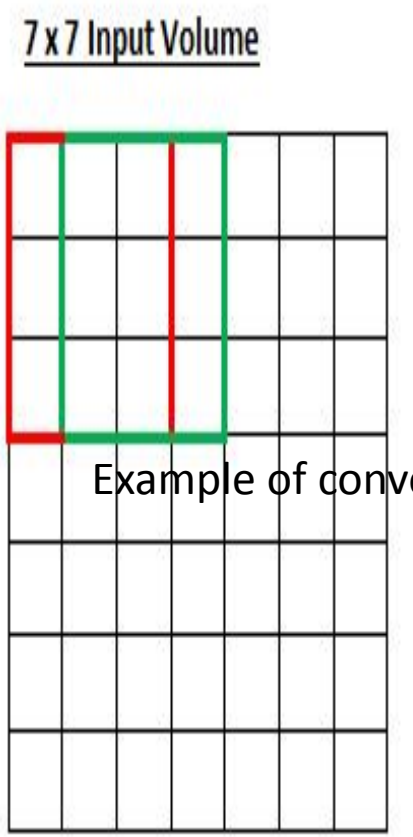
While applying convolutions we will not obtain the output image dimensions the same as input image we. We will lose data over borders.

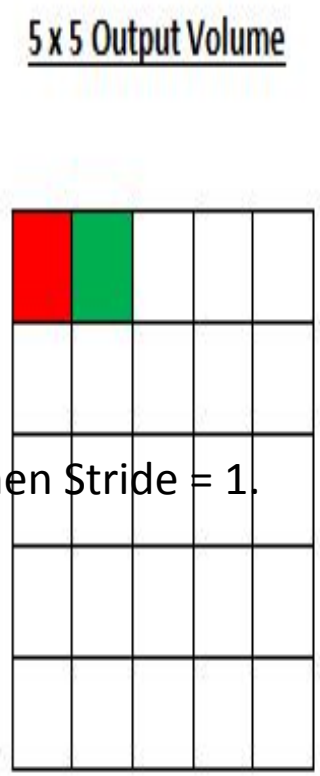
So we append a border of zeros and calculate the convolution covering all the input values.



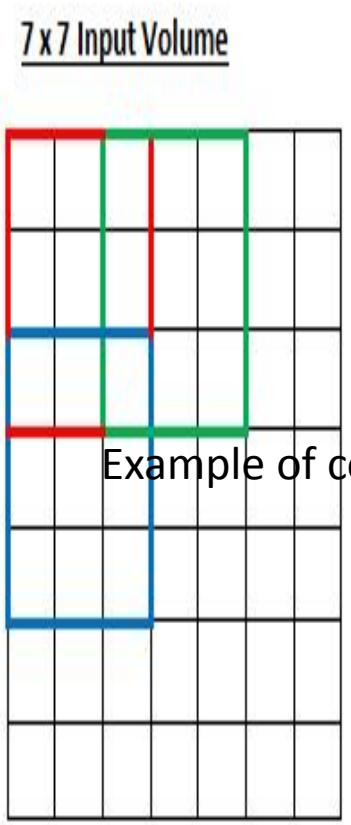
Striding

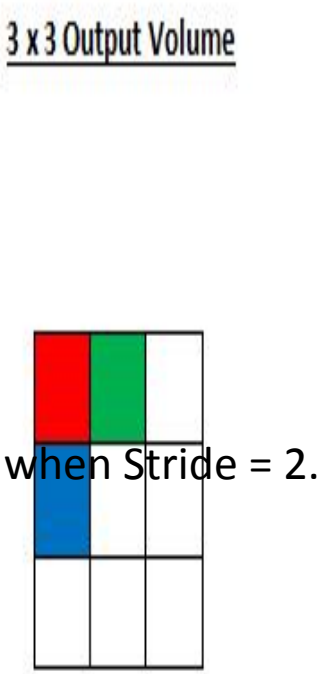
- Stride is the amount of filter shift in the image.
- The bigger the stride, the smaller the feature map in the next layer.





Example of convolution when Stride = 1.





Example of convolution when Stride = 2.

Hyperparameters that control the behavior of each convolved layer:

- Stride
- Padding
- Number of filters (depth (no of images)of the next layer)
- Size of the filter

Number of filters (depth of the next layer)

Example: 6x6x3 image with four 3x3 filters.
After convolving, will get 4x4xn, n depends on the number of filters you use, in other words, it is the number of features detector you use. In this case, n will be 4.
So, we are getting n feature maps/images of 4x4 size.

Size of the filter

The size of the filter is a usually odd number so that the filter has the central pixel/central position of the filter. As if the size of the filter is even, then you need some asymmetric padding.

