Deep Learning - Foundations and Concepts

Chapter 10. Convolutional Networks

nonlineark@github

March 13, 2025

Outline

Computer vision

- Computer vision was one of the first fields to be transformed by the deep learning revolution, predominantly thanks to the CNN architecture.
- Recently alternative architectures based on transformers have become competitive with convolutional networks in some applications.
- Some applications for machine learning in computer vision: Classification, detection, segmentation, caption generation, synthesis, inpainting, style transfer, super-resolution, depth prediction, scene reconstruction.

Image data

- The structure of an image:
  - An image comprises a rectangular array of pixels.
  - Each pixel has either a grey-scale intensity or a triplet of red, green and blue channels each with its own intensity value.
- Challenges of applying neural networks to image data:
  - Images generally have a high dimensionality.
  - Image data is highly structured.
- Local correlations can be used to encode strong inductive biases into a neural network, leading to models with far fewer parameters and with much better generalization accuracy.

Inductive biases

To exploit the two-dimensional structure of image data to create inductive biases, we can use four interrelated concepts:

- Hierarchy.
- Locality.
- Equivariance.
- Invariance.

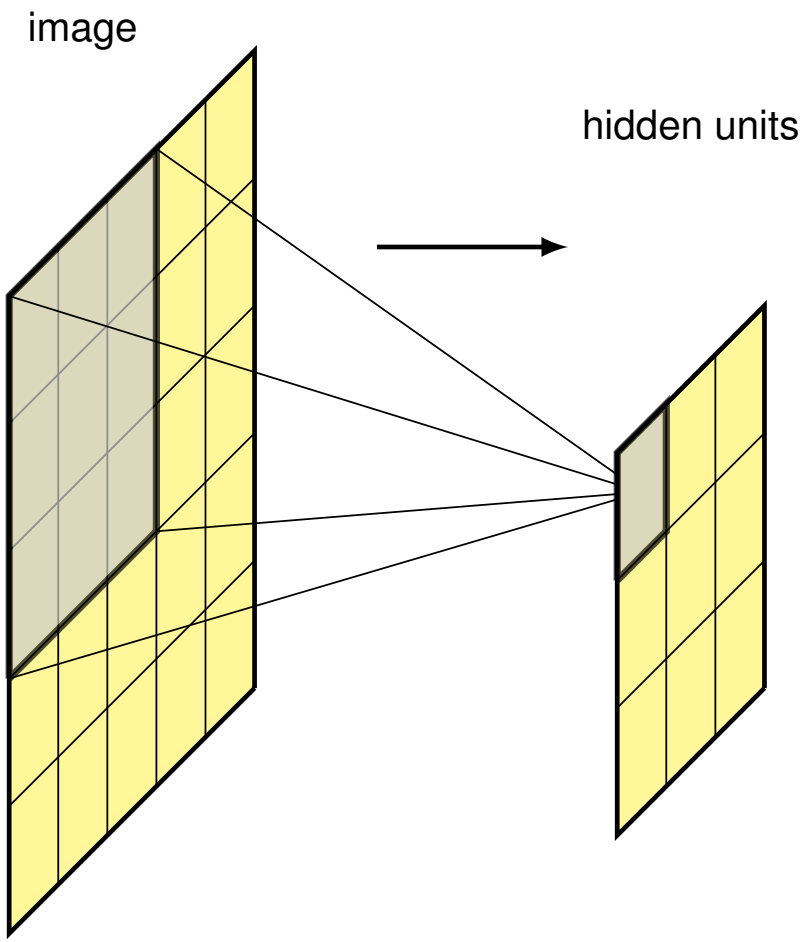
Feature detectors

Let's restrict our attention to grey-scale images first. Consider a single unit in the first layer of a neural network:

- It takes as input just the pixel values from a small rectangular region, or patch, from the image. This patch is referred to as the receptive field of that unit.
- The output of this unit is given by the usual functional form comprising a weighted linear combinations of the input values, which is subsequently transformed using a nonlinear activation function: $z = \mathrm{ReLU}(w^T x + w_0)$.
- The weights themselves form a small two-dimensional grid known as a filter, sometimes also called a kernel.
- The unit acts as a feature detector that signals when it finds a sufficiently good match to its kernel (basic properties of inner product).

Feature detectors

Figure: A single unit in the first layer of a neural network



Translation equivariance

- To generalize what our neural network has learned in one location to all possible locations in the image, we can simply replicate the same hidden-unit weight values at multiple locations across the image.
- The units of the hidden layer form a feature map in which all the units share the same weights.

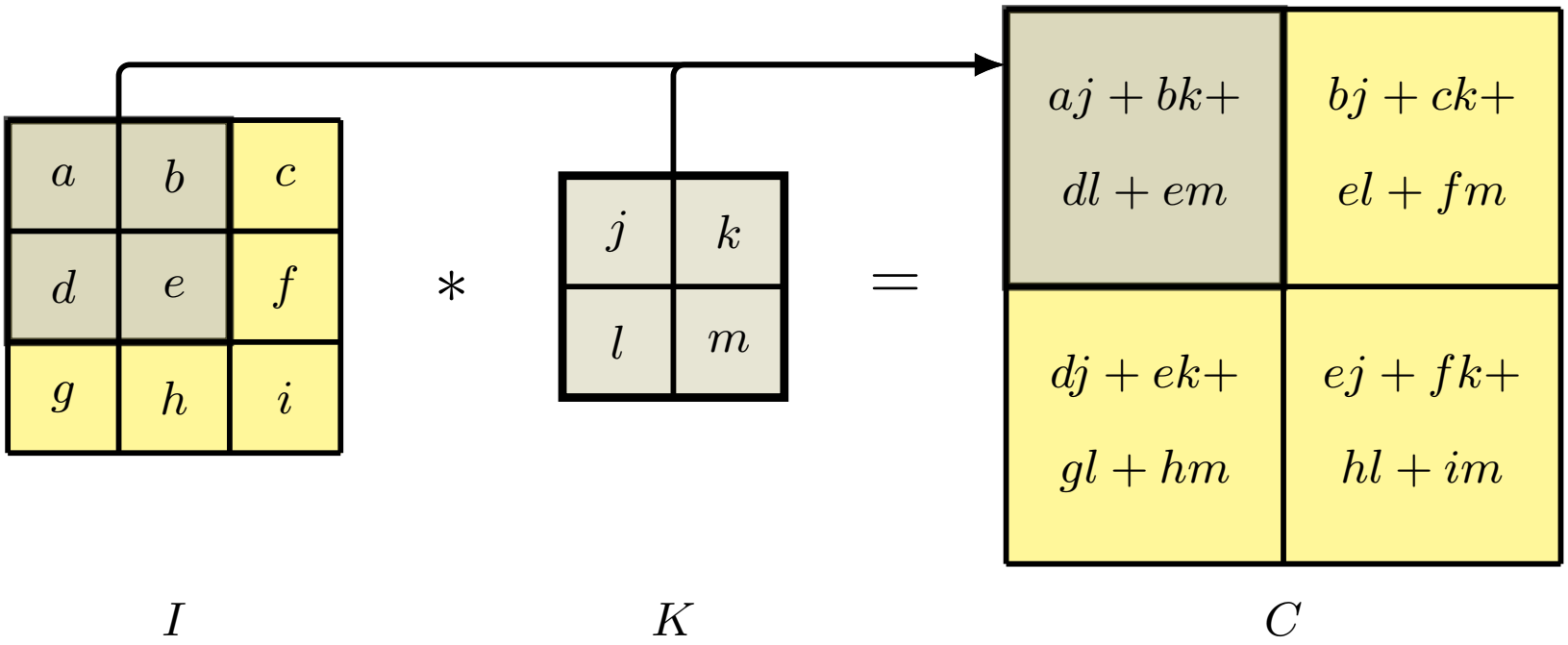
For an image I with pixel intensities $I(j,k)$, and a filter K with pixel values $K(l,m)$, the feature map C has activation values given by:

