iTrain (M) Sdn Bhd
KL: C-19-8, KL Trillion, 338 Jalan Tun Razak, 50400 Kuala Lumpur. Tel: +603 2733 0337
Website: www.itrain.com.my

Convolution Neural Networks

By: Faris Hassan

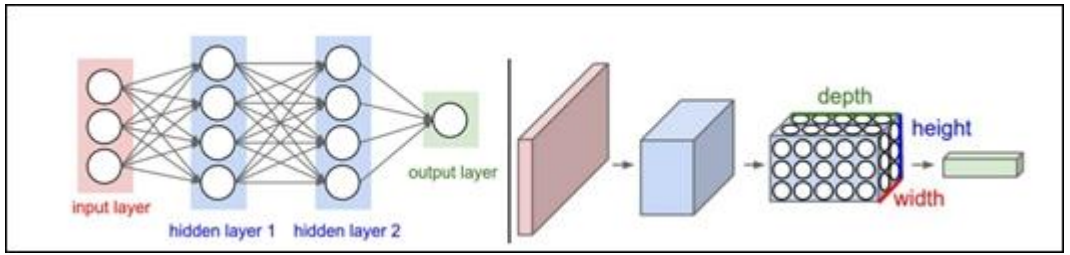
Recap

Regular neural networks have a single vector-based input that is transformed through a series of hidden layers where neurons in each layer are connect with the neurons in its neighbouring layers.

The last layer in the series provides the output and called the output layer. When the input to the neural network is an image and doesn't fit into a single vector structure, the complexity grows.

CNN have a slight variation where the input assumed to be a 3-d vector with depth (D), height (H), and width (W).

This assumption changes the way neural networks organized and the way it functions.



Why convnets instead of fully connected layers.

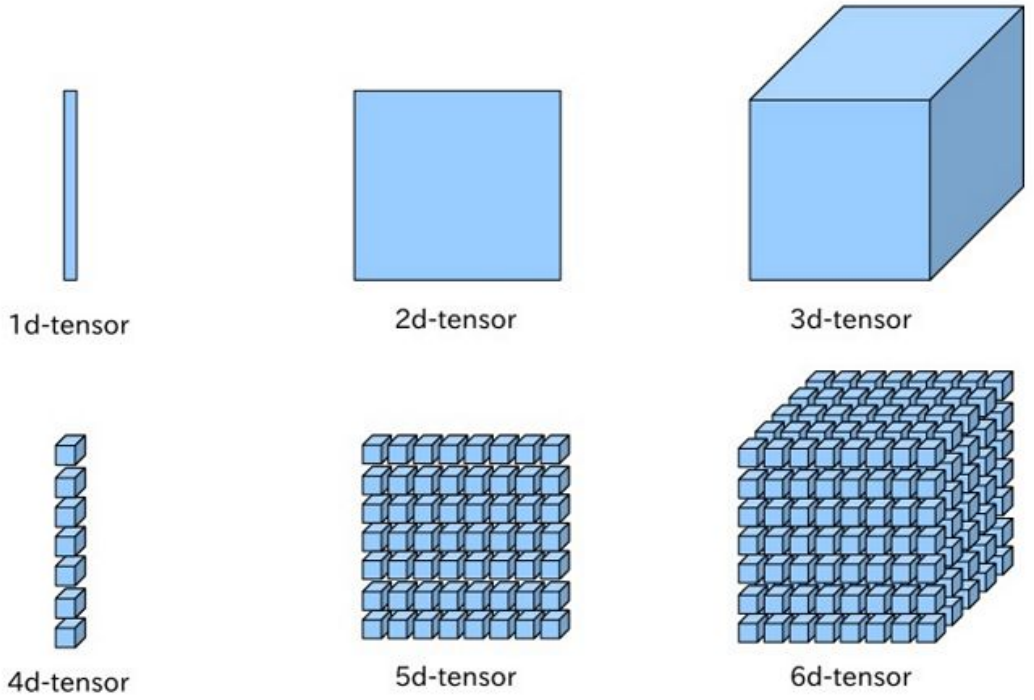
Fully connected layers doesn't take into account the spatial structures of the images. For instance, it treats input pixels which are far apart and close together on exactly the same footing. Such concepts of spatial structure must instead be inferred from the training data.

But what if, instead of starting with a network architecture which is tabula rasa, we used an architecture which tries to take advantage of the spatial structure?

Convnets use a special architecture which is particularly well-adapted to classify images. Using this architecture makes convolutional networks fast to train. This, in turn, helps us train deep, many-layer networks, which are very good at classifying images. Today, deep convolutional networks or some close variant are used in most neural networks for image recognition.

Tensors

A tensor is a multidimensional array. A cube of multiple arrays.



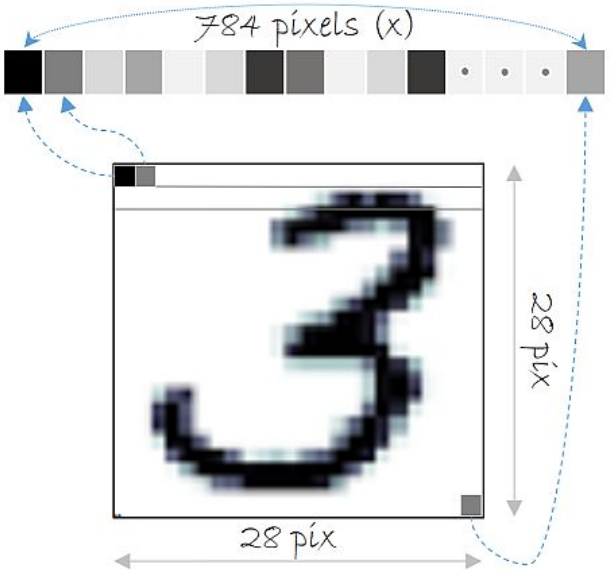
Basic Ideas of Convnets

1. Local Receptive fields.
2. Shared weights.
3. pooling

Local Receptive Fields

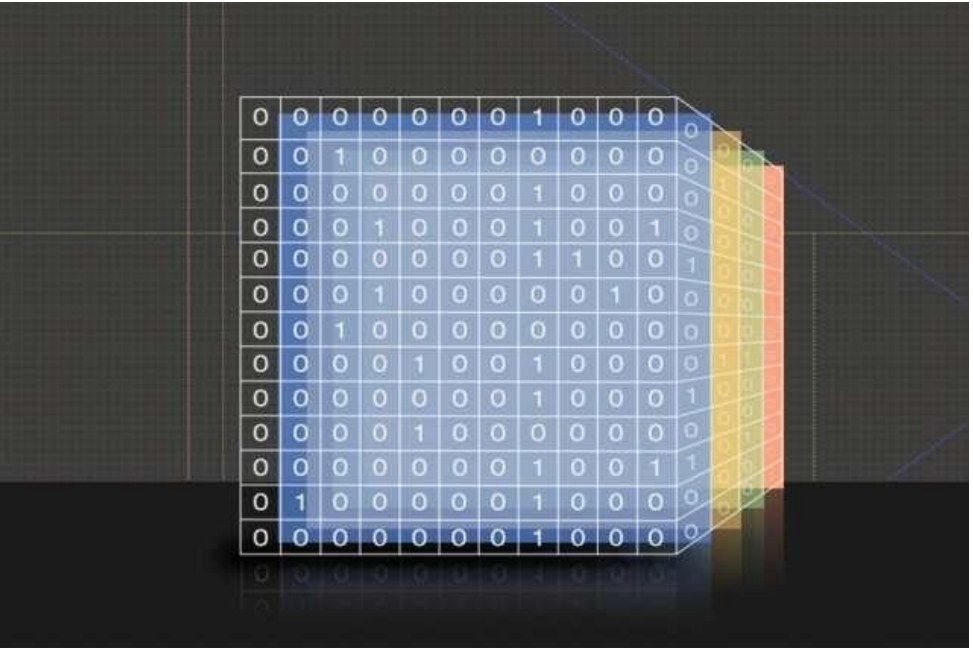
In the fully-connected layers shown earlier, the inputs were depicted as a vertical line of neurons. In a convolutional net, it'll help to think instead of the inputs as a 28×28 square of neurons, whose values correspond to the 28×28 pixel intensities we're using as inputs.

As per usual, we'll connect the input pixels to a layer of hidden neurons. But we won't connect every input pixel to every hidden neuron. Instead, we only make connections in small, localized regions of the input image.