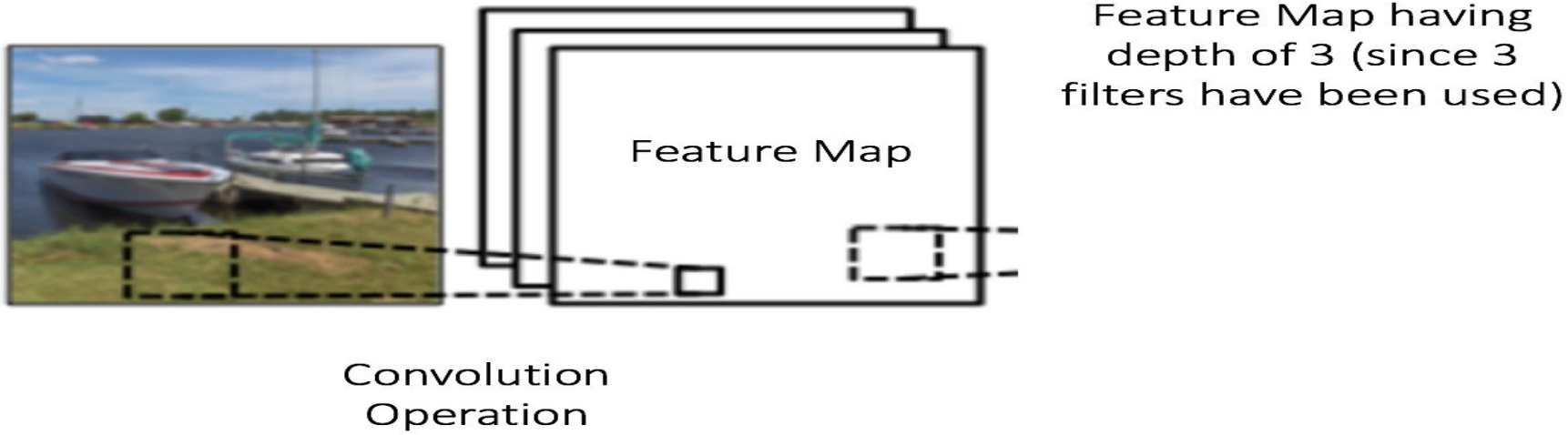
The weights/parameters of the convolutional layer are the values of the filters in the convolutional layer. Those are the learnable values within a CNN.

In practice, a CNN learns the values of these filters on its own during the training process.

But we need to specify parameters such as
number of filters,
filter size,
architecture of the network etc
before the training process.

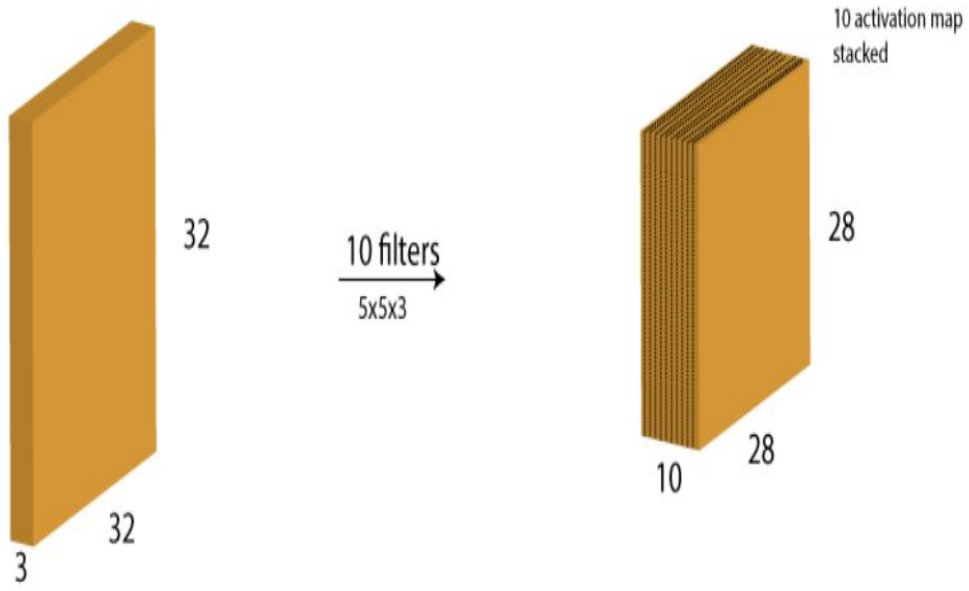
The more number of filters we use, the more image features get extracted and the better our network becomes at recognizing patterns in unseen images.

Depth: Depth corresponds to the number of filters we use for the convolution operation. In the network shown in Figure below, we are performing convolution of the original boat image using three distinct filters, thus producing three different feature maps as shown. You can think of these three feature maps as stacked 2d matrices, so, the ‘depth’ of the feature map would be three.



Multiple filters and the activation map

- One thing to keep in mind is that the depth dimension of the weight would be same as the depth dimension of the input image. The weight extends to the entire depth of the input image.
- Therefore, convolution with a single weight matrix would result into a convolved output with a single depth dimension. In most cases instead of a single filter(weight matrix), we have multiple filters of the same dimensions applied together.
- The output from the each filter is stacked together forming the depth dimension of the convolved image. Suppose we have an input image of size 32*32*3. And we apply 10 filters of size 5*5*3 . The output would have the dimensions as 28*28*10.

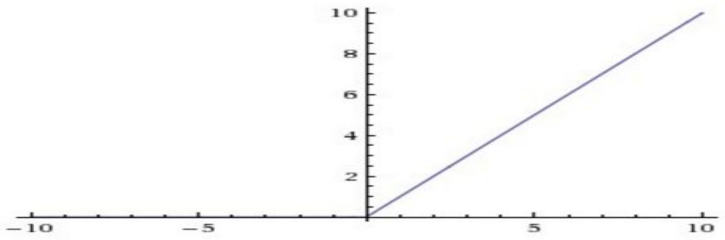


This activation map is the output of the convolution layer.