$$C(j,k) = \sum_l \sum_m I(j+l, k+m) K(l,m)$$

where we have omitted the nonlinear activation function for clarity. This is an example of a convolution and is sometimes expressed as $C = I * K$.

Translation equivariance

Figure: A 3×3 image convolved with a 2×2 filter



Translation equivariance

Comparing this convolutional structure with a standard fully connected network, we see several advantages:

- The connections are sparse, leading to far fewer weights even with large images.
- The weight values are shared, greatly reducing the number of independent parameters and consequently reducing the required size of the training set.
- The same network can be applied to images of different sizes without the need for retraining.
- Convolutions can be implemented very efficiently by exploiting the massive parallelism of GPUs.

Padding

Consider an image of dimensionality $J \times K$ and a kernal of dimensionality $M \times M$:

- The feature map is of dimensionality $(J - M + 1) \times (K - M + 1)$, which is smaller than the original image.
- We can pad the original image with additional pixels around the outside. If we pad with P pixels then the output map has dimensionality $(J + 2P - M + 1) \times (K + 2P - M + 1)$:
  - If $P = 0$, this is called a valid convolution.
  - If $P = \frac{M-1}{2}$, this is called a same convolution.
- Padding values are typically set to zero, after first subtracting the mean from each image so that zero represents the average value of the pixel intensity.

Strided convolutions

Sometimes we wish to use feature maps that are significantly smaller than the original image to provide flexibility in the design of convolutional network architectures. One way to achieve this is to use strided convolutions in which, the filter is moved in larger steps of size S, called the stride. The feature map will have dimensionality:

$$(\lfloor \frac{J + 2P - M}{S} \rfloor + 1) \times (\lfloor \frac{K + 2P - M}{S} \rfloor + 1)$$

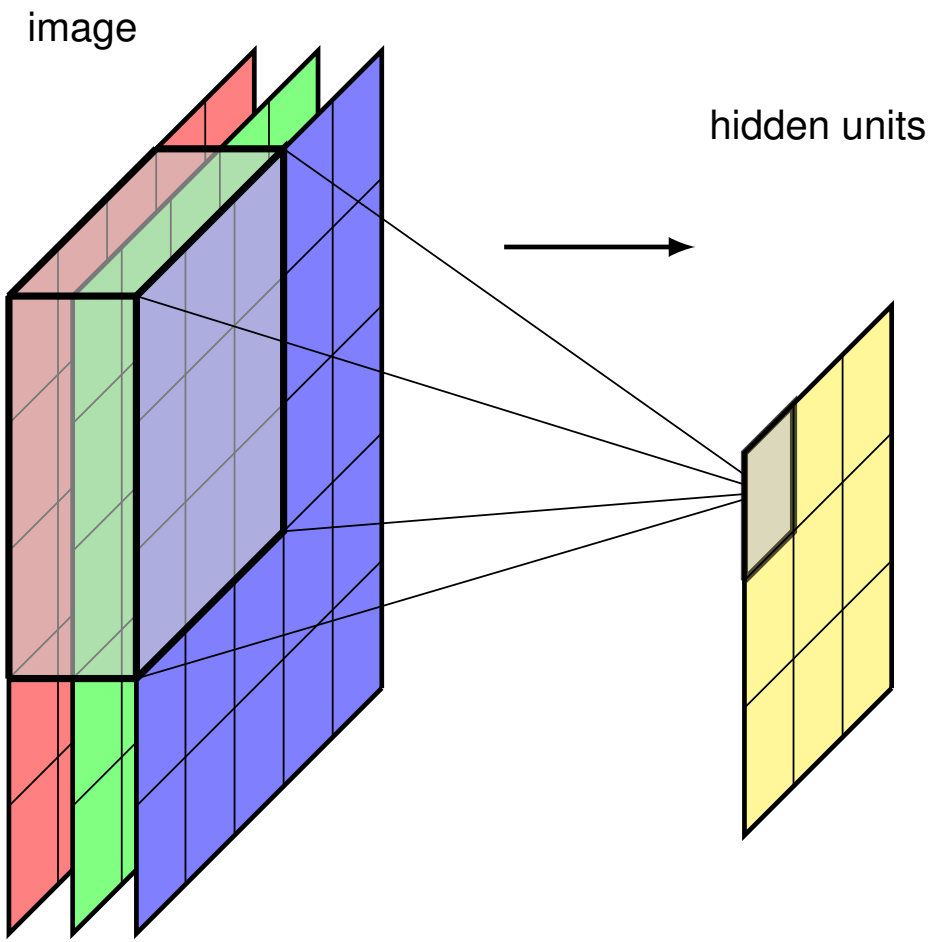
Multi-dimensional convolutions

We can easily extend convolutions to cover multiple channels by extending the dimensionality of the filter:

- An image with $J \times K$ pixels and C channels will be described by a tensor of dimensionality $J \times K \times C$.
- We can introduce a filter described by a tensor of dimensionality $M \times M \times C$.

Multi-dimensional convolutions

Figure: A multi-dimensional filter



Multi-dimensional convolutions

To build more flexible models, we simply include multiple filters, in which each filter has its own independent set of parameters giving rise to its own independent feature map:

- We again refer to these separate feature maps as channels.
- The filter tensor now has dimensionality $M \times M \times C \times C_{\mathrm{OUT}}$ where C is the number of input channels and C_{OUT} is the number of output channels.
- Each output channel will have its own associated bias parameter, so the total number of parameters will be $(M^2C + 1)C_{\mathrm{OUT}}$.

