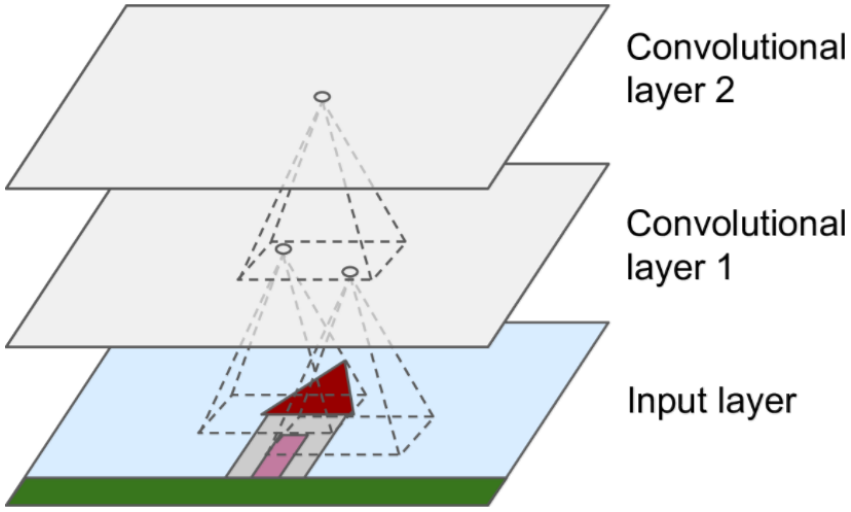
Convolutional Neural Networks – part 2

Let's review:

- In CNN, we use special layers instead of a regular fully connected deep neural network to reduce the number of parameters (connections weights)

- The Convolutional Neural Networks have two new types of layer

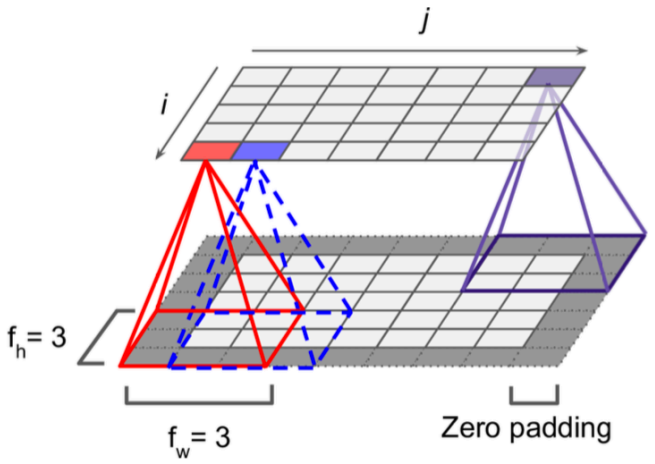
(1) Convolutional layer:



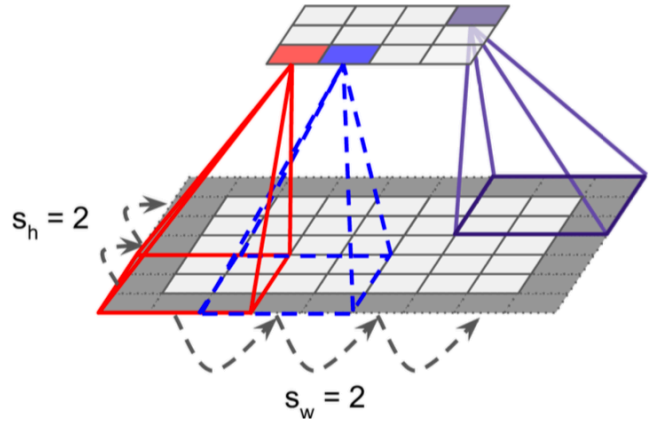
❖ Connections between convolutional layers:

- With zero padding (layer size is preserved)

  The neuron at position (i, j) *is connected to the outputs of neurons in previous layer in*
  rows: i *to* $i + f_h - 1$
  columns: j *to* $j + f_w - 1$