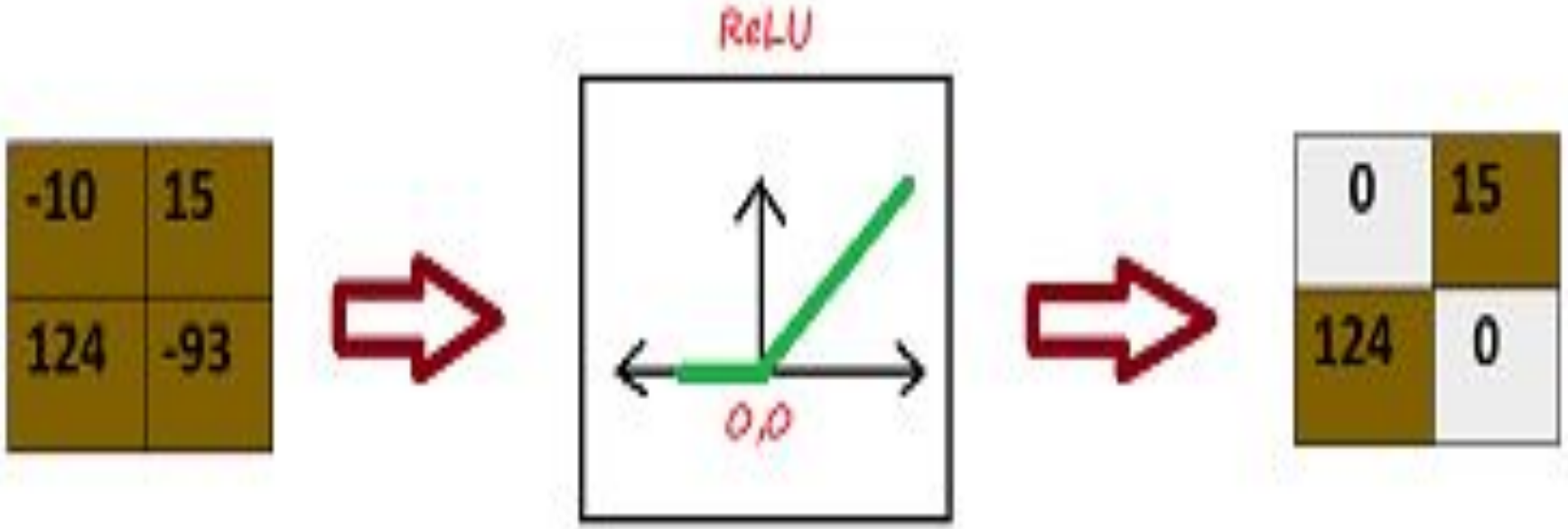
Introducing **Non-linearity** (ReLU)**to the feature maps**

- After convolution is performed, an activation function will be applied to **introduce non-linearity to the feature maps.**
- The reason that we apply non-linearity is that convolution is a linear operation — element-wise matrix multiplication and addition, so we account for non-linearity by introducing a non-linear function like ReLU, prevent the model from computing the linear function, which will be a bad model (because the model will not be able to handle complex non-linear problems).
- An additional operation called ReLU has been used after every Convolution operation. ReLU stands for Rectified Linear Unit and is a non-linear operation. Its output is given by:

$$\text{Output} = \text{Max}(\text{zero}, \text{Input})$$



- ReLU is an element wise operation (applied per pixel) and replaces all negative pixel values in the feature map by zero. The purpose of ReLU is to introduce non-linearity in our ConvNet, since most of the real-world data we would want our ConvNet to learn would be non-linear (Convolution is a linear operation – element wise matrix multiplication and addition, so we account for non-linearity by introducing a non-linear function like ReLU).



- The ReLU operation can be understood clearly from Figure below. It shows the ReLU operation applied to one of the feature maps . The output feature map here is also referred to as the ‘Rectified’ feature map.

Figure ReLU operation

When the ReLu function is applied to the Feature Map, a result as above is produced.Black values in Feature Map are negative. After the Relu function is applied, the black values are removed and 0 is replaced.

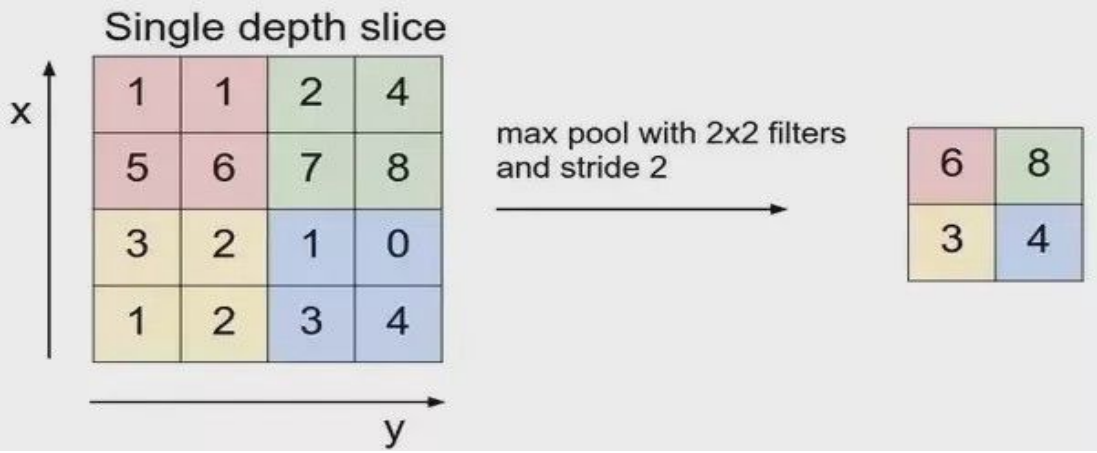
Other non linear functions such as tanh or sigmoid can also be used instead of ReLU, but ReLU has been found to perform better in most situations.