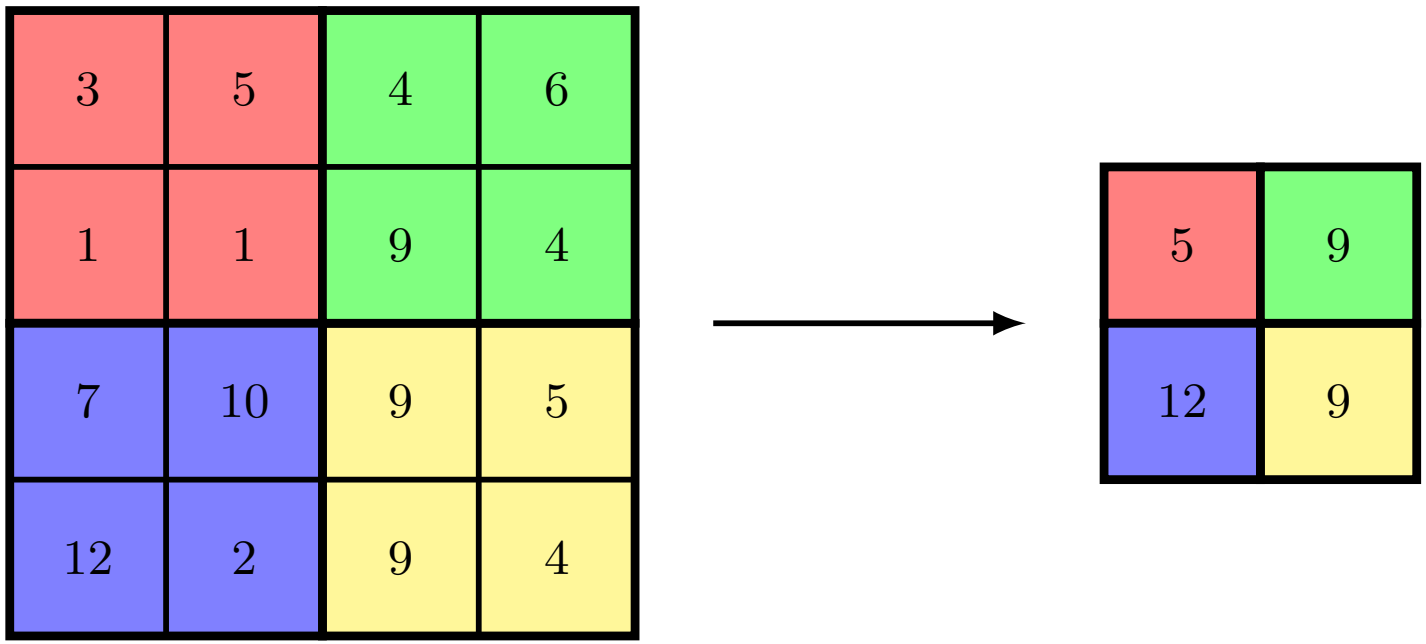
Pooling

Small changes in the relative locations do not affect the classification, and we want to be invariant to such small translations of individual features. This can be achieved using pooling:

- A pooling layer is similar to a convolutional layer:
  - Each unit takes input from a receptive field in the previous feature map layer.
  - There is a choice of filter size and of stride length.
- The difference is that the output of a pooling unit is a simple, fixed function of its inputs:
  - Max-pooling.
  - Average pooling.

Pooling

Figure: Illustration of max-pooling

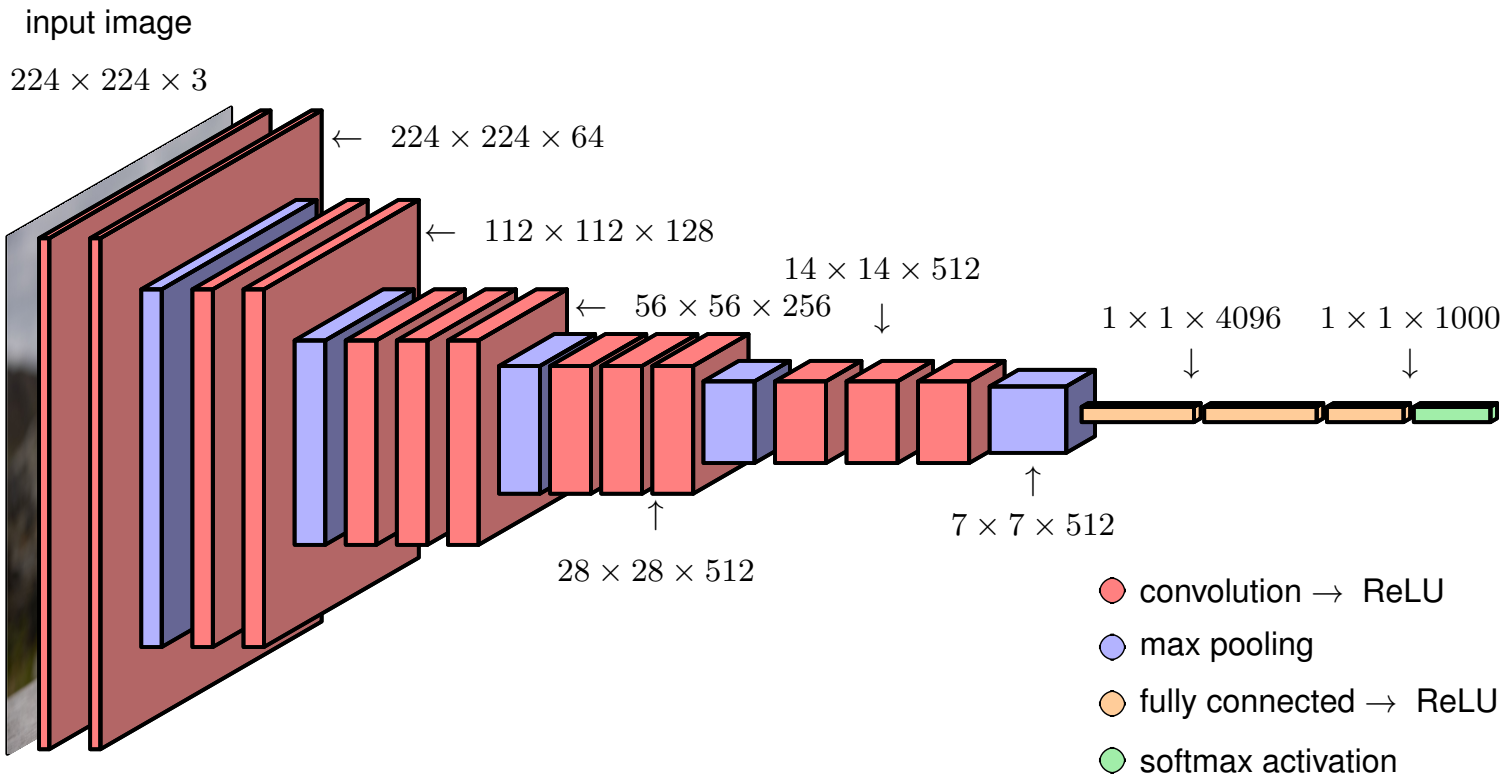


Multilayer convolutions

- To allow the network to discover and represent hierarchical structure in the data, we extend the architecture by considering multiple layers.
- Each convolutional layer is described by a filter tensor of dimensionality $M \times M \times C_{\mathrm{IN}} \times C_{\mathrm{OUT}}$ in which the number of independent weight and bias parameters is $(M^2 C_{\mathrm{IN}} + 1) C_{\mathrm{OUT}}$.
- Each such convolutional layer can optionally be followed by a pooling layer.
- In many applications (e.g., classification tasks), the output units need to combine information from across the whole of the input image. This is typically achieved by introducing one or two standard fully connected layers as the final stages of the network.