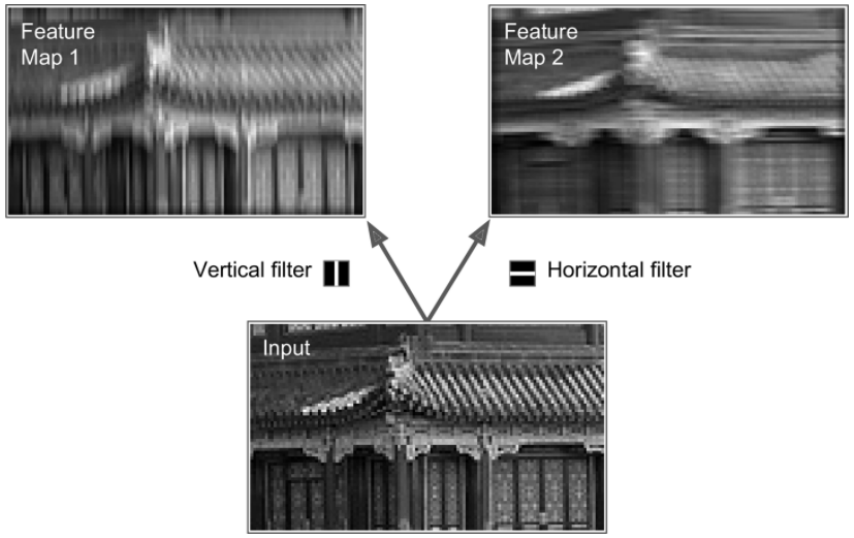
- With stride to reduce dimensionality (stride=2):



- Filters / convolutional kernels:
  - A filter is a given set of neuron weights for example a 7x7 matrix full with 0s except the central column which is full with 1s

Vertical filter 

  - Neurons using these weights will ignore everything except vertical lines
  - A whole layer of neurons has these same weights - so they can detect vertical lines anywhere in the image
  - Applying two different filters to get two feature maps:

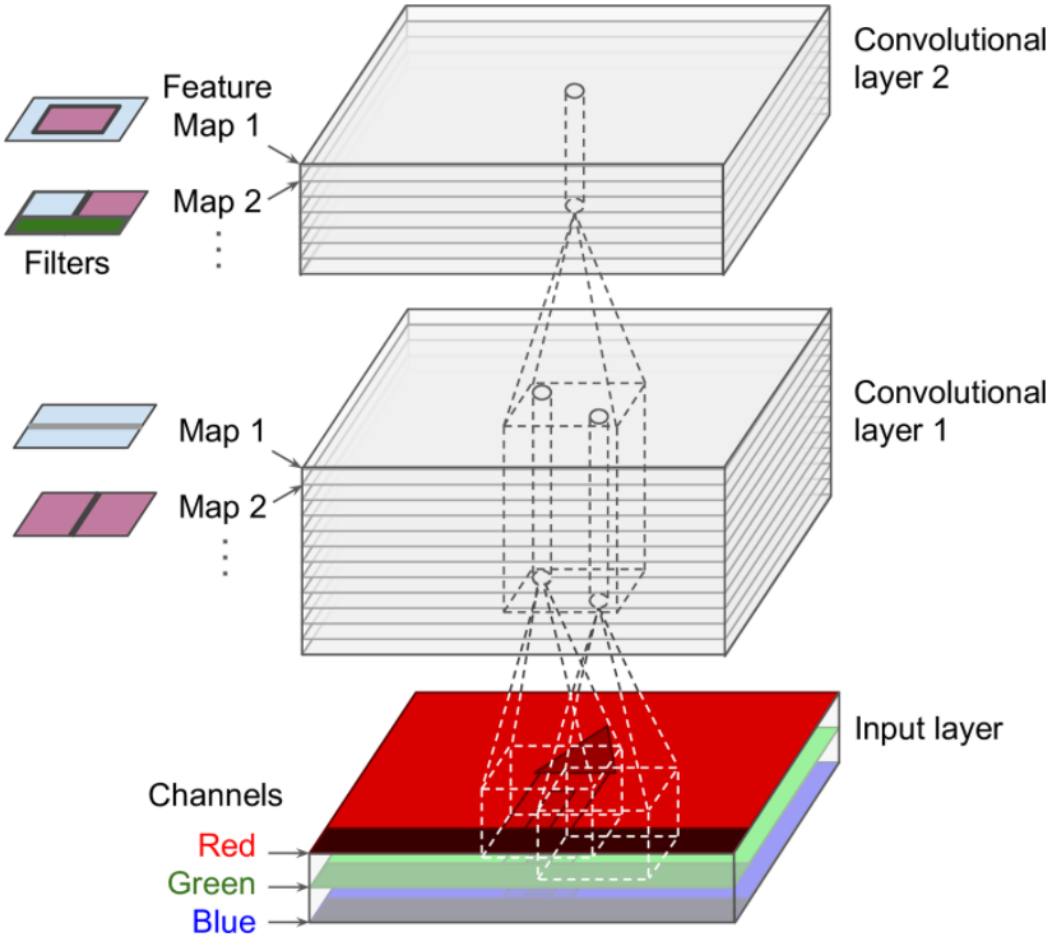


❖ Stacking multiple feature maps:

– Convolutional layers are really 3 dimensional, as each of these layers consists of multiple feature maps

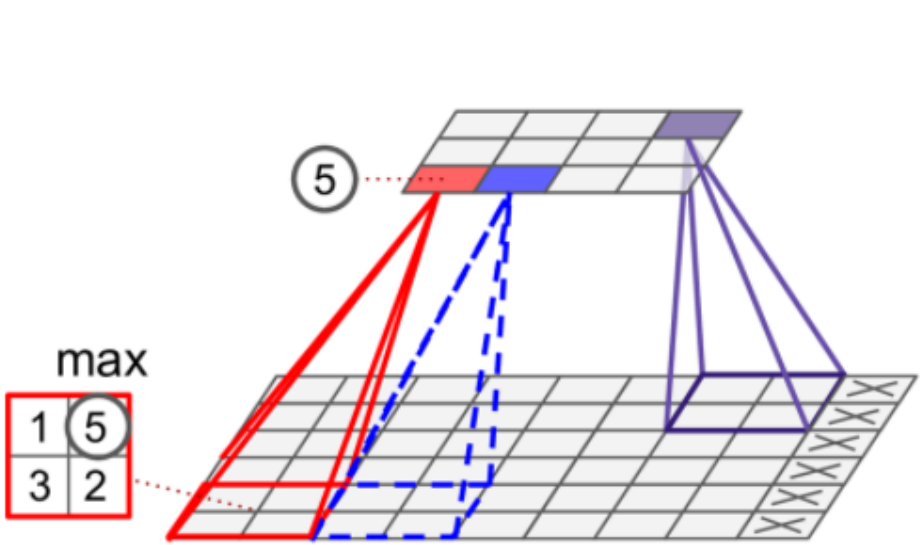
– Within one feature map all neurons share the same parameters (weights and bias)

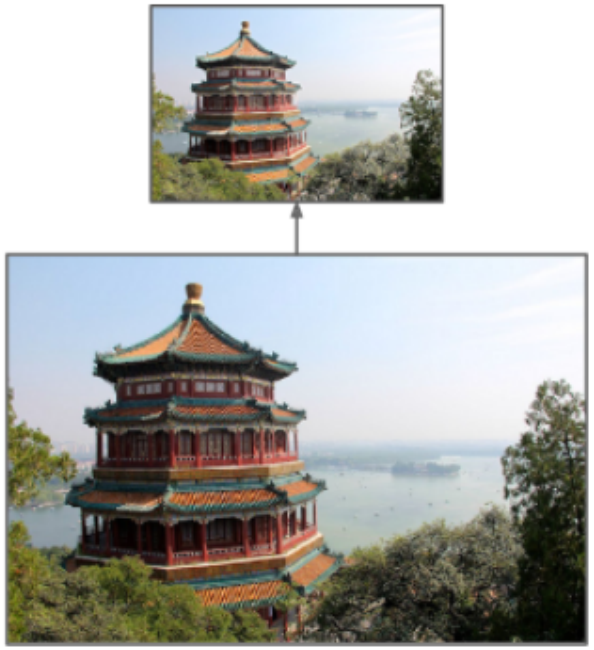
– The neurons' receptive field is the same in all feature maps of the given layer