Pooling layer

- Spatial Pooling (also called subsampling or downsampling) reduces the dimensionality of each feature map and retains the most important information of an image. Spatial Pooling can be of different types: Max, Average, Sum, etc.

  In practice, Max Pooling has been shown to work better.

- Max pooling being the most popular. This basically takes a filter (example: size 2x2) and a stride of the same length(which is 2). It then applies it to the input volume and outputs the maximum number in every sub-region that the filter convolves around.
- The pooling layer ***doesn't have any learnable parameters***, hence adding more pooling layers in the CNN doesn't add complexity to the model directly.



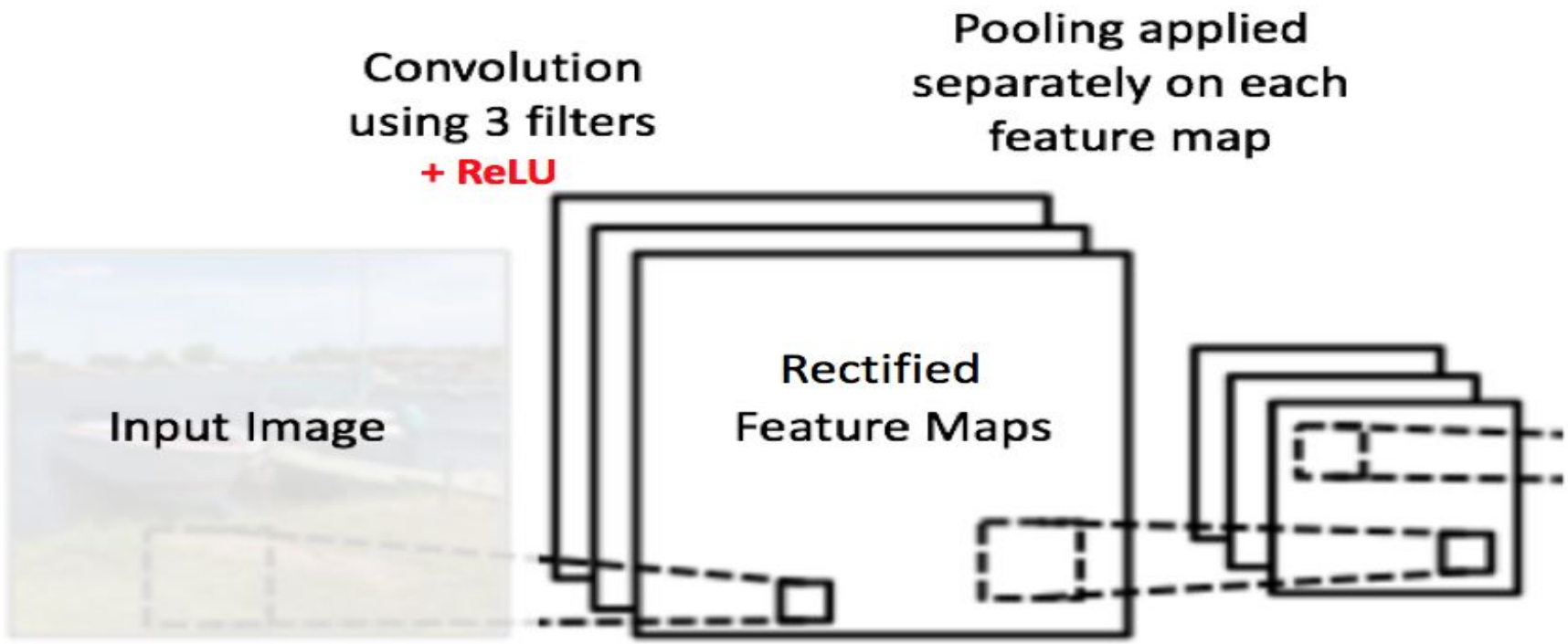
The function of Pooling is to progressively reduce the spatial size(the length and the width change but not the depth) of the input representation.