Example network architectures

Figure: The architecture of a convolutional network called VGG-16



Visual cortex

Historically, much of the motivation for CNNs came from pioneering research in neuroscience:

- Simple cells: Respond to visual inputs with a simple edge oriented at a particular angle and located at a particular position within the visual field.
- Complex cells: Respond to more complex stimuli and which seem to be derived by combining and processing the output of simple cells.
- Grandmother cells: Respond if, and only if, the visual input corresponds to a person's grandmother, irrespective of location, scale, lighting, or other transformations of the scene.

Visual cortex

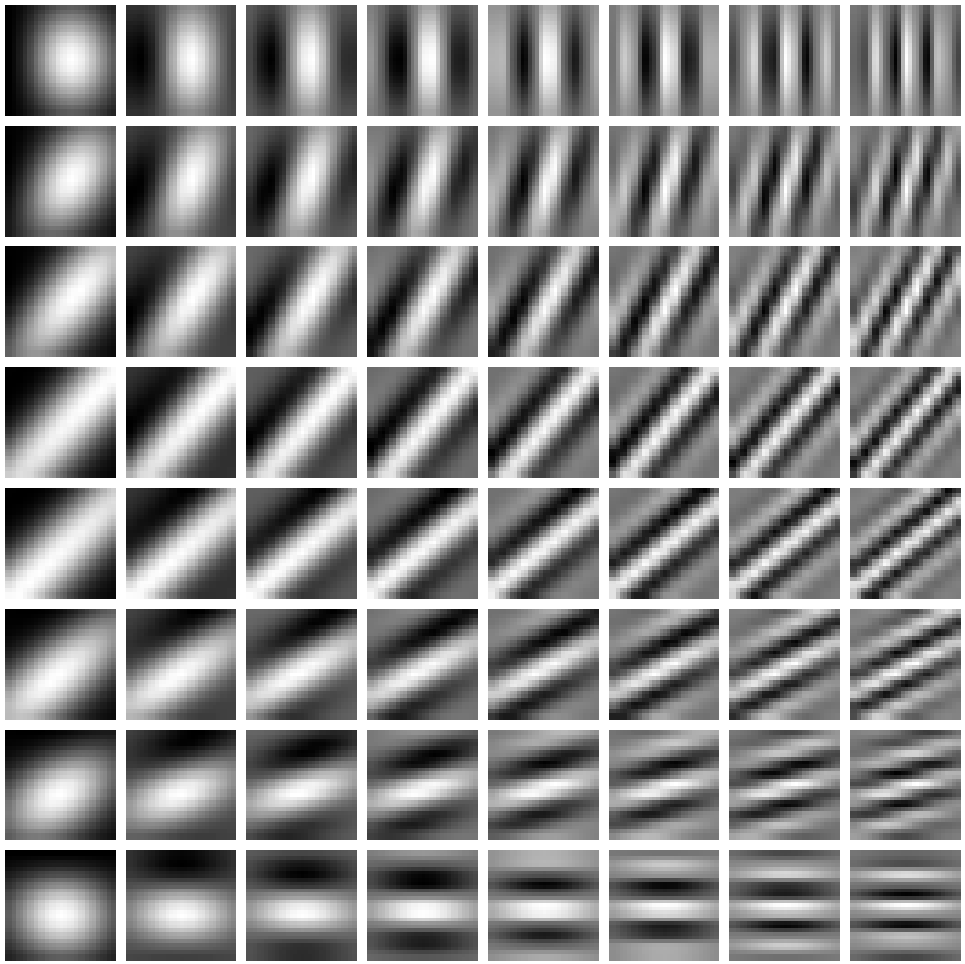
The response of the simple cells can be modelled using Gabor filters:

$$
\begin{aligned}
G(x, y) &= A \exp(-\alpha \tilde{x}^2 - \beta \tilde{y}^2) \sin(\omega \tilde{x} + \phi) \\
\tilde{x} &= (x - x_0) \cos\theta + (y - y_0) \sin\theta \\
\tilde{y} &= -(x - x_0) \sin\theta + (y - y_0) \cos\theta
\end{aligned}
$$

- The $\sin$ term represents a sinusoidal spatial oscillation oriented in a direction defined by the polar angle θ, with frequency ω and phase angle ϕ.
- The exponential factor creates a decay envelope that localizes the filter in the neighborhood of position (x_0, y_0) and with decay rates governed by α and β.

Visual cortex

Figure: Examples of Gabor filters



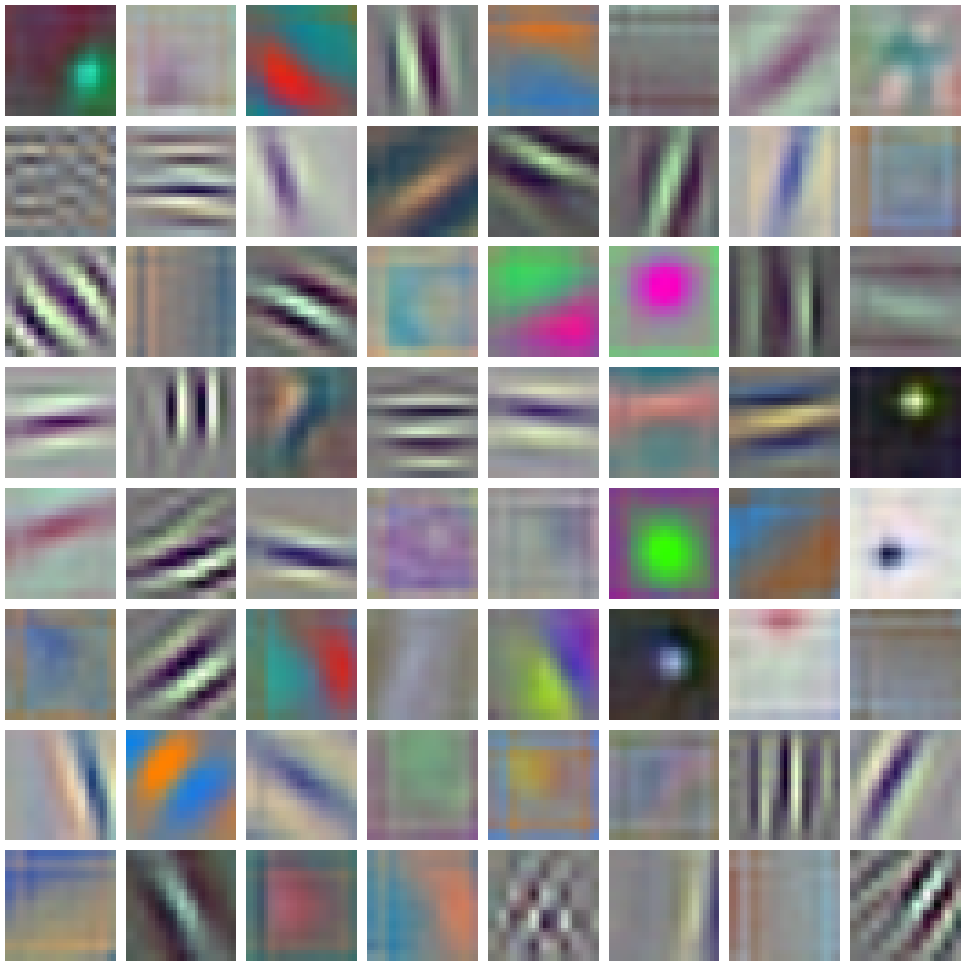
Visualizing trained filters

For the filters in the first convolutional layer, we can visualize the network weights associated with these filters directly as small images (Cauchy-Schwarz inequality).