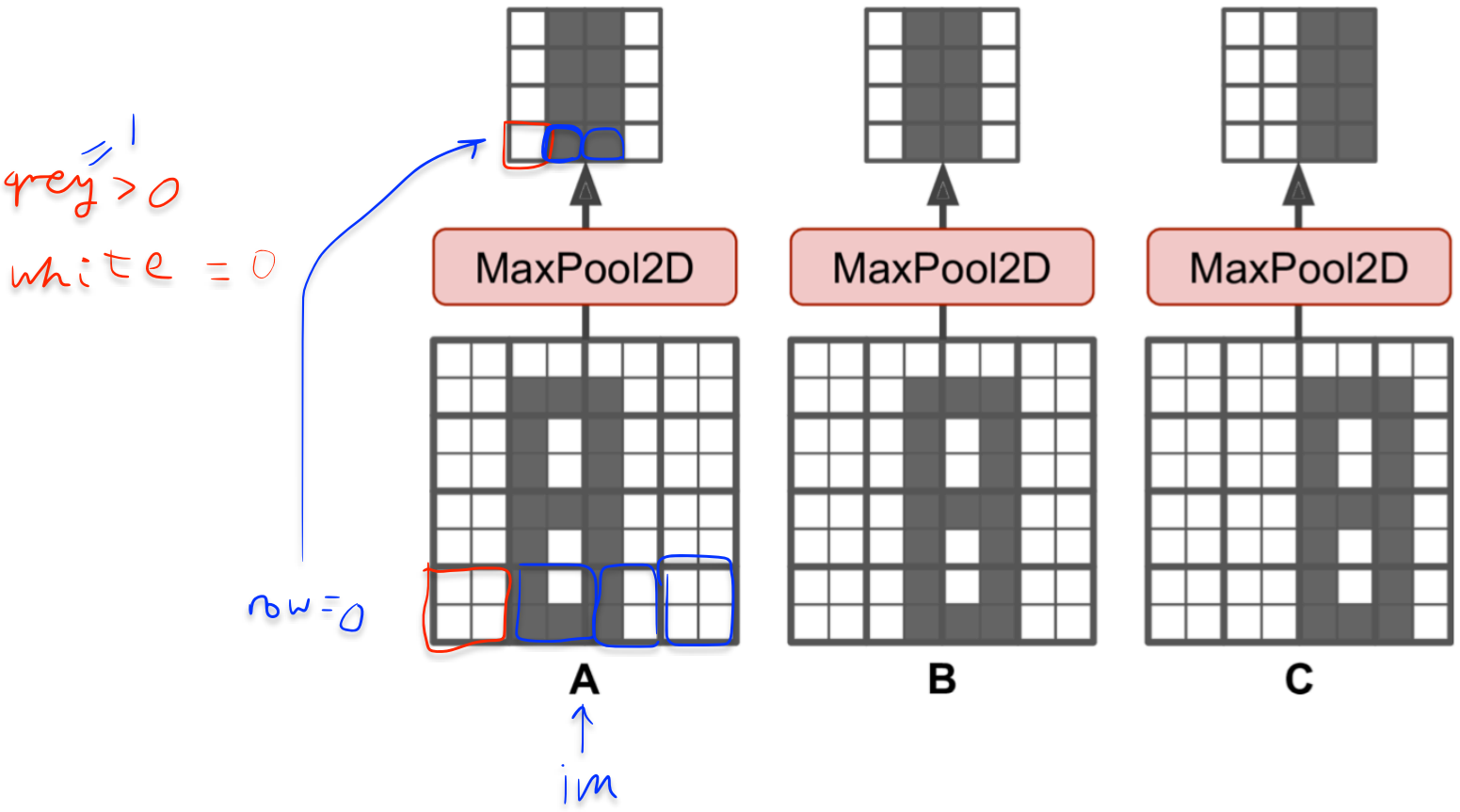
(2) Pooling/sub-sampling layer

- They subsample (shrink) the input image
- It applies some kind of aggregation (max / average) to the connected neurons (no weights!)
- Pooling layers typically operate on each input channel independently, so the depth of the layer doesn't change
- A max pooling layer with a 2x2 kernel and stride of 2 and no padding:

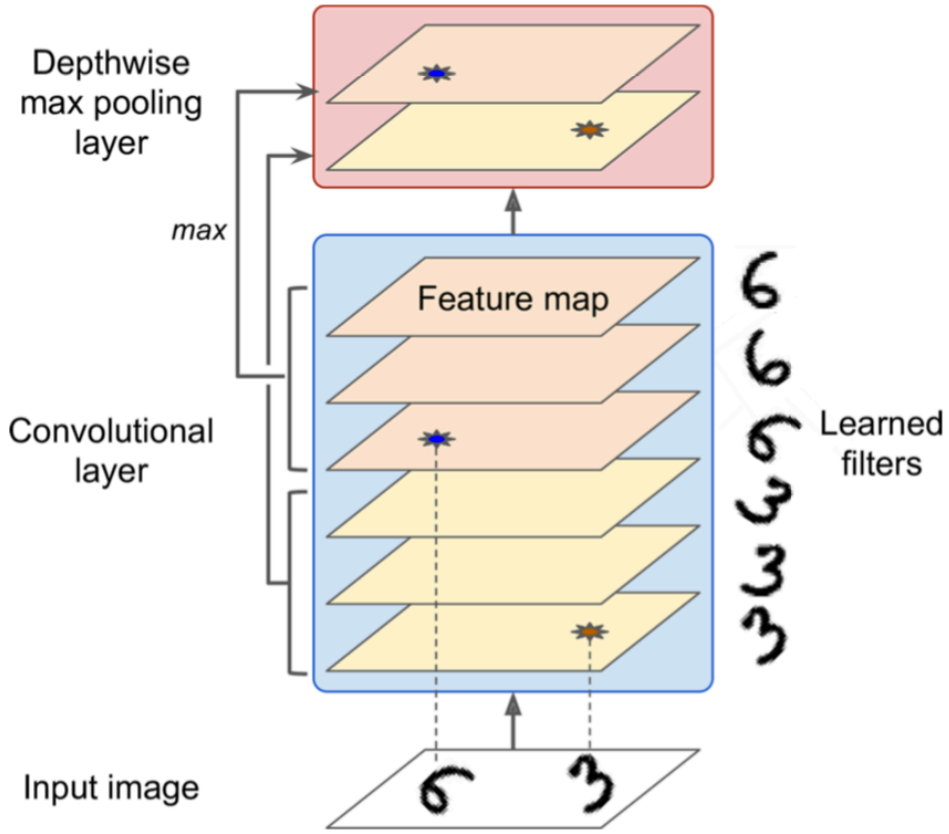




- Another benefit of applying a max pooling is some level of translational invariance.
- Let's apply the same 2x2 kernel and stride of 2 and no padding max pooling to images A, B, and C below. The result from the pooling for A and B are the same!