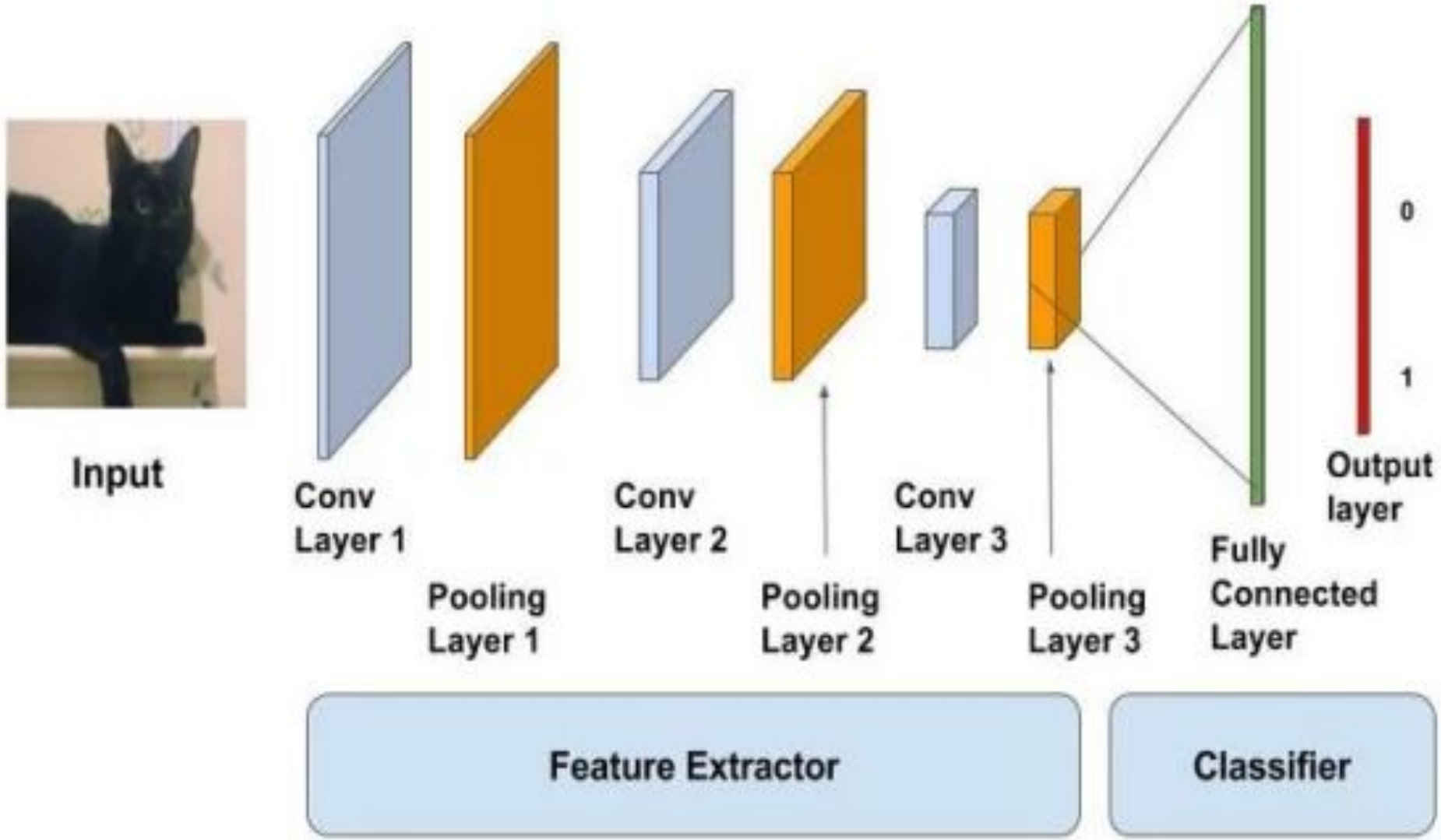
In particular, pooling:
Makes the input representations (feature dimension) smaller and more manageable.

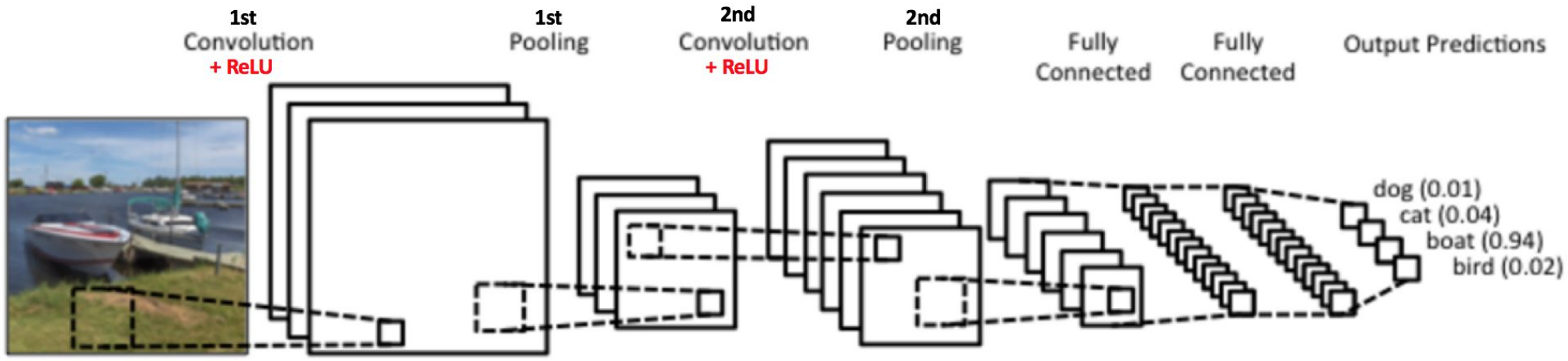
Reduces the number of parameters and computations in the network, therefore, controlling overfitting.

Makes the network invariant to small transformations, distortions, and translations in the input image (a small distortion in the input will not change the output of Pooling — since we take the maximum/average value in a local neighborhood).

pooling operation is applied separately to each feature map (notice that, due to this, we get three output maps from three input maps).




Input
Conv Layer 1
Pooling Layer 1
Conv Layer 2
Pooling Layer 2
Conv Layer 3
Pooling Layer 3
Fully Connected Layer
0
1
Output layer
Feature Extractor
Classifier



So far we have seen how Convolution, ReLU and Pooling work. It is important to understand that these layers are the basic building blocks of any CNN. As shown in above Figure , we have two sets of Convolution, ReLU & Pooling layers – the 2nd Convolution layer performs convolution on the output of the first Pooling Layer using six filters to produce a total of six feature maps. ReLU is then applied individually on all of these six feature maps. We then perform Max Pooling operation separately on each of the six rectified feature maps.

Together these layers extract the useful features from the images, introduce non-linearity in our network and reduce feature dimension while aiming to make the features somewhat equivariant to scale and translation [18].

The output of the 2nd Pooling Layer acts as an input to the Fully Connected Layer, which we will discuss in the next section.