We see a remarkable similarity between these filters and the Gabor filters. This is because these characteristic filters are a general property of the statistics of natural images and therefore prove useful for image understanding in both natural and artificial systems.

Visualizing trained filters

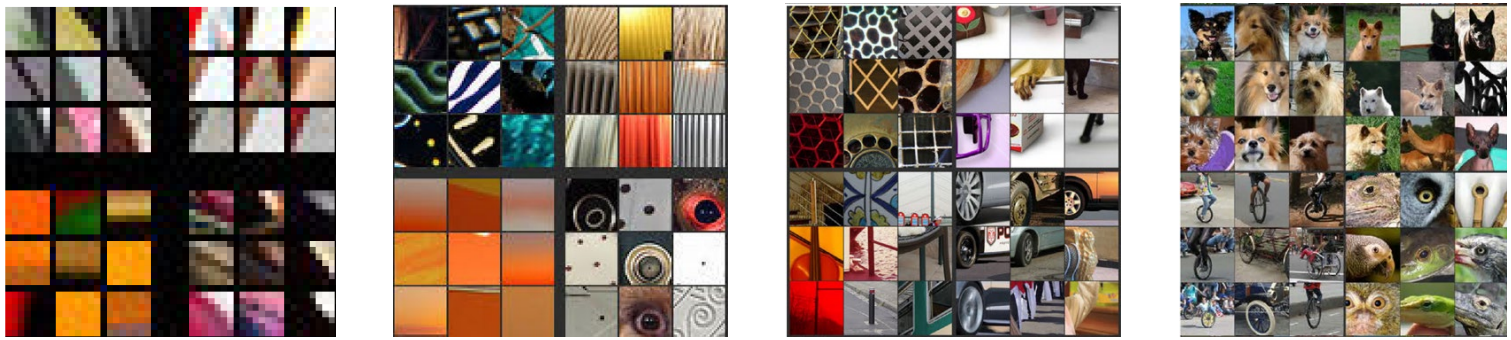
Figure: Examples of learned filters from the first layer of AlexNet



Visualizing trained filters

One approach to visualize the filters in subsequent layers is to present a large number of image patches to the network and see which produce the highest activation value in any particular hidden unit.

Figure: Examples of image patches that produce the strongest activation in the hidden units in a network

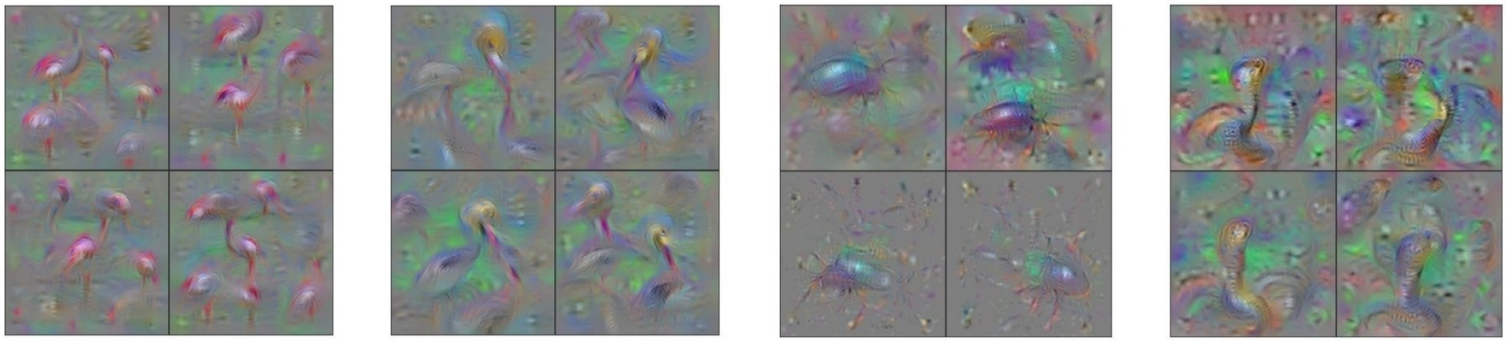


Layer 1 | Layer 2 | Layer 3 | Layer 5

Visualizing trained filters

We can extend this technique to perform a numerical optimization over the input variables to maximize the activation of a particular unit. If we chose the unit to be one of the outputs then we can look for an image that is most representative of the corresponding class label.

Figure: Examples of synthetic images generated by maximizing the class probability



Saliency maps

- Saliency maps aim to identify those regions of an image that are most significant in determining the class label.
- This is best done by investigating the final convolutional layer:
  - This layer still retains spatial localization.
  - This layer has the highest level of semantic representation.

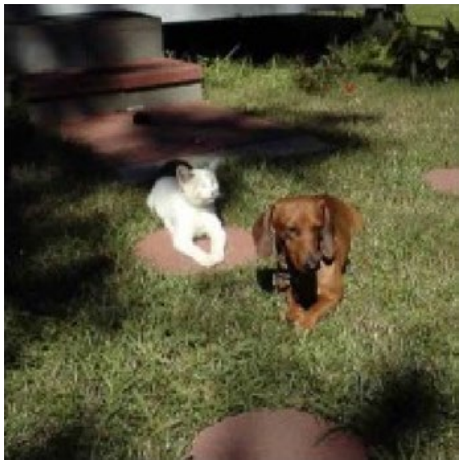
Saliency maps

The Grad-CAM (gradient class activation mapping) method. For a given input image:

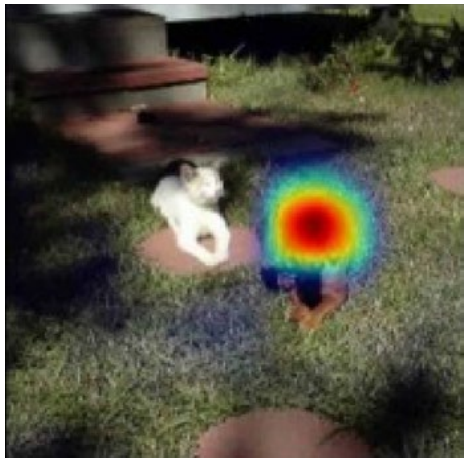
- First compute the derivatives of the output-unit pre-activation a^c for a given class c with respect to the pre-activations $a_{ij}^{(k)}$ of all the units in the final convolutional layer for channel k.
- For each channel in that layer, the average of those derivatives is evaluated to give: $\alpha_k = \frac{1}{M_k} \sum_i \sum_j \frac{\partial a^c}{\partial a_{ij}^{(k)}}$, where M_k is the total number of units in that channel.
- These averages are then used to form a weighted sum of the form: $L = \sum_k \alpha_k A^{(k)}$, where $A^{(k)}$ is a matrix with elements $a_{ij}^{(k)}$.