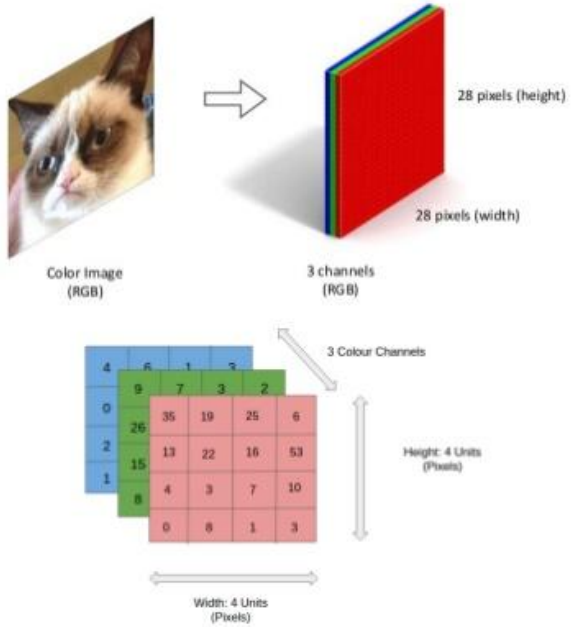


Convnets instead of fully connected layers

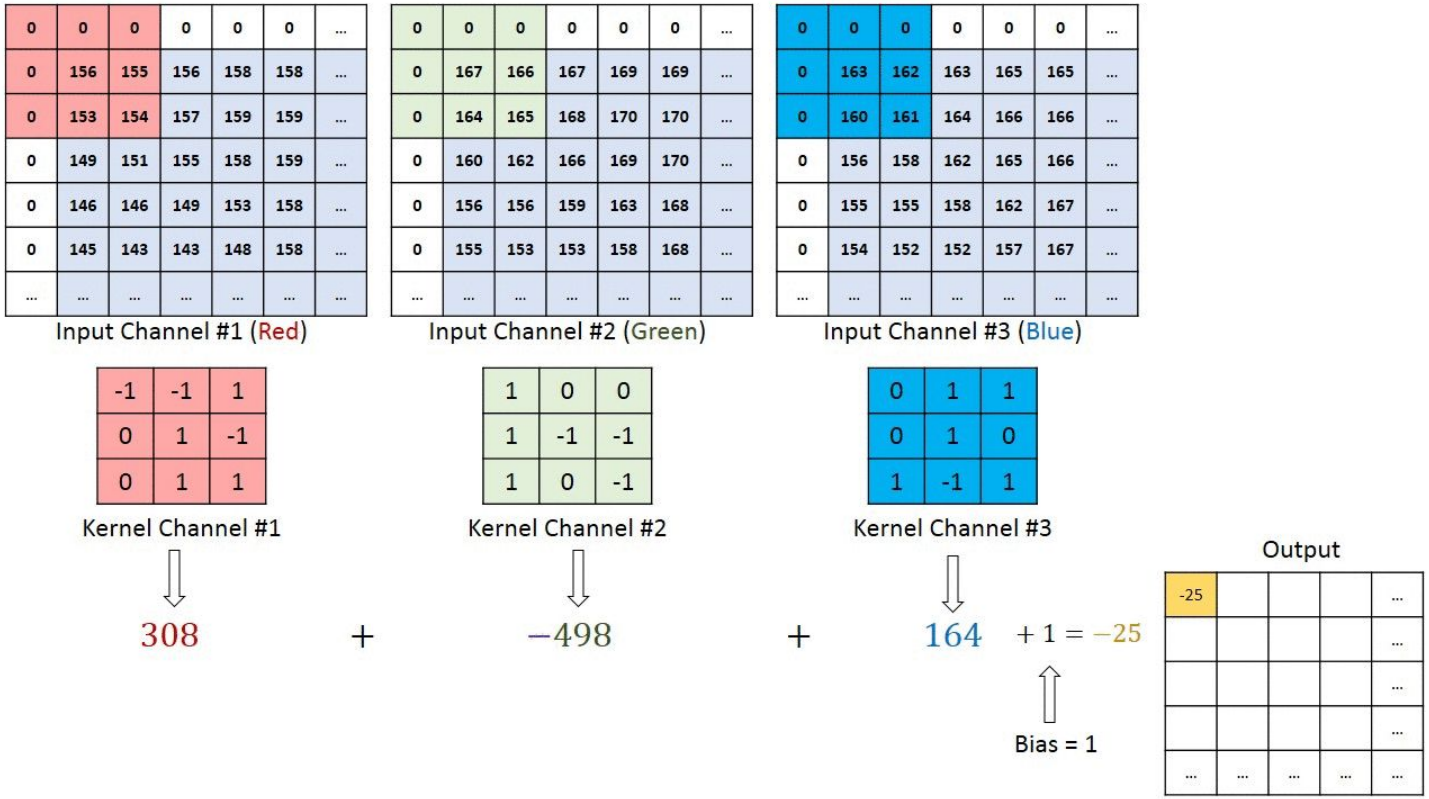
Preserving the spatial features.



color image is 3rd-order tensor

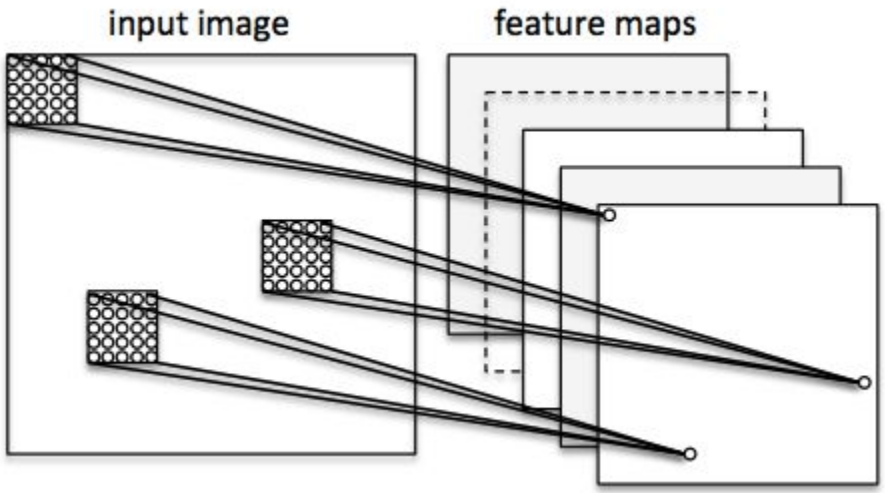


The Convolution layer (kernel movement)



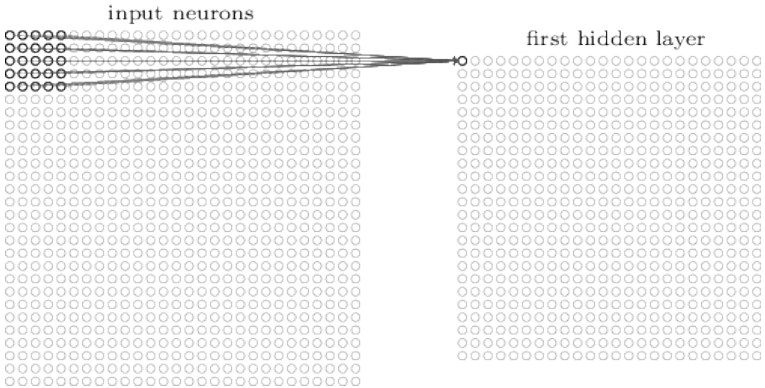
Local Receptive Fields

To be more precise, each neuron in the first hidden layer will be connected to a small region of the input neurons, say, for example, a 5×5 region, corresponding to 25 input pixels. So, for a particular hidden neuron, we might have connections that look like this:

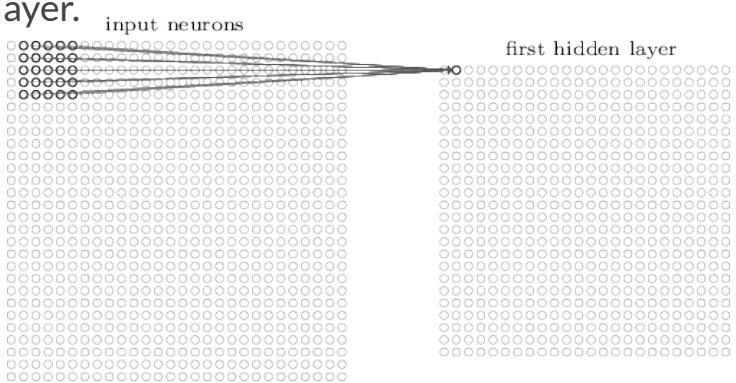


Local Receptive Fields

We then slide the local receptive field across the entire input image. For each local receptive field, there is a different hidden neuron in the first hidden layer. To illustrate this concretely, let's start with a local receptive field in the top-left corner:



Then we slide the local receptive field over by one pixel to the right (i.e., by one neuron), to connect to a second hidden neuron And so on, building up the first hidden layer. Note that if we have a 28×28 input image, and 5×5 local receptive fields, then there will be 24×24 neurons in the hidden layer.



Shared weights and biases

I've said that each hidden neuron has a bias and 5×5 weights connected to its local receptive field. What I did not yet mention is that we're going to use the *same* weights and bias for each of the 24×24 hidden neurons.