Output dimensions at the end of each convolution layer

- Let us understand the input and output dimensions at the end of each convolution layer. Three hyperparameter would control the size of output volume.
- The number of filters – the depth of the output volume will be equal to the number of filter applied. Remember how we had stacked the output from each filter to form an activation map. The depth of the activation map will be equal to the number of filters.
- Stride – When we have a stride of one we move across and down a single pixel. With higher stride values, we move large number of pixels at a time and hence produce smaller output volumes.
- Zero padding – This helps us to preserve the size of the input image. If a single zero padding is added, a single stride filter movement would retain the size of the original image.
-

Output dimensions contd..

- We can apply a simple formula to calculate the output dimensions. The spatial size of the output image can be calculated as [W-F+2P]/S)+1.
- Here, W is the input volume size, F is the size of the filter, P is the number of padding applied and S is the number of strides. Suppose we have an input image of size 32*32*3, we apply 10 filters of size 3*3*3, with single stride and no zero padding.
-

Example

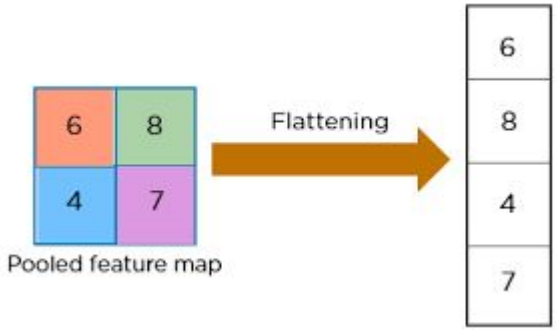
- Here W=32, F=3, P=0 and S=1.

- The size of the output volume will be ([32-3+0]/1)+1 = 30.
- Therefore the output volume will be 30*30*10.
- The output depth will be equal to the number of filters applied i.e. 10.

Flattening

- It is used to convert the data into 1D arrays to create a single feature vector. After flattening we forward the data to a fully connected layer for final classification.

- The output of the convolutional part of the CNN is converted into a 1D feature vector, to be used by the Fully connected layer.
- This operation is called flattening.
- It gets the output of the convolutional layers, flattens all its structure to create a single long feature vector to be used by the fully connected layer for the final classification.



The Output layer

- After multiple layers of convolution and padding, we would need the output in the form of a class.
- The convolution and pooling layers would only be able to extract features and reduce the number of parameters from the original images.
- However, to generate the final output we need to apply a fully connected layer to generate an output equal to the number of classes we need.
- Convolution layers generate 3D activation maps while we just need the output as whether or not an image belongs to a particular class.
- The output layer has a loss function like categorical cross-entropy, to compute the error in prediction.
- Once the forward pass is complete the backpropagation begins to update the weight and biases for error and loss reduction.

Fully Connected Layer

The Fully Connected layer is a traditional Multi Layer Perceptron that uses a softmax activation function in the output layer (other classifiers like SVM can also be used,).

The term “Fully Connected” implies that every neuron in the previous layer is connected to every neuron on the next layer.

The output from the convolutional and pooling layers represent high-level features of the input image.

The purpose of the Fully Connected layer is to use these features for classifying the input image into various classes based on the training dataset.

Softmax Function

- This one of the methods that we can use to perform classification in neural networks.
- The output from the Fully Connected Layer is then passed through the softmax function.
- The Softmax function takes a vector of arbitrary real-valued scores and squashes it to a vector of values between zero and one that sums to one.

Fully Connected Layer

This layer basically takes an input volume (whatever the output is of the conv or ReLU or pool layer preceding it) and outputs an N dimensional vector where N is the number of classes that the program has to choose from.

For example, if you wanted a digit classification program, N would be 10 since there are 10 digits. Each number in this N dimensional vector represents the probability of a certain class.

Fully Connected Layer… contd

For example, if the resulting vector for a digit classification program is [0 .1 .1 .75 0 0 0 0 0 .05], then this represents a 10% probability that the image is a 1, a 10% probability that the image is a 2, a 75% probability that the image is a 3, and a 5% probability that the image is a 9 (Side note: There are other ways that you can represent the output, but just showing the softmax approach).