Saliency maps

Figure: Saliency maps for the VGG-16 network with respect to the dog and cat categories



Original image



Saliency map for ‘dog’

Saliency map for ‘cat’

Adversarial attacks

- Adversarial attacks involve making very small modifications to an image, at level that is imperceptible to a human, which cause the image to be misclassified by the neural network.
- The fast gradient sign method: Changing each pixel value in an image x by a fixed amount ϵ with a sign determined by the gradient of an error function $E(x, t)$ with respect to the pixel values: $x' = x + \epsilon \text{sign}(\nabla_x E(x, t))$, where t is the true label of x.

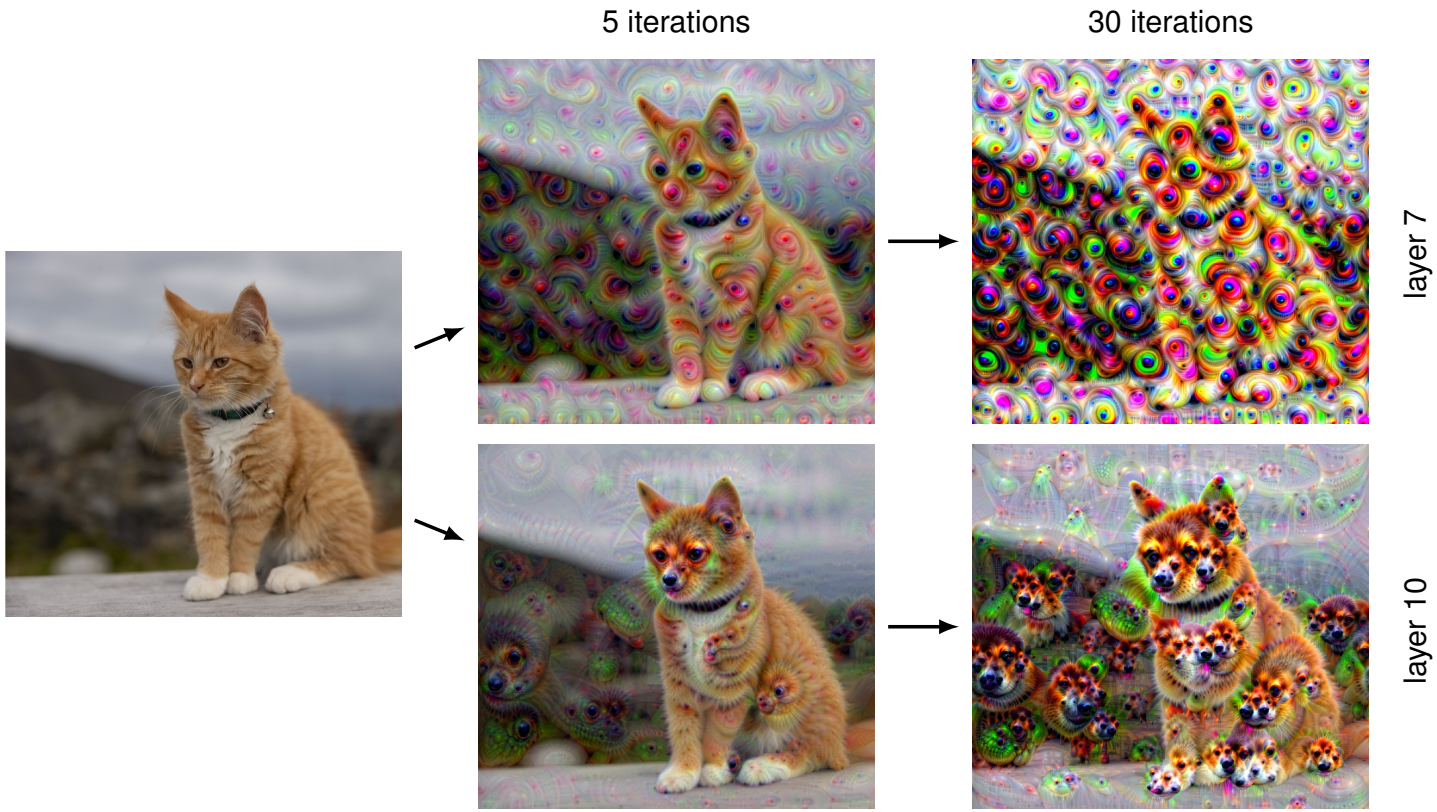
Synthetic images

DeepDream: Determine which nodes in a particular hidden layer of the network respond strongly to a particular image and then modify the image to amplify those responses:

- We apply an image to the input of the network and forward propagate through to some particular layer.
- We then set the backpropagation δ variables for that layer equal to the pre-activations of the nodes and run backpropagation to the input layer to get a gradient vector over the pixels of the image.
- Finally, we modify the image by taking a small step in the direction of the gradient vector.

Synthetic images

Figure: Examples of DeepDream applied to an image



Bounding boxes

- When there are multiple objects in an image, we may want to detect the presence and class of each object as well as their locations.
- The location of an object in an image is usually defined by a bounding box: $b = (b_x, b_y, b_W, b_H)$, where (b_x, b_y) is the coordinates of the center of the bounding box, and b_W and b_H its width and height.

Intersection-over-union

- We need a meaningful way to measure the performance of a trained network that can predict bounding boxes.
- Intersection-over-union (IoU) is the ratio of the area of the intersection of the predicted bounding box and the ground truth bounding box to that of their union:
  - IoU lies in the range $[0, 1]$.
  - Predictions can be labelled as correct if the IoU exceeds a threshold, which is typically set at 0.5.

Sliding windows

The sliding window way of object detection and object localization:

- Create a training set consisting of tightly cropped examples of the object to be detected, as well as examples of similarly cropped sections of images that do not contain any object (the background class).
- Train a classifier (e.g., a deep CNN) using the data set, whose outputs represent the probability of there being an object of each particular class in the input window.
- Scan an input window across the image and, for each location, taking the resulting subset of the image as input to the classifier.
- When an object is detected with high probability, the associated window location then defines the corresponding bounding box.