This means that all the neurons in the first hidden layer detect exactly the same feature* , just at different locations in the input image. To see why this makes sense, suppose the weights and bias are such that the hidden neuron can pick out, say, a vertical edge in a particular local receptive field

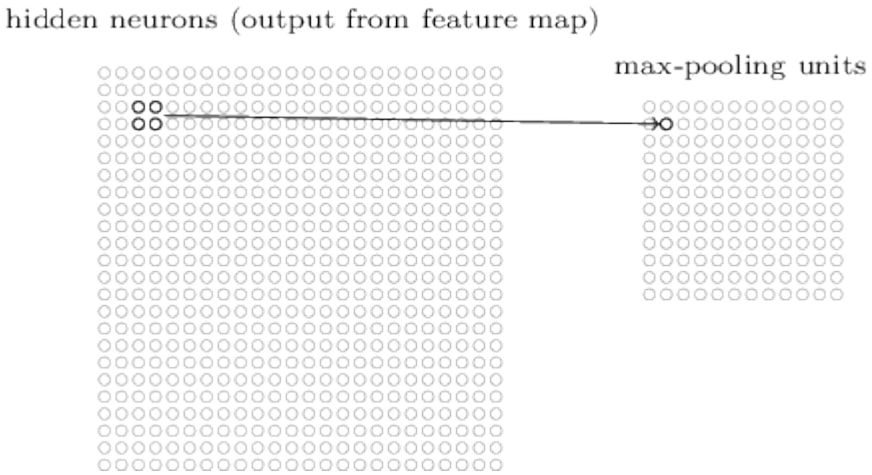
That ability is also likely to be useful at other places in the image. And so it is useful to apply the same feature detector everywhere in the image. To put it in slightly more abstract terms, convolutional networks are well adapted to the translation invariance of images: move a picture of a cat (say) a little ways, and it's still an image of a cat.

*Informally, think of the feature detected by a hidden neuron as the kind of input pattern that will cause the neuron to activate: it might be an edge in the image, for instance, or maybe some other type of shape.

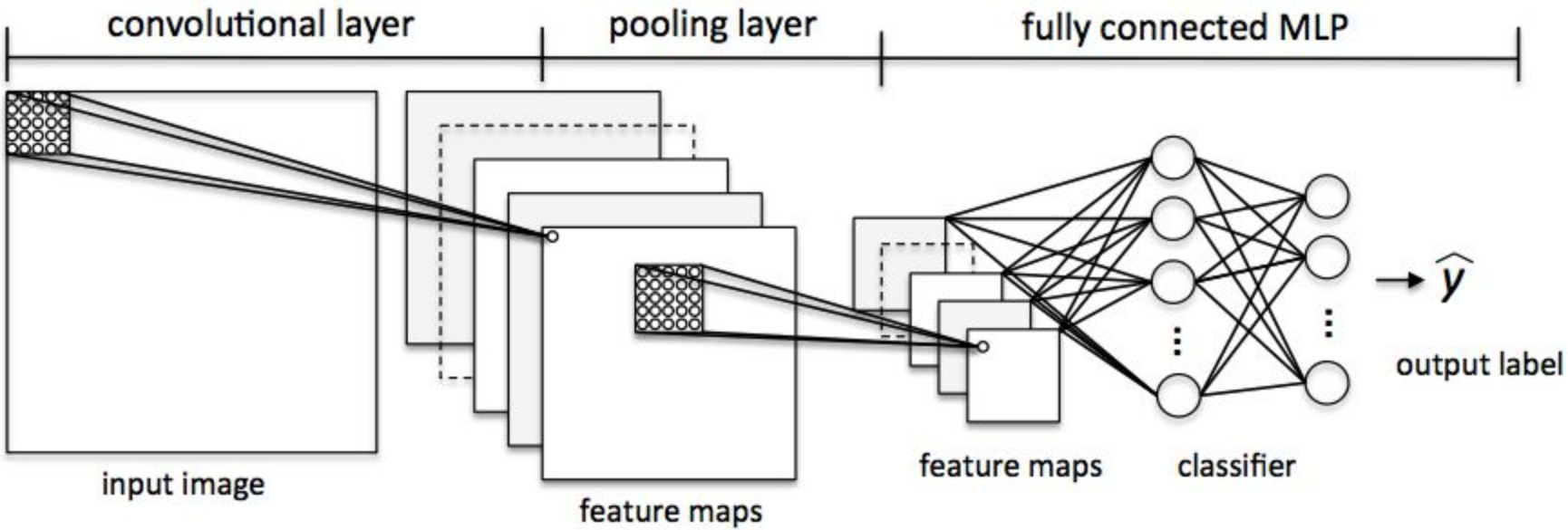
Pooling Layers

Pooling layers are usually used immediately after convolutional layers. What the pooling layers do is simplify the information in the output from the convolutional layer.

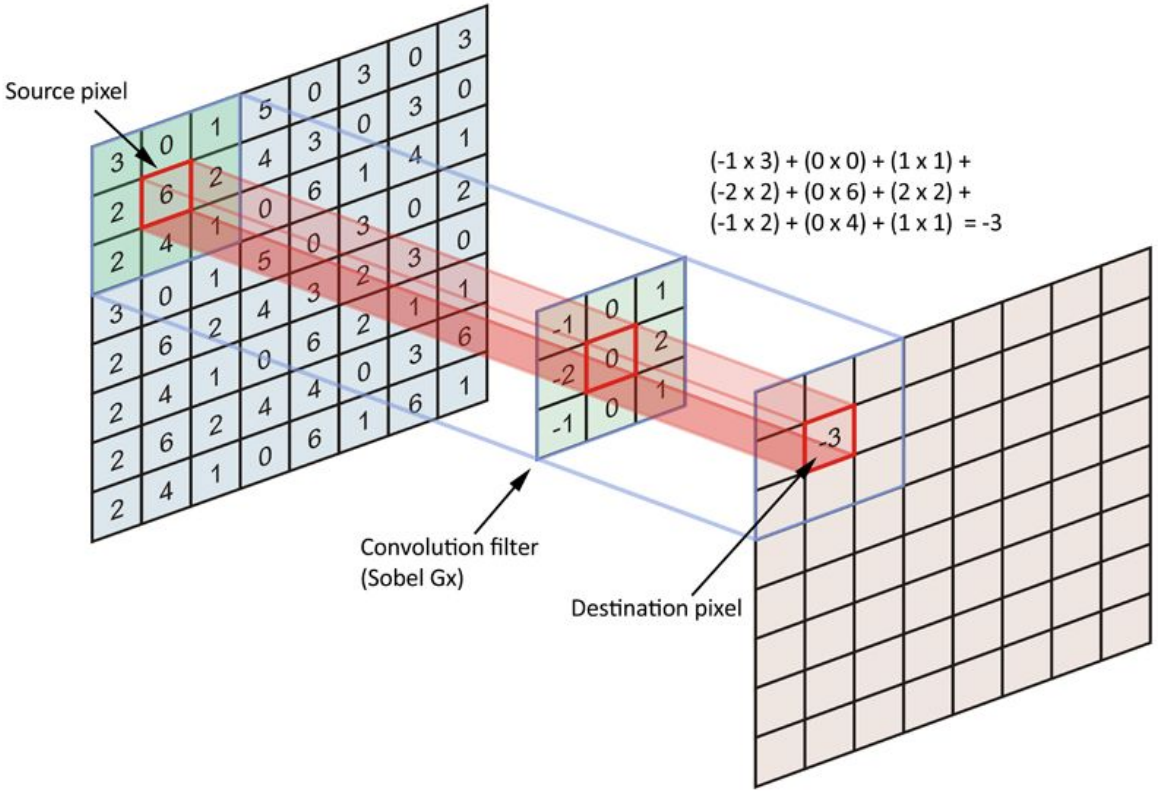
In other words it down samples the features in the feature map.



Typical Convnet Architecture



How convolution work?

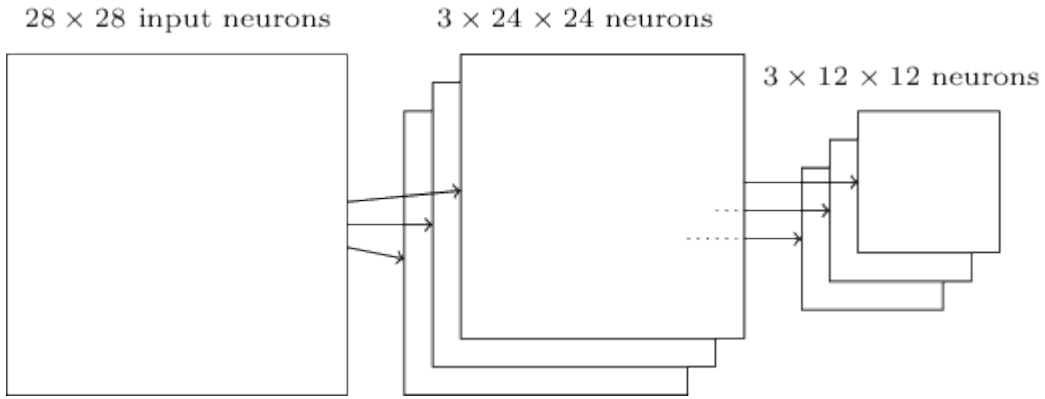


1x1	1x0	1x1	0	0
0x0	1x1	1x0	1	0
0x1	0x0	1x1	1	1
0	0	1	1	0
0	1	1	0	0

4		

Feature Maps

the convolutional layer usually involves more than a single feature map. We apply max-pooling to each feature map separately. So if there were three feature maps, the combined convolutional and max-pooling layers would look like:

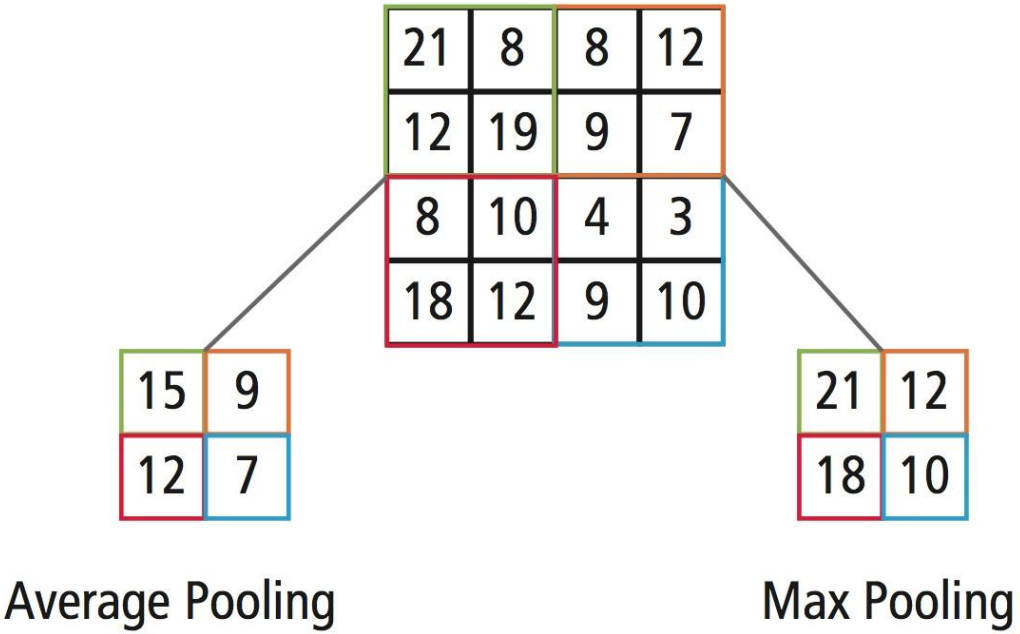


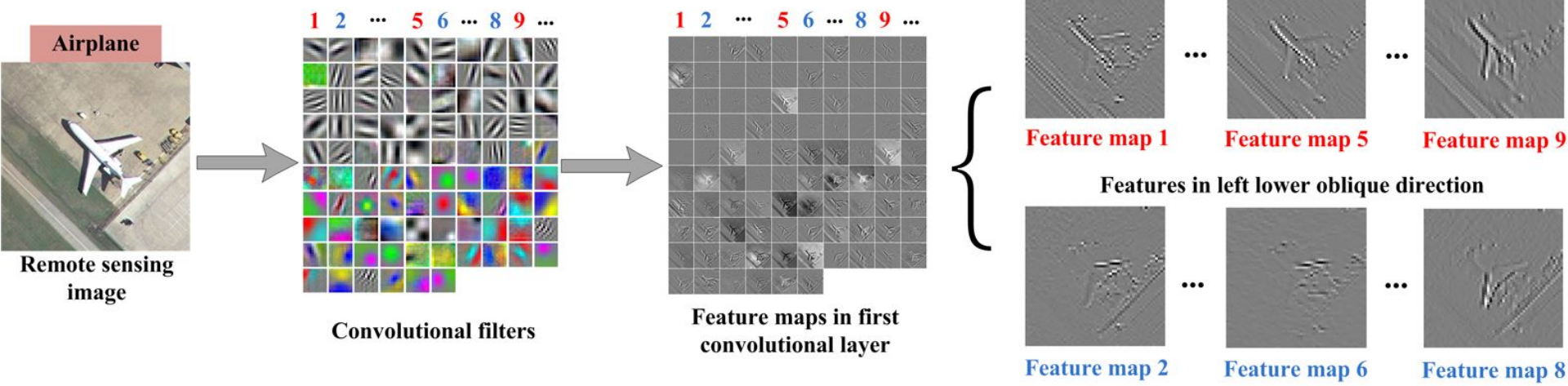
Pooling Techniques

Max Pooling: taking the maximum activation of a 2×2 region of neurons.

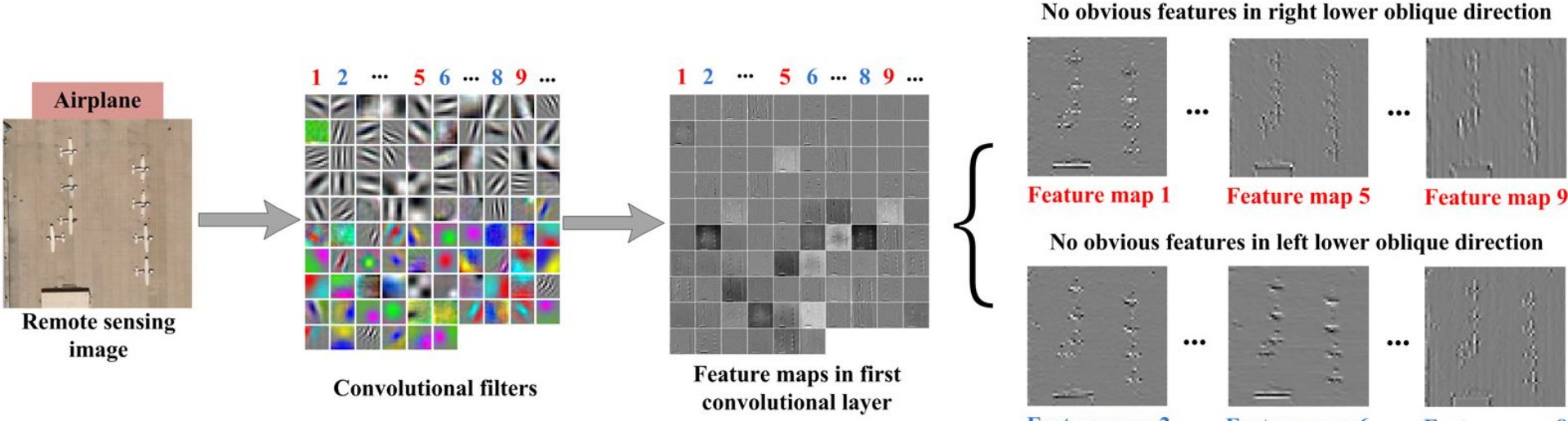
L2 Pooling: taking the square root of the sum of squares in the activation region.

Average Pooling: Taking the average of the values in the activation region.



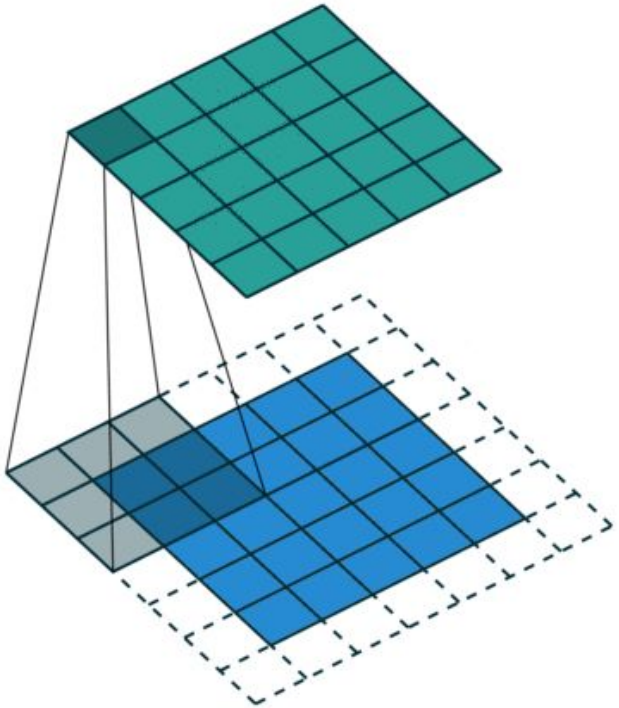

Airplane
Remote sensing image
1 2 ··· 5 6 ··· 8 9 ···
Convolutional filters
1 2 ··· 5 6 ··· 8 9 ···
Feature maps in first convolutional layer
Feature map 1
Feature map 5
Feature map 9
Features in left lower oblique direction
Feature map 2
Feature map 6
Feature map 8

(a)


Airplane
Remote sensing image
1 2 ··· 5 6 ··· 8 9 ···
Convolutional filters
1 2 ··· 5 6 ··· 8 9 ···
Feature maps in first convolutional layer
No obvious features in right lower oblique direction
Feature map 1
Feature map 5
Feature map 9
No obvious features in left lower oblique direction

Strides

The stride defines the step size of the kernel when traversing the image. While its default is usually 1, we can use a stride of 2 for downsampling an image similar to MaxPooling.