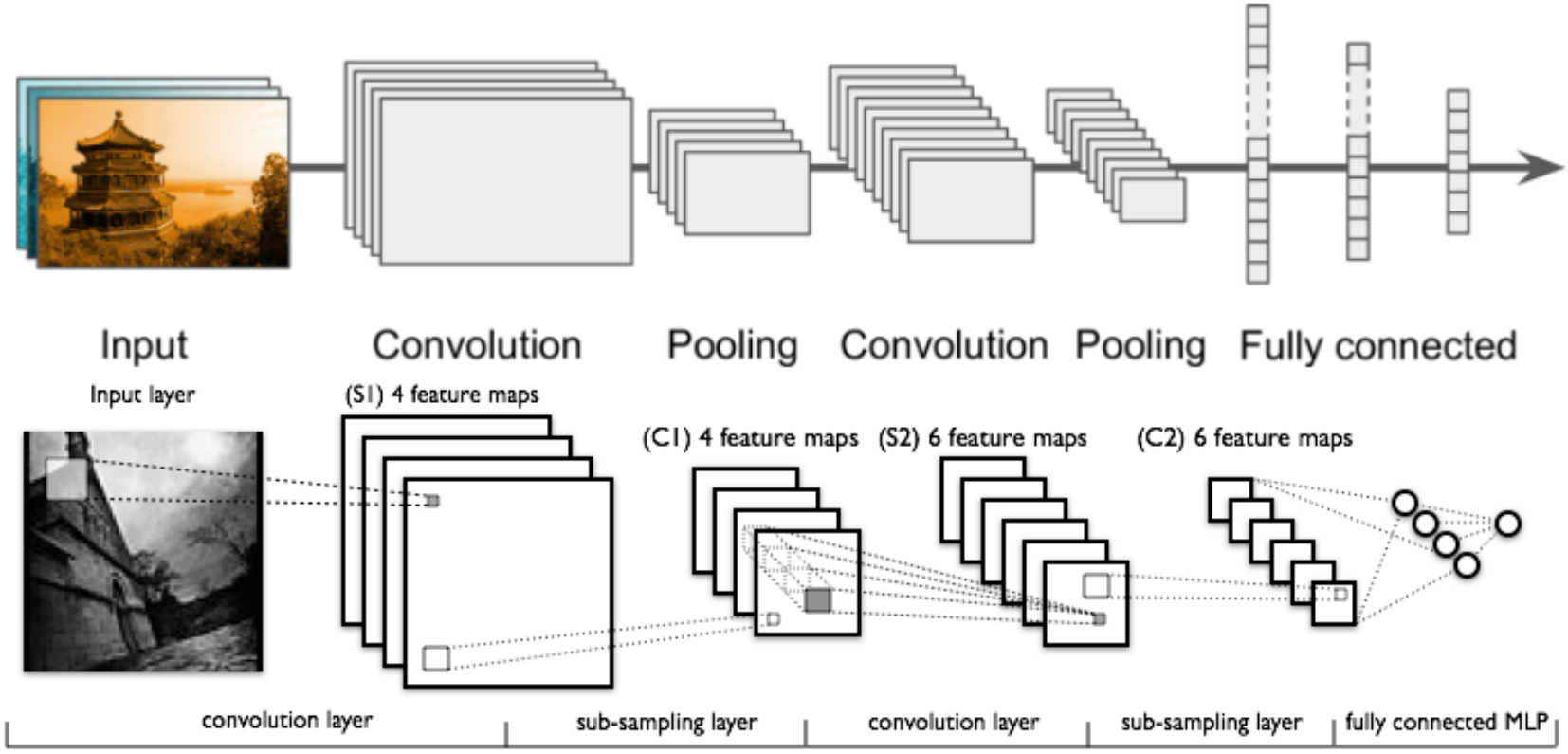


❖ Depth-wise pooling is not common but can allow the CNN to become invariant to various features, e.g. to rotation



- CNN architectures stack multiple convolutional layers (with ReLU) and pooling layers on top of each other

- At the end, you usually have a fully connected network



CNN architectures

- LeNet-5 architecture (1998):
  - Input is 0-padded (from 28x28 to 32x32), no padding afterwards
- AlexNet architecture (2012) - ImageNet challenge 17% top-5 error rate (read)
- GoogLeNet architecture (2014) - ImageNet challenge 7% top-5 error rate Uses 'inception' modules (read this and about Xception net)
- ResNet – 2015 winner in ILSVRC challenge, only 3.6% top-5 error rate
  - The winning variant used an extremely deep CNN composed of 152 layers which confirmed the general trend: models are getting deeper and deeper, with fewer and fewer parameters.
  - Using skip connections (also called shortcut connections)
  - Implement a ResNet-34 CNN using Keras (in 40 lines of code)

Pre-trained Models for Transfer Learning

- In general, you won't have to implement standard models like GoogLeNet or ResNet manually –they are readily available with a single line of code in the keras.applications package:

  ```
  model = keras.applications.resnet50.ResNet50(weights="imagenet")
  ```

- The pretrained models assume that the images are preprocessed in a specific way. So, some pre-processing needs to be done – see code

- The prediction output is a matrix with 1 row per image and 1 column per class (1,000 classes for this datase). The top *3* predictions, look like this

```
Image #0
  n03877845 - palace       42.87%
  n02825657 - bell_cote    40.57%
  n03781244 - monastery    14.56%
```

- If we want to build an image classifier but you do not have enough training data, then we could reuse the lower layers of a pretrained model. For example, we'll train a model to classify pictures of flowers, reusing a pretrained Xception model.

- If you want to perform some data augmentation, you can shift each image, rotate it, rescale it, flip it horizontally or vertically, shear it, or apply any transformation function you want to it.

- It's usually a good idea to freeze the weights of the pretrained layers, at least at the beginning of training.

- This will be very slow, without GPUs. We can run the code in Colab

- After training the model for a few epochs, its validation accuracy should stabilize around 75–80% => the top layers are now pretty well trained, so unfreeze all the layers (or just the top ones) and continue training with much lower learning rate. This model can achieve ~95% accuracy

Classification and Localization

- We are able to classify images quite well. But if we decide there is a flower in a given image, where is the flower is in the picture?

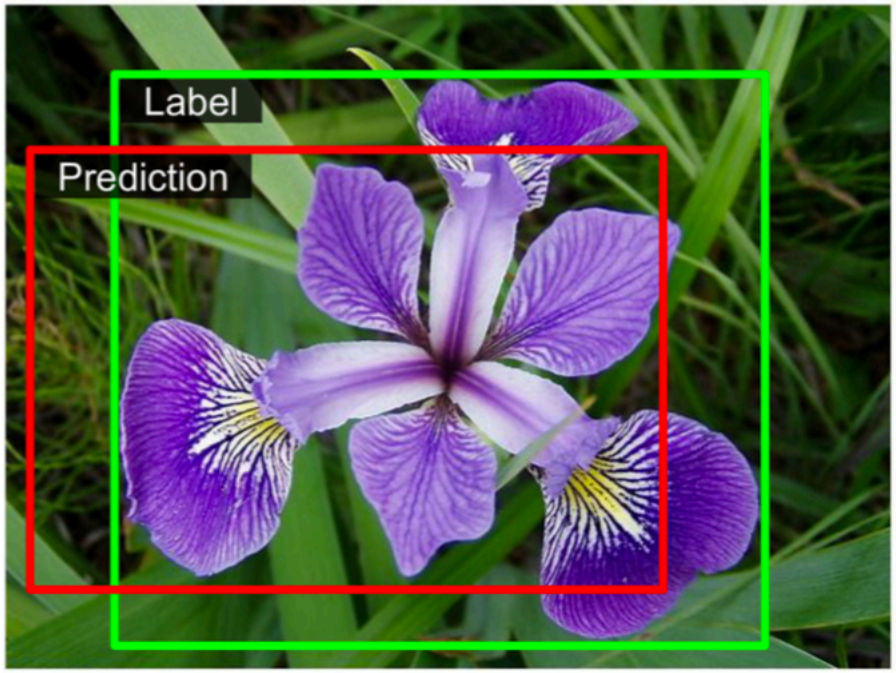
- We can approach it as a regression task - predict a bounding box around the object by predicting the horizontal and vertical coordinates of the object's center, as well as its height and width. This means we have four numbers to predict.