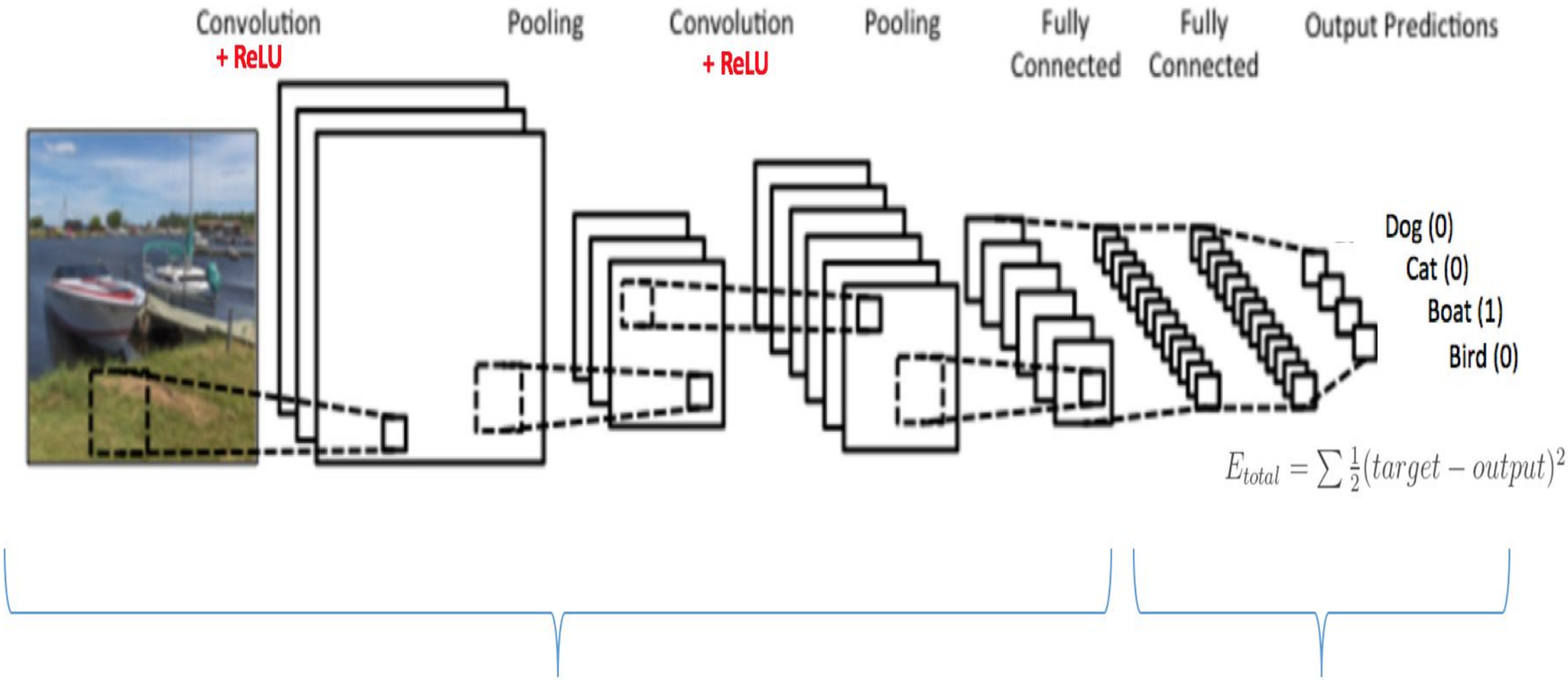
The way this fully connected layer works is that it looks at the output of the previous layer and determines which features most correlate to a particular class

Putting it all together

- The Convolution + Pooling layers act as Feature Extractors from the input image while Fully Connected layer acts as a classifier.
- Note that in Figure 5 below, since the input image is a boat, the target probability is 1 for Boat class and 0 for other three classes, i.e.

  Input Image = Boat

  Target Vector = [0, 0, 1, 0]


Convolution
+ ReLU
Pooling
Convolution
+ ReLU
Pooling
Fully
Connected
Fully
Connected
Output Predictions
Dog (0)
Cat (0)
Boat (1)
Bird (0)
$E_{total} = \sum \frac{1}{2}(target - output)^2$
Feature Extraction from Image
Classification

The overall training process of the Convolution Network may be summarized as below:

Step1: We initialize all filters and parameters / weights with random values

Step2: The network takes a training image as input, goes through the forward propagation step (convolution, ReLU and pooling operations along with forward propagation in the Fully Connected layer) and finds the output probabilities for each class.
Lets say the output probabilities for the boat image above are [0.2, 0.4, 0.1, 0.3]
Since weights are randomly assigned for the first training example, output probabilities are also random.

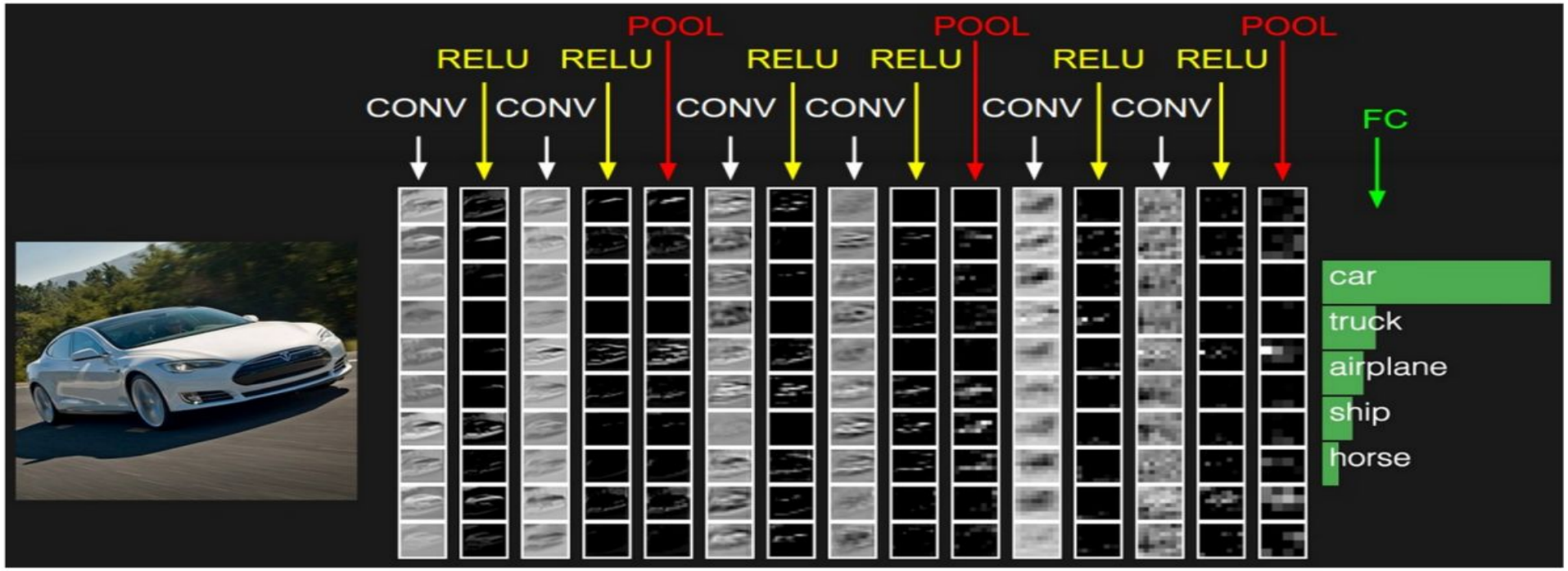
Step3: Calculate the total error at the output layer (summation over all 4 classes)
Total Error = $\sum$ ½ (target probability – output probability) ^{2}