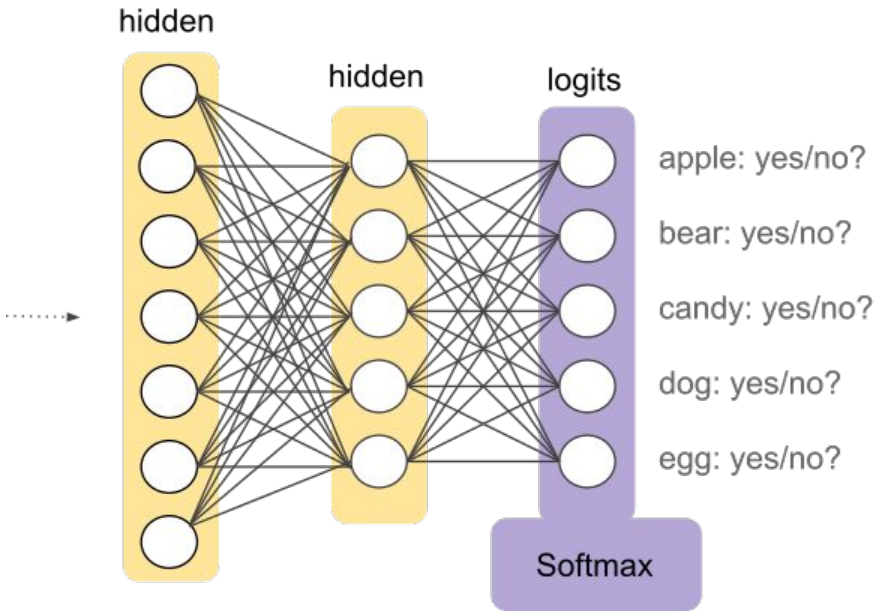


Fully connected layers

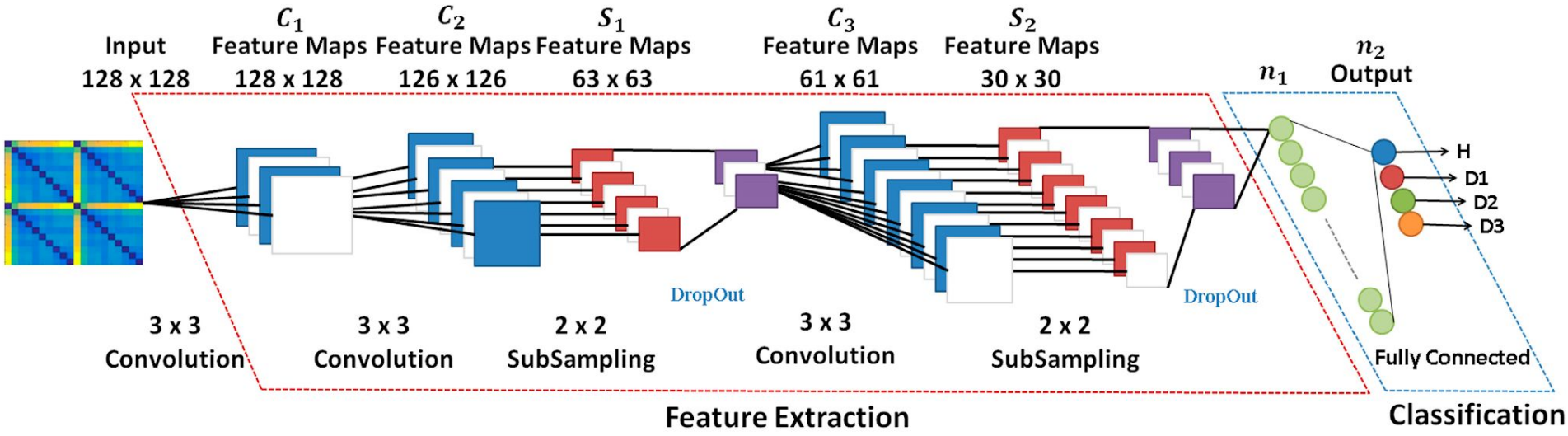
For classification purpose adding a fully connected layer is the best way to learn non-linear combination of the high level features.

Images will be flatten then fed into the fully connected neural network. Over the training iterations with backpropagation applied the network will learn to distinguish between dominating and low level features.

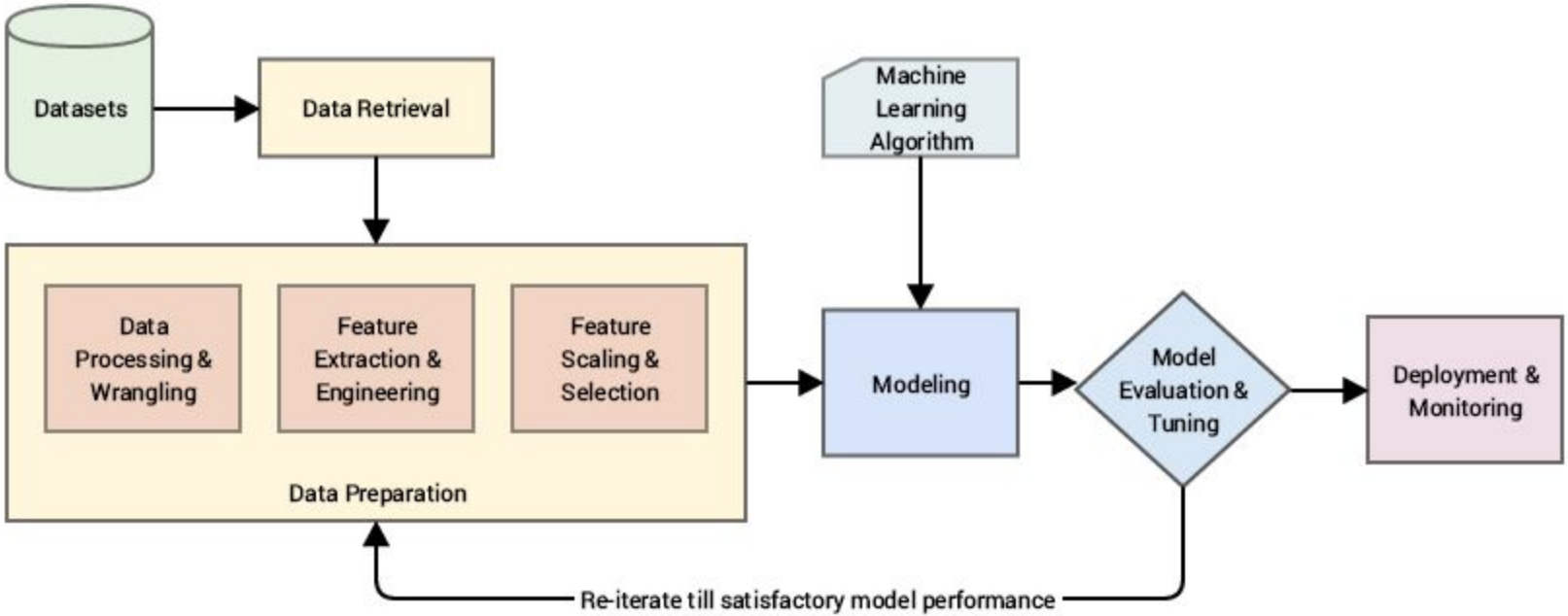
Then use softmax classification technique.