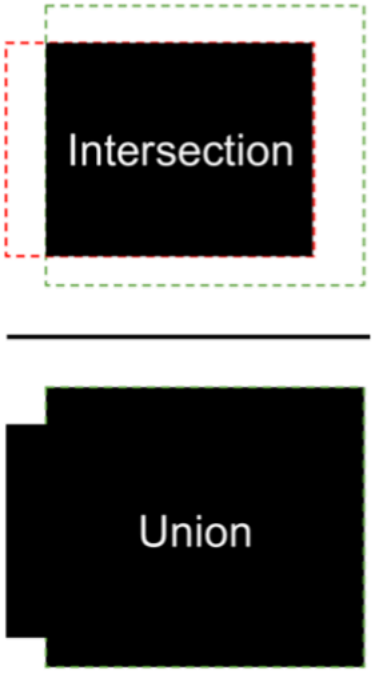
- We just add a second dense output layer with four units (typically on top of the global average pooling layer), and it can be trained using the MSE loss.

- But the flowers dataset does not have bounding boxes around the flowers! So, we need to add them ourselves!

- Each item should be a tuple of the form (images, (class_labels, bounding_boxes))

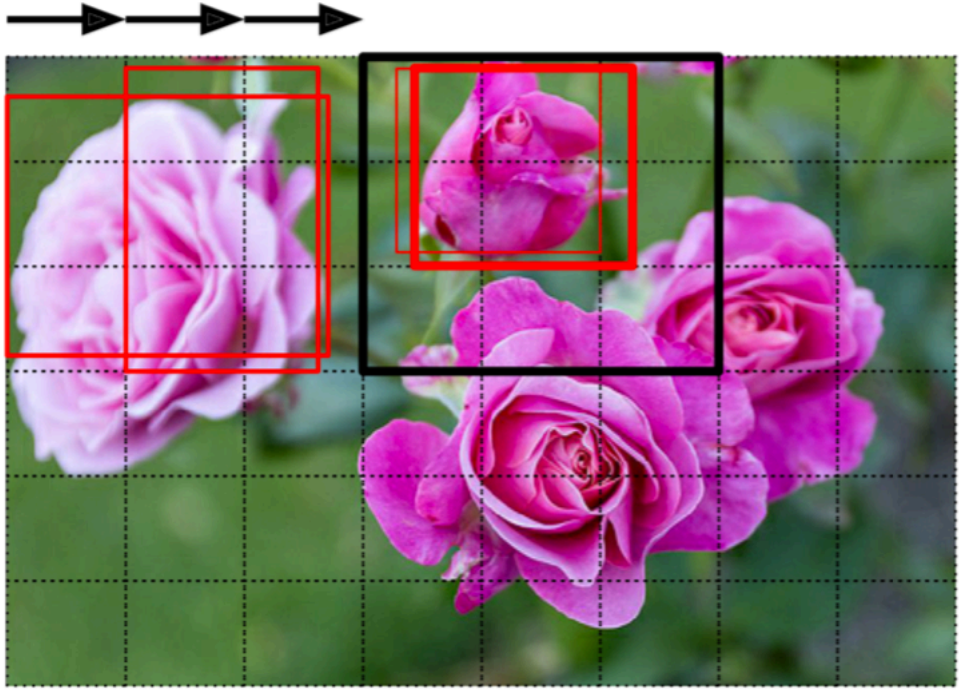
- The most common metric for predicting bounding boxes is not MSE but the *Intersection over Union* (IoU): the area of overlap between the predicted bounding box and the target bounding box