Sliding windows

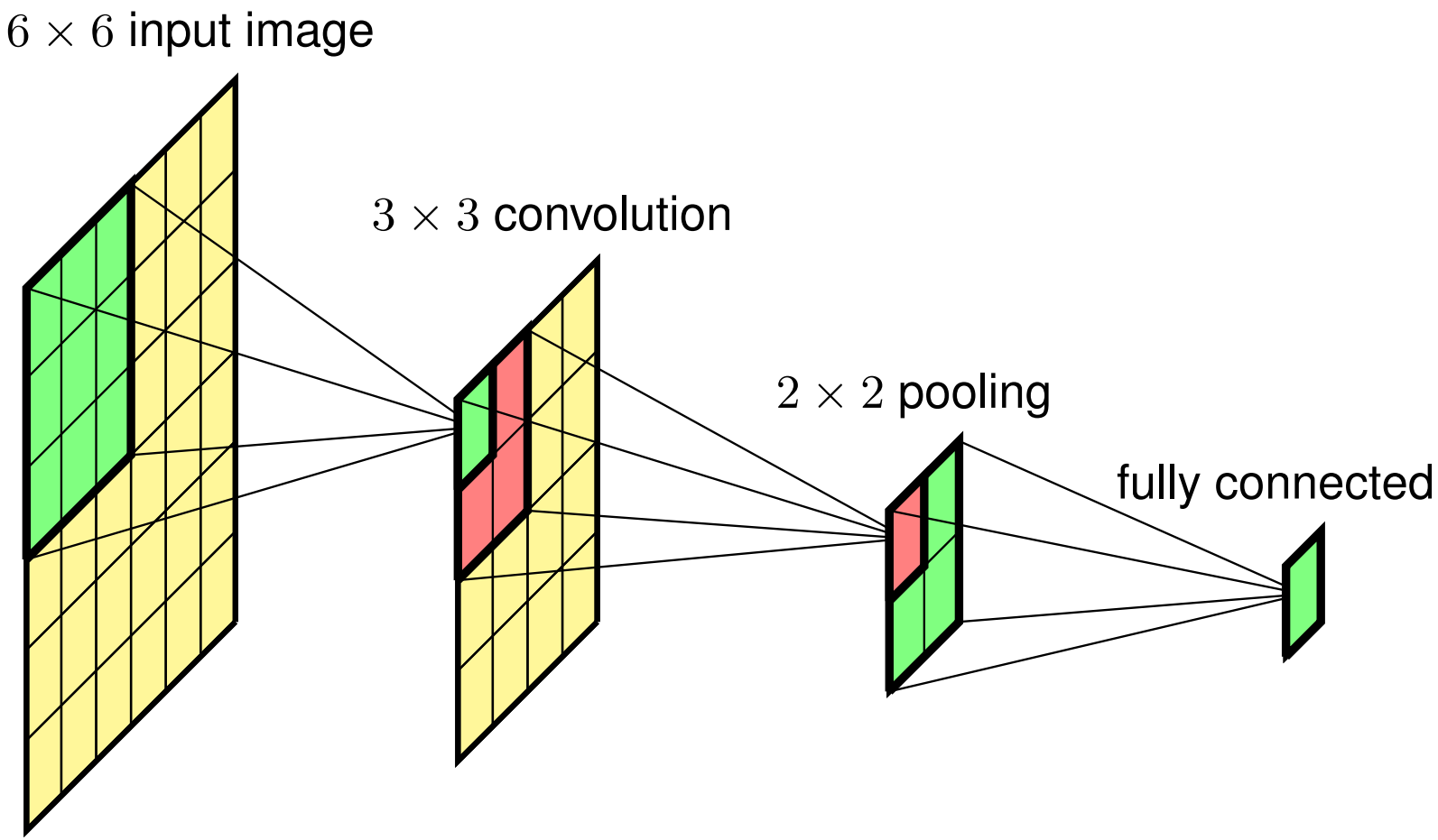
Potential drawbacks of the sliding window solution:

- It can be computationally very costly due to the large number of potential window positions in the image.
  - Cost can be saved by moving the input window in strides across the image.
  - There is a trade-off between precision of location using a small stride and reducing the computational cost by using a larger stride.
- The process may have to be repeated using windows of various scales to allow for different sizes of object within the image.

Fortunately, it is remarkably simple to implement sliding windows efficiently in a convolutional network.

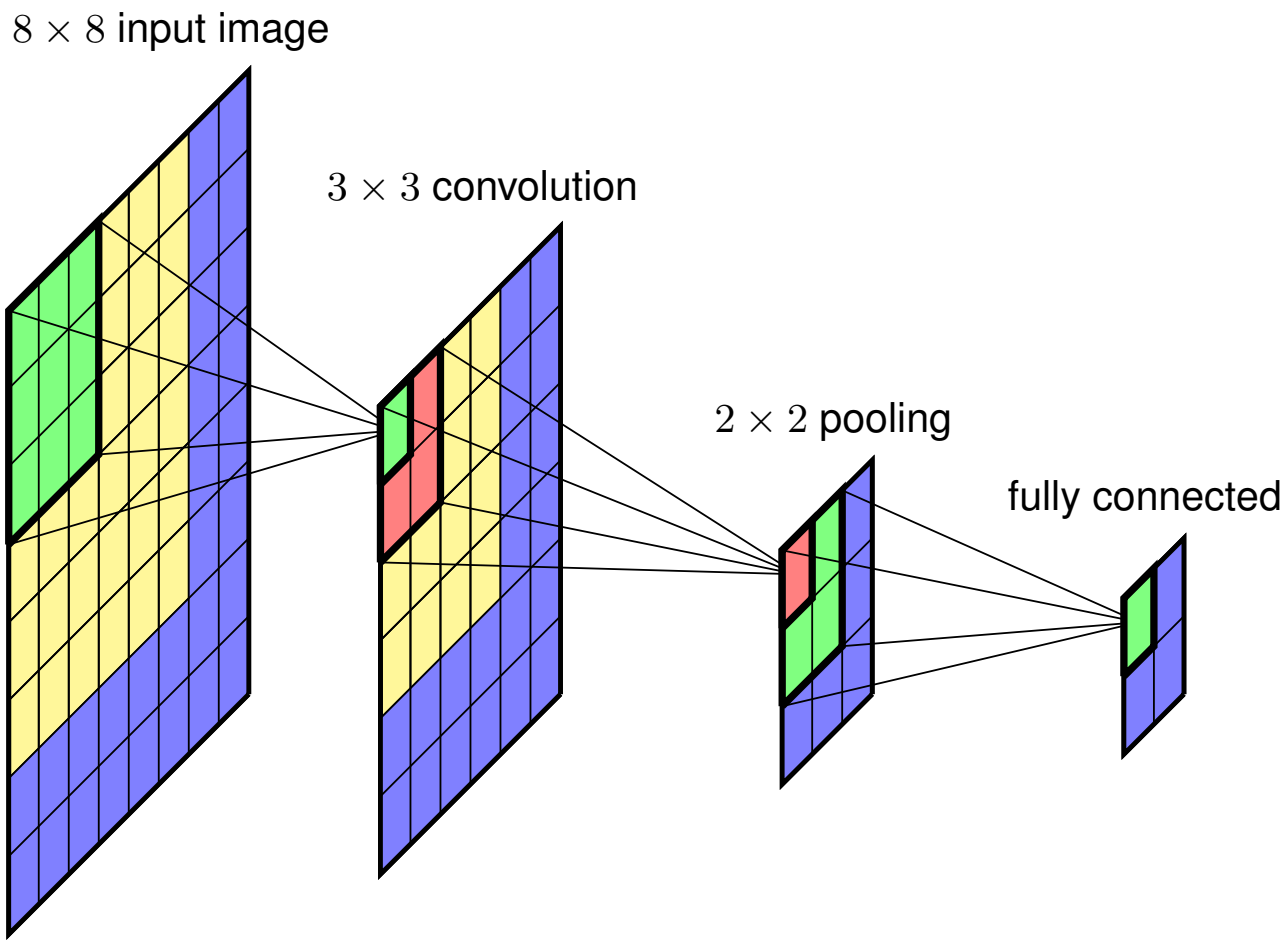
Sliding windows

Figure: Example of a simple convolutional network



Sliding windows

Figure: Application of the network to a larger image



Detection across scales

We still need to solve the problem of detecting objects at different scales and at different aspect ratios:

- [No] Use multiple detectors with different sizes and shapes of input window.
- [Yes] Use a fixed input window and to make multiple copies of the input image each with a different pair of horizontal and vertical scaling factors.
- The associated scaling factors are then used to transform the bounding box coordinates back into the original image space.