Making Sense of CNN workflow



Deep learning/Machine learning Workflow

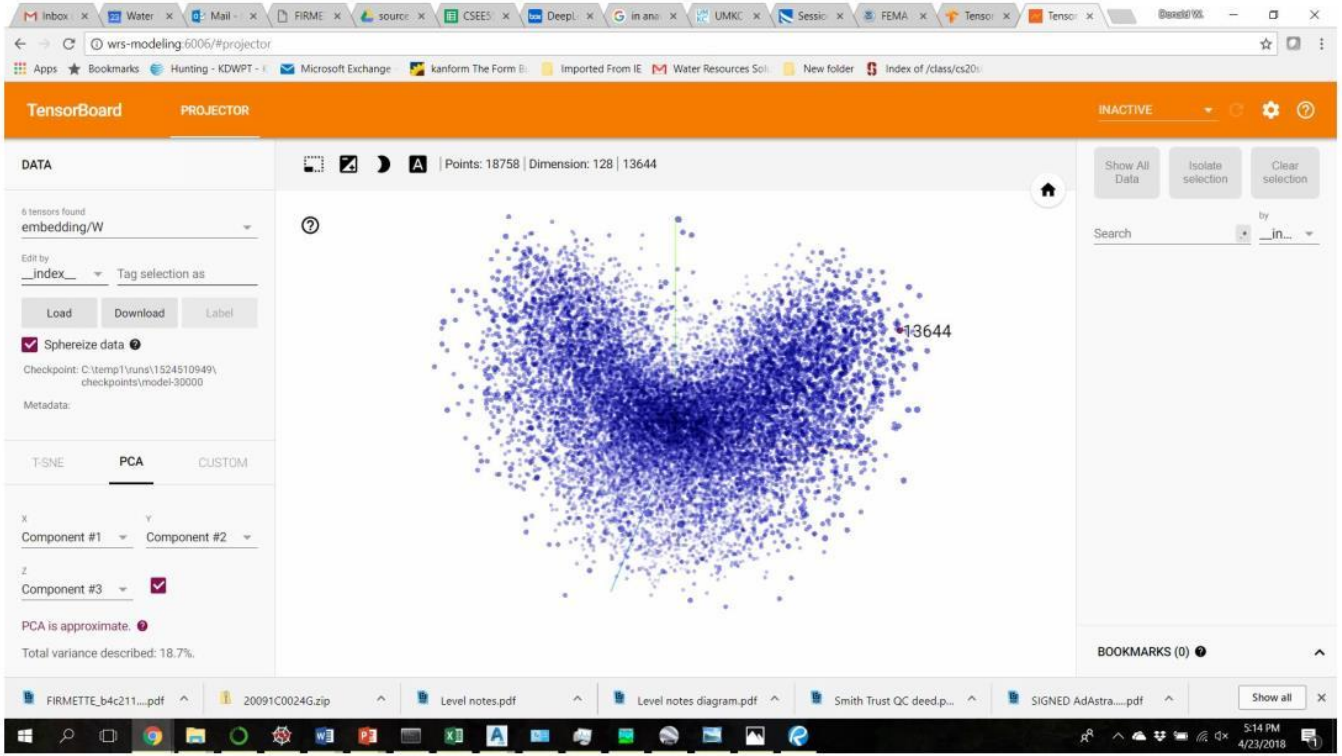


Building CNN network on KERAS

MNIST - classification of images

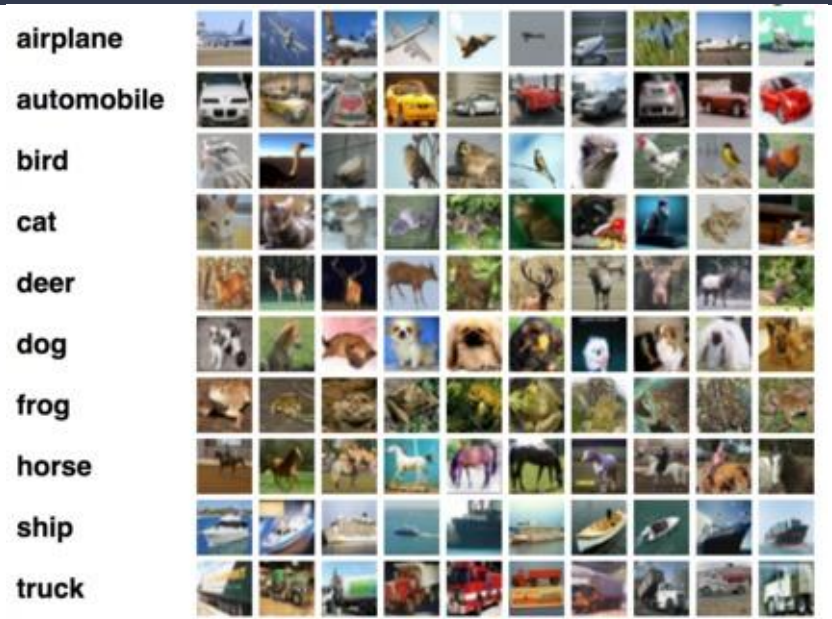
IMDB - sentiment analysis

Using Tensorboard to Visualize networks



CIFAR-10 Data object detection

If there is time...



Complete Tutorial

Best Tips on building CNN

1. Find the best pre-trained model and use transfer learning.
2. Optimize the learning rate with small value.
3. You can play with dropout but it can be disastrous.
4. First layer is the make or break, don’t touch it.
5. Modify the output layer, MNIST needs 10 neurons but you may only need 2.

KERAS Library

Tips for pooling layer

Recurrent Neural Network

Recurrent Neural Networks (RNNs) are popular models that have shown great promise in many NLP tasks. But despite their recent popularity I've only found a limited number of resources that thoroughly explain how RNNs work, and how to implement them.

The idea behind RNNs is to make use of sequential information. In a traditional neural network we assume that all inputs (and outputs) are independent of each other. But for many tasks that's a very bad idea.

If you want to predict the next word in a sentence you better know which words came before it. RNNs are called recurrent because they perform the same task for every element of a sequence, with the output being depended on the previous computations. Another way to think about RNNs is that they have a "memory" which captures information about what has been calculated so far.

RNN for translation

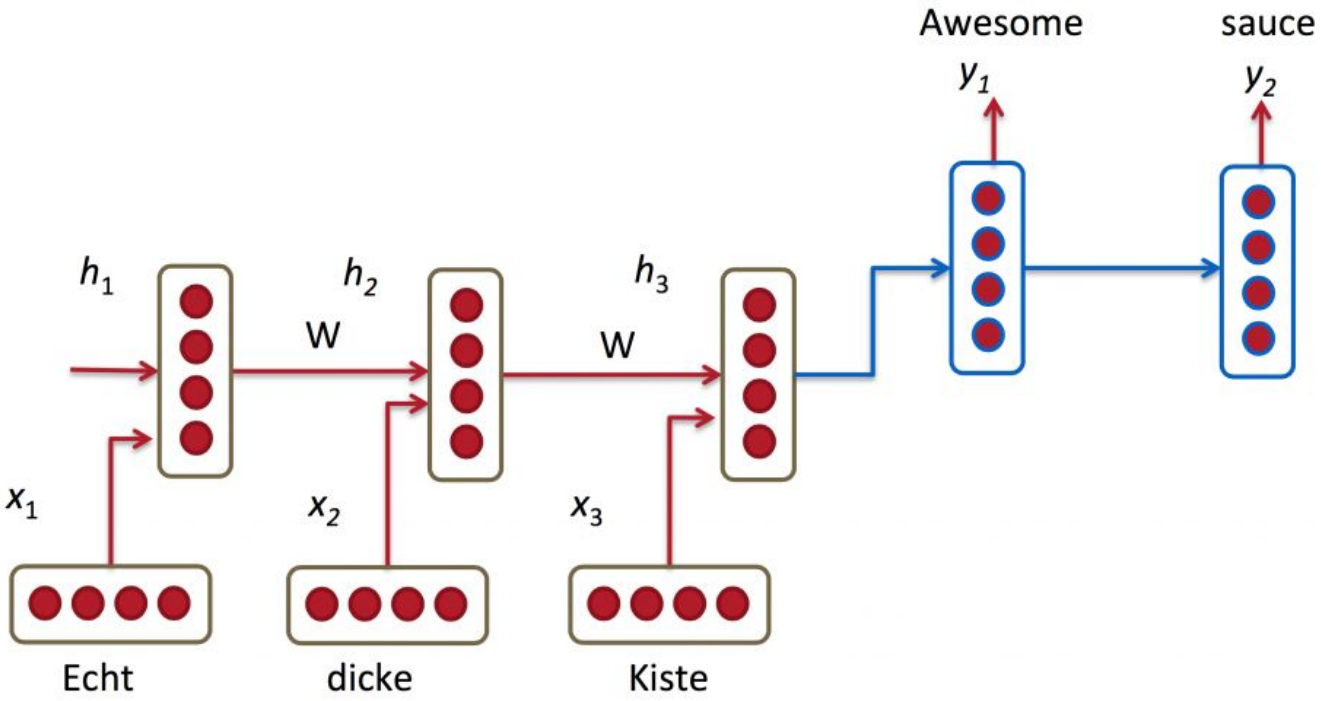
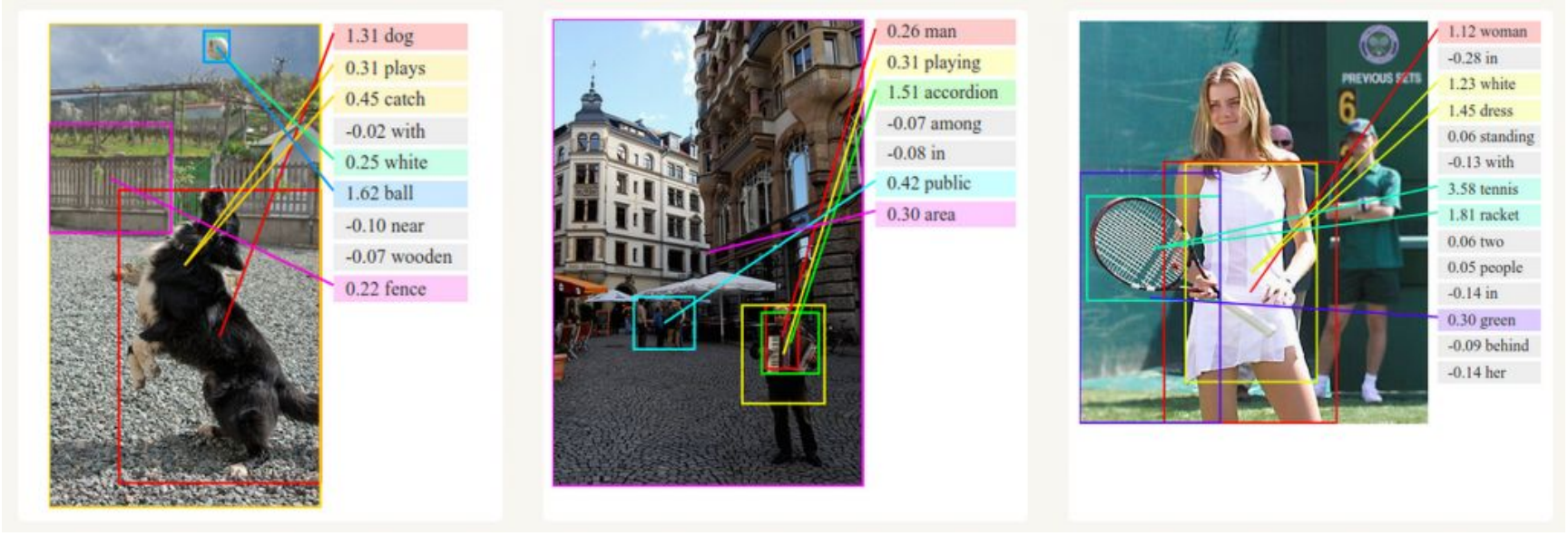


Image Descriptions Generation

Together with convolutional Neural Networks, RNNs have been used as part of a model to generate descriptions for unlabeled images. It's quite amazing how well this seems to work. The combined model even aligns the generated words with features found in the images.