Object Detection

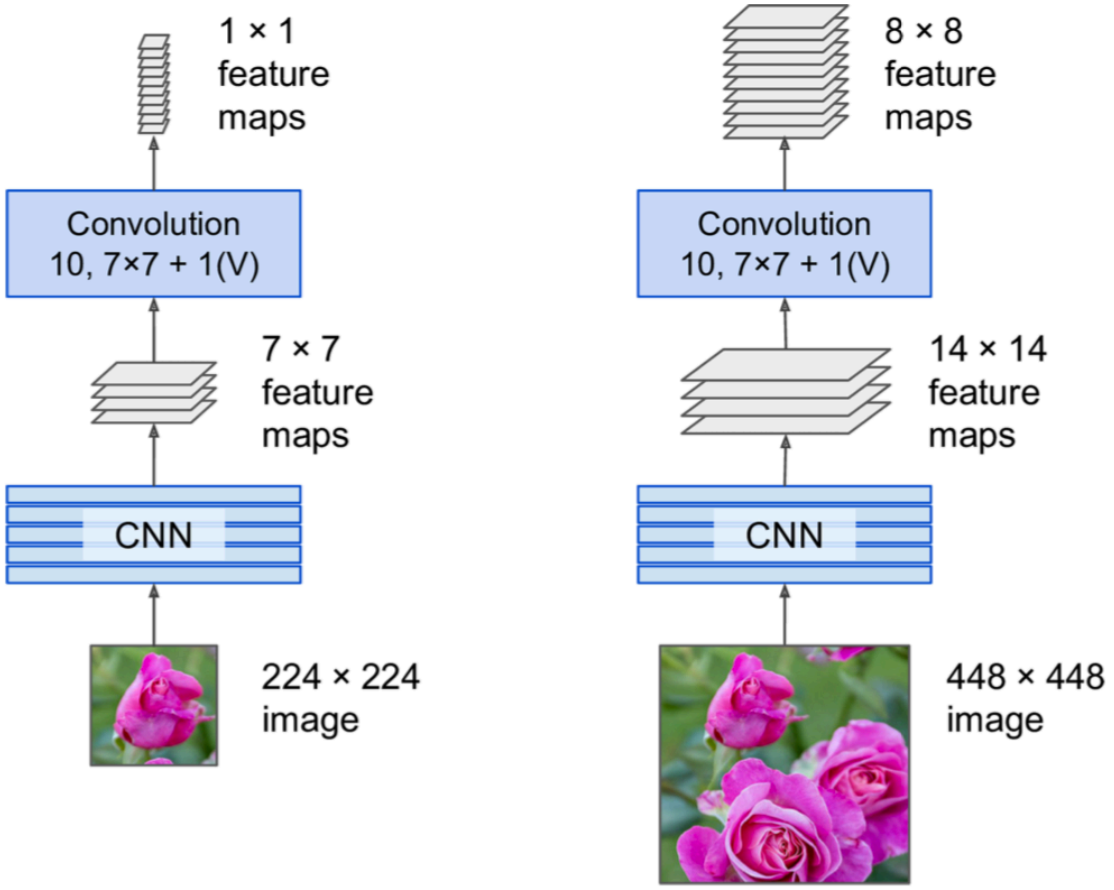
- What if there is more than 1 object in the bounding box?
- Till recently the approach was to slide a CNN over the image:



- The black box above is a CNN sliding over 3x3 regions. The red boxes correspond to the same object detected multiple times

- We need to get rid of all the unnecessary bounding boxes. Can use non-max suppression
    - Add an extra objectness output to your CNN, to estimate the probability that a flower is indeed present in the image. It must use the sigmoid activation function, and you can train it using binary cross-entropy loss. Then get rid of all the bounding boxes for which the objectness score is below some threshold: this will drop all the bounding boxes that don't actually contain a flower.
    - Find the bounding box with the highest objectness score, and get rid of all the other bounding boxes that overlap a lot with it (e.g., with an IoU greater than 60%).
- This approach works pretty well, but is "expensive" (fits CNN many times) There is a much faster way to slide a CNN across an image.

Fully Convolutional Network (FCN):



OR use You Only Look Once (YOLO) – very fast!

- A very common metric used in object detection tasks is the mean Average Precision (mAP).

- Compute the maximum precision you can get with at least 0% recall, then 10% recall, 20%, and so on up to 100%, and then calculate the mean of these maximum precisions => Average Precision (AP)

- When there are >= 2 classes, we can compute the AP for each class, and then compute the mean AP (mAP).

Semantic Segmentation

- Each pixel is classified according to the class of the object it belongs to (e.g., road, car, pedestrian, building, etc. Different objects of the same class are not distinguished:

- Read