Step4: Use Backpropagation to calculate the gradients of the error with respect to all weights in the network and use gradient descent to update all filter values / weights and parameter values to minimize the output error.

- The weights are adjusted in proportion to their contribution to the total error.
- When the same image is input again, output probabilities might now be [0.1, 0.1, 0.7, 0.1], which is closer to the target vector [0, 0, 1, 0].
- This means that the network has learnt to classify this particular image correctly by adjusting its weights / filters such that the output error is reduced.
- Parameters like number of filters, filter sizes, architecture of the network etc. have all been fixed before Step 1 and do not change during training process – only the values of the filter matrix and connection weights get updated.

Step5: Repeat steps 2-4 with all images in the training set.

Note : In the example above we used two sets of alternating Convolution and Pooling layers. Please note however, that these operations can be repeated any number of times in a single ConvNet. In fact, some of the best performing ConvNets today have tens of Convolution and Pooling layers! Also, it is not necessary to have a Pooling layer after every Convolutional Layer. As can be seen in the Figure 6 below, we can have multiple Convolution + ReLU operations in succession before a having a Pooling operation. Also notice how each layer of the ConvNet is visualized in the Figure below.



Underfitting and Overfitting.

- The factors determining how well a machinelearning algorithm will perform are its ability to

1. Make the training error small.
2. Make the gap between training and test error small.

These two factors correspond to the two central challenges in machine learning:underfitting and overfitting.

Underfitting occurs when the model is not able to obtain a sufficiently low error value on the training set.

Overfitting occurs when the gap between the training error and test error is too large.

Applications

CNNs have wide applications in

- image and video recognition,
- Image classification
- Object detection
- recommender systems and
- natural language processing.

Case studies

There are several architectures in the field of Convolutional Networks that have a name. The most common are:

- **LeNet.** The first successful applications of Convolutional Networks were developed by Yann LeCun in 1990's. Of these, the best known is the LeNet architecture that was used to read zip codes, digits, etc.
- **AlexNet.** The first work that popularized Convolutional Networks in Computer Vision was the AlexNet, developed by Alex Krizhevsky, Ilya Sutskever and Geoff Hinton. The AlexNet was submitted to the ImageNet ILSVRC challenge in 2012 and significantly outperformed the second runner-up (top 5 error of 16% compared to runner-up with 26% error). The Network had a very similar architecture to LeNet, but was deeper, bigger, and featured Convolutional Layers stacked on top of each other (previously it was common to only have a single CONV layer always immediately followed by a POOL layer).
- **ZF Net**. The ILSVRC 2013 winner was a Convolutional Network from Matthew Zeiler and Rob Fergus. It became known as the ZFNet (short for Zeiler & Fergus Net). It was an improvement on AlexNet by tweaking the architecture hyperparameters, in particular by expanding the size of the middle convolutional layers and making the stride and filter size on the first layer smaller.
- **GoogLeNet**. The ILSVRC 2014 winner was a Convolutional Network from Szegedy et al. from Google. Its main contribution was the development of an Inception Module that dramatically reduced the number of parameters in the network (4M, compared to AlexNet with 60M). Additionally, this paper uses Average Pooling instead of Fully Connected layers at the top of the ConvNet, eliminating a large amount of parameters that do not seem to matter much. There are also several followup versions to the GoogLeNet, most recently Inception-v4.

Case studies (contd:)

- **VGGNet**. The runner-up in ILSVRC 2014 was the network from Karen Simonyan and Andrew Zisserman that became known as the VGGNet. Its main contribution was in showing that the depth of the network is a critical component for good performance. Their final best network contains 16 CONV/FC layers and, appealingly, features an extremely homogeneous architecture that only performs 3x3 convolutions and 2x2 pooling from the beginning to the end. Their pretrained model is available for plug and play use in Caffe. A downside of the VGGNet is that it is more expensive to evaluate and uses a lot more memory and parameters (140M). Most of these parameters are in the first fully connected layer, and it was since found that these FC layers can be removed with no performance downgrade, significantly reducing the number of necessary parameters.
- **ResNet**. Residual Network developed by **Kaiming He et al**. was the winner of ILSVRC 2015. It features special skip connections and a heavy use of batch normalization. The architecture is also missing fully connected layers at the end of the network. The reader is also referred to Kaiming's presentation (video, slides), and some recent experiments that reproduce these networks in Torch. ResNets are currently by far state of the art Convolutional Neural Network models and are the default choice for using ConvNets in practice (as of May 10, 2016). In particular, also see more recent developments that tweak the original architecture from Kaiming He et al. Identity Mappings in Deep Residual Networks (published March 2016).

References:

- https://towardsdatascience.com/beginners-guide-for-convolutional-neural-network-cnn-convnets-5a5e725ea581