RNN layers: Embedding

A word embedding is a class of approaches for representing words and documents using a dense vector representation.

It is an improvement over more the traditional bag-of-word model encoding schemes where large sparse vectors were used to represent each word or to score each word within a vector to represent an entire vocabulary.

These representations were sparse because the vocabularies were vast and a given word or document would be represented by a large vector comprised mostly of zero values.

Instead, in an embedding, words are represented by dense vectors where a vector represents the projection of the word into a continuous vector space.

Word Embedding

The position of a word within the vector space is learned from text and is based on the words that surround the word when it is used.

The position of a word in the learned vector space is referred to as its embedding.

Two popular examples of methods of learning word embeddings from text include:

- Word2Vec.
- GloVe.

Keras offers an Embedding layer that can be used for neural networks on text data.

It requires that the input data be integer encoded, so that each word is represented by a unique integer. This data preparation step can be performed using the Tokenizer API also provided with Keras.

The Embedding layer is initialized with random weights and will learn an embedding for all of the words in the training dataset.

Learn more from here

Fully Convolutional Neural Networks

There exist 3 hard problems we are trying to solve:

- **Image Classification:** Classify the object (Recognize the object class) within an image.
- **Object Detection:** Classify and detect the object(s) within an image with bounding box(es) bounded the object(s). That means we also need to know the class, position and size of each object.
- **Semantic Segmentation:** Classify the object class for each pixel within an image. That means there is a label for each pixel.

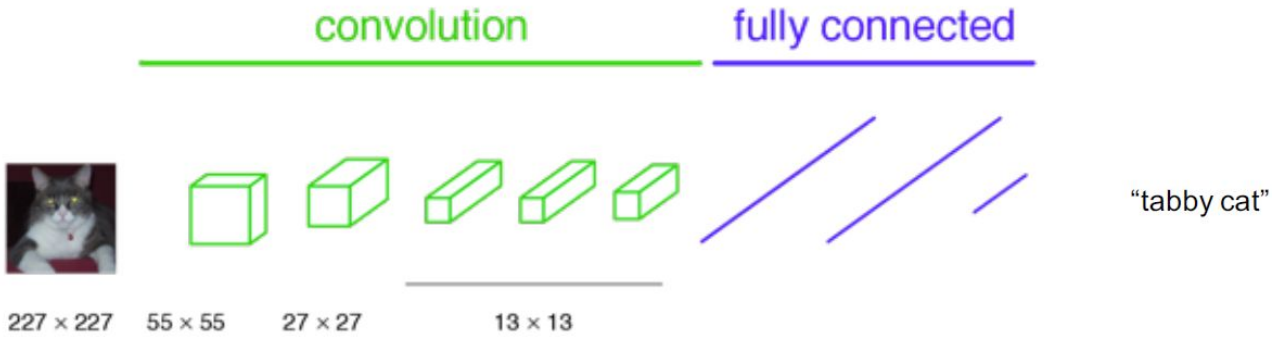
Semantics of an image

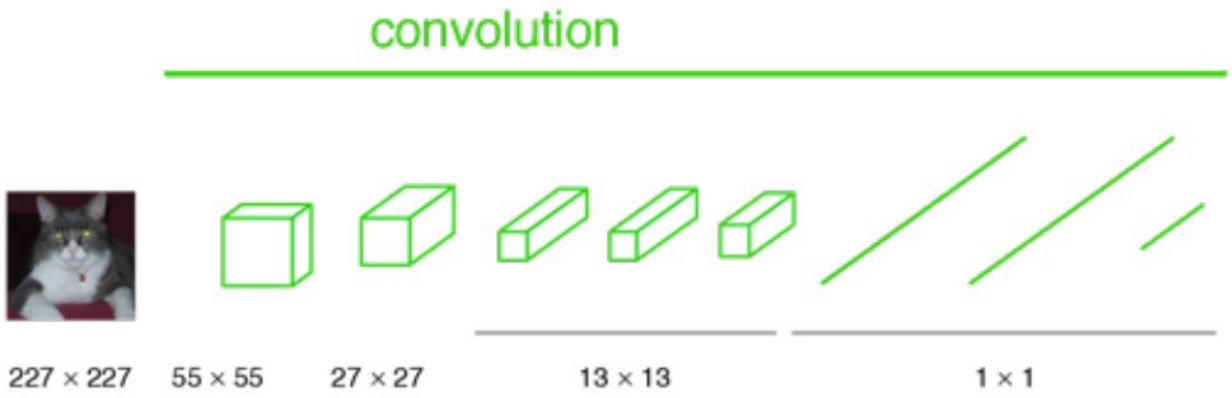
Original Image (Leftmost), Ground Truth Label Map (2nd Left), Predicted Label Map (2nd Right), Overlap Image and Predicted Label (Rightmost)

From Image Classification to Semantic Segmentation

From fully connected layer

Into 1*1 convolution layer.



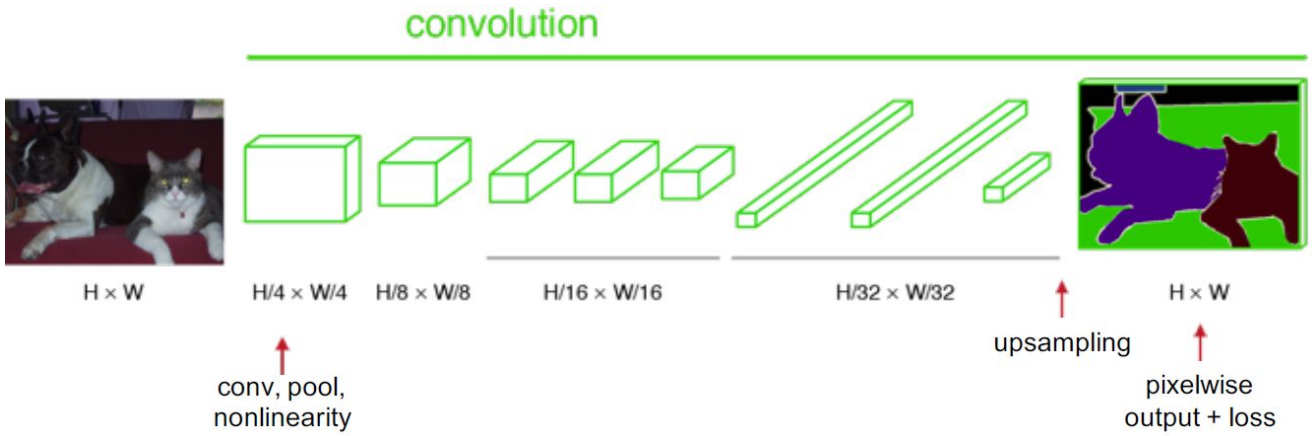


Path of FCN

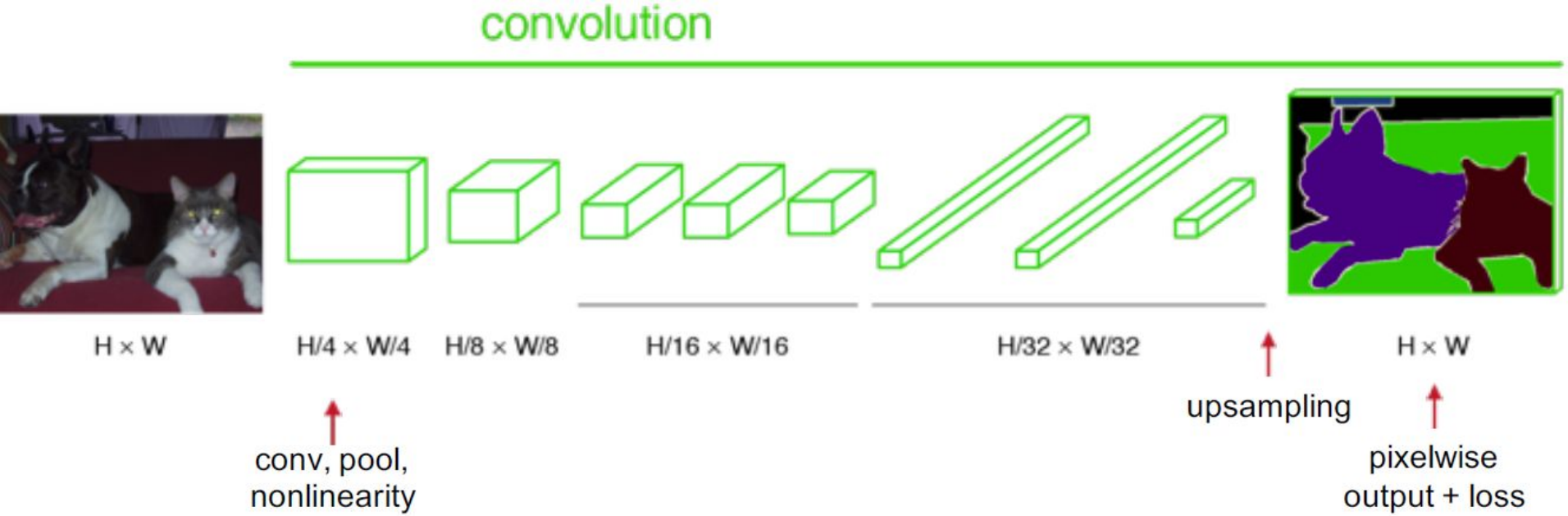
To obtain a segmentation map (output), segmentation networks usually have 2 parts :

Downsampling path : capture semantic/contextual information. Extract and Interpret the context (what).

Upsampling path : recover spatial information. Enable precise localization (where).

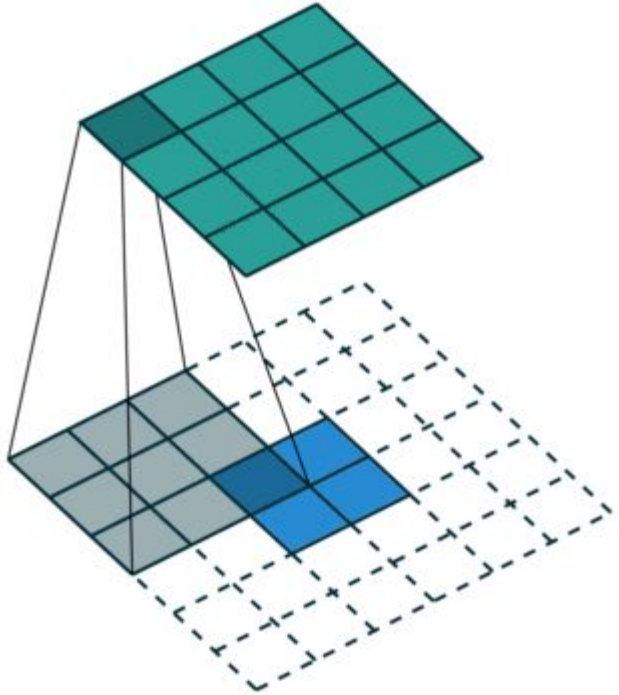


Upsampling


convolution
H × W
H/4 × W/4
H/8 × W/8
H/16 × W/16
H/32 × W/32
H × W
conv, pool,
nonlinearity
upsampling
pixelwise
output + loss

Upsampling via Deconvolution

In convolution we get the output size smaller. This process if the taking the opposite route. Taking the output into a larger size.

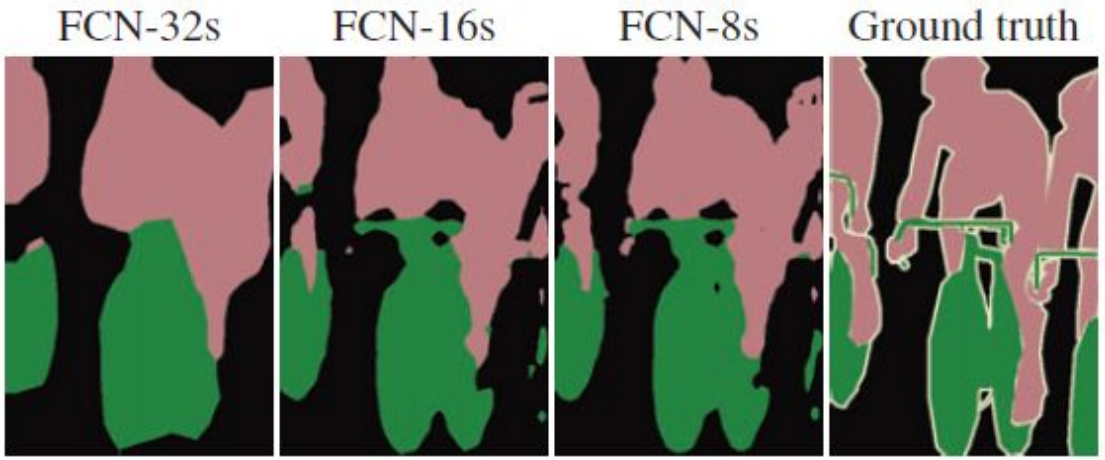


FCN variants

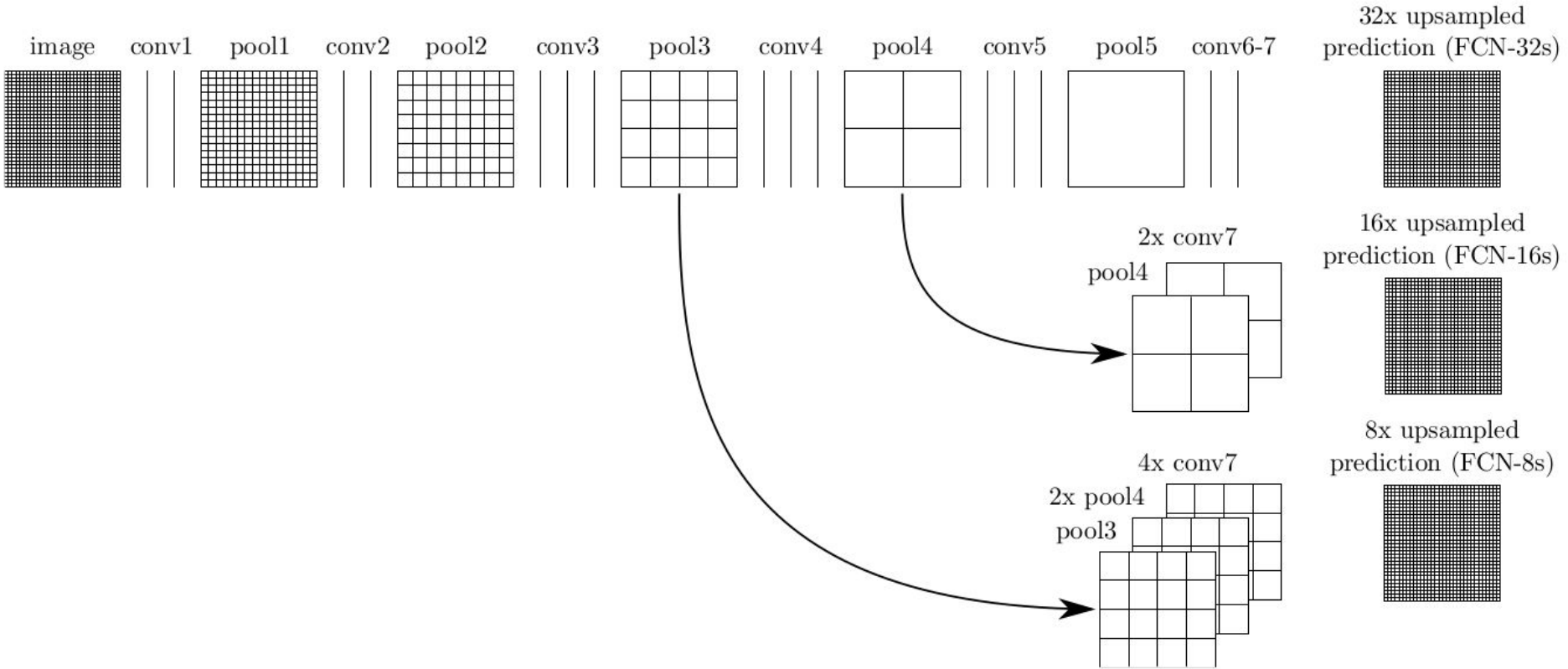
FCN - 32

FCN - 16

FCN - 8



FCN architecture (from FCN paper)



Visualize FCN8 network in tensorboard

How to evaluate FCN?

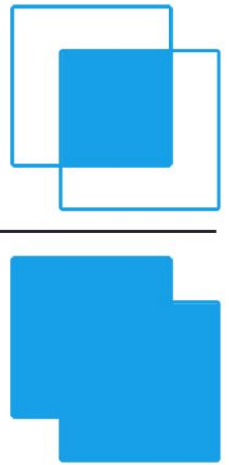
Accuracy per pixel

$$acc(P, GT) = \frac{|\text{pixels correctly predicted}|}{|\text{total nb of pixels}|}$$

Jaccard method

$$jacc(P(class), GT(class)) = \frac{|P(class) \cap GT(class)|}{|P(class) \cup GT(class)|}$$

$$\text{IoU} = \frac{\text{Area of Overlap}}{\text{Area of Union}}$$



Intersection over union

This evaluation metric is often used for image segmentation, since it is more structured. The jaccard is a per class evaluation metric, which computes the number of pixels in the intersection between the predicted and ground truth segmentation maps for a given class, divided by the number of pixels in the union between those two segmentation maps, also for that given class.

IoU method