Non-max suppression

By scanning a trained convolutional network over an image, it is possible to produce multiple detections of the same object at similar locations. This can be addressed using non-max suppression. For each object class:

1. Run the sliding window over the whole image and evaluates the probability of an object of that class being present at each location.
2. Elimiate all the associated bounding boxes whose probability is below some threshold.
3. The box with the highest probability is considered to be a successful detection, and the corresponding bounding box is recorded as a prediction.
4. Any other boxes whose IoU with the successful detection box exceeds some threshold is discarded.
5. Go to step 3 and repeat until all bounding boxes have either been discarded or declared as successful detections.

Fast region CNNs

Instead of applying the full power of a deep convolutional network to all areas of the image, we can apply some form of computationally cheaper technique to identify parts of the image where there is a higher probability of finding an object, and then apply the full network only to these areas:

- Fast region proposals with CNN (fast R-CNN).
- Use a region proposal convolutional network to identify the most promising regions: faster R-CNN.

Convolutional segmentation

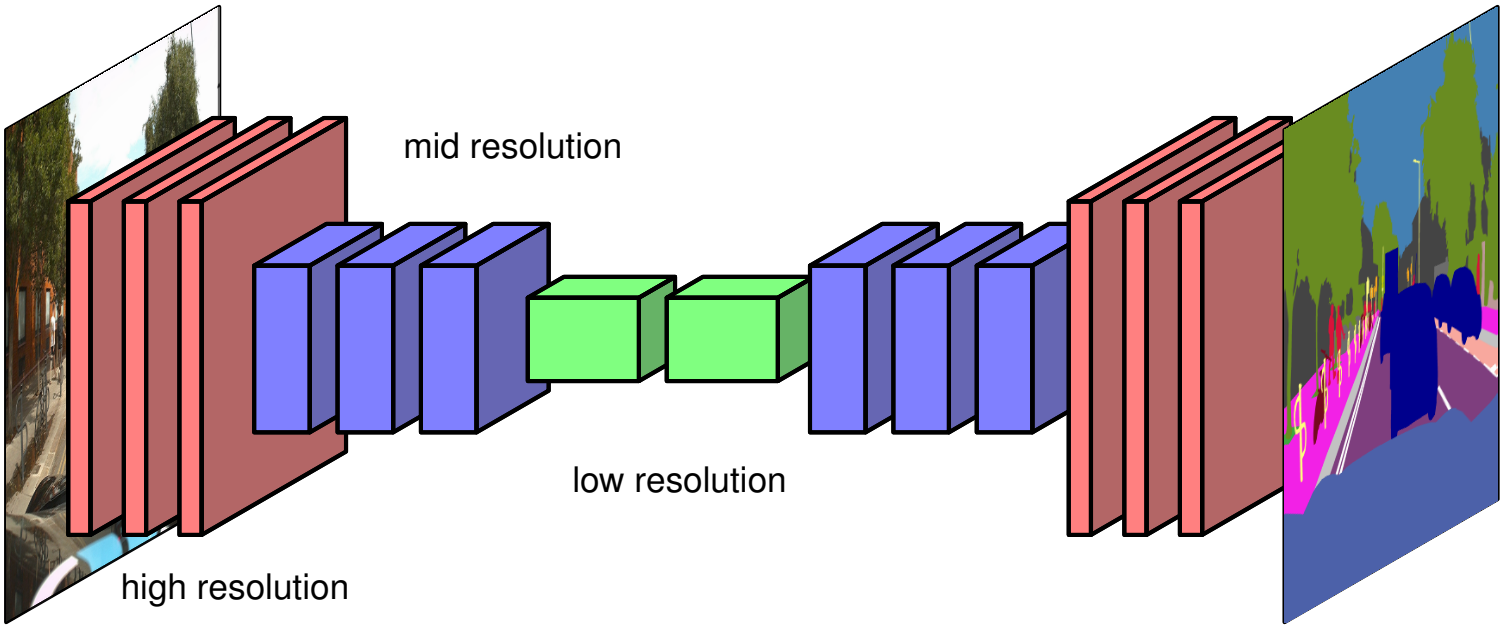
Semantic segmentation: Every pixel of an image is assigned to one of a predefined set of classes.

- Create a CNN in which each layer has the same dimensionality as the input image:
  - Stride 1 at each layer.
  - Same padding.
  - No pooling.
- Each output unit has a softmax activation function with weights that are shared across all outputs.
- Such a network would still need many layers, with multiple channels in each layer, and overall this would be prohibitively costly for images of reasonable resolution.

Up-sampling

We can create a more efficient architecture for semantic segmentation by taking a standard deep convolutional network and adding additional learnable layers that take the low-dimensional internal representation and transform it back up to the original image resolution.

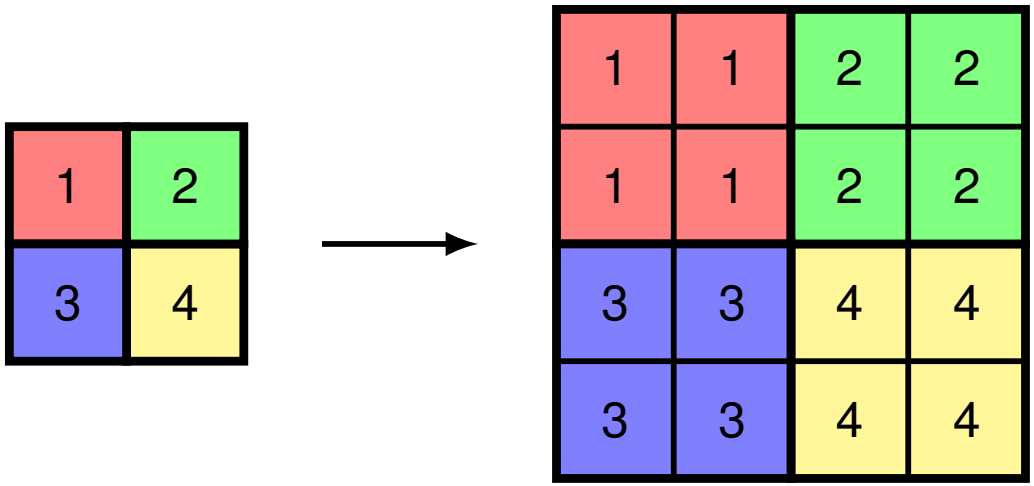
Figure: Illustration of a CNN used for semantic image segmentation



Up-sampling

To reverse the down-sampling effects of average pooling, we can copy over each input value into all the corresponding output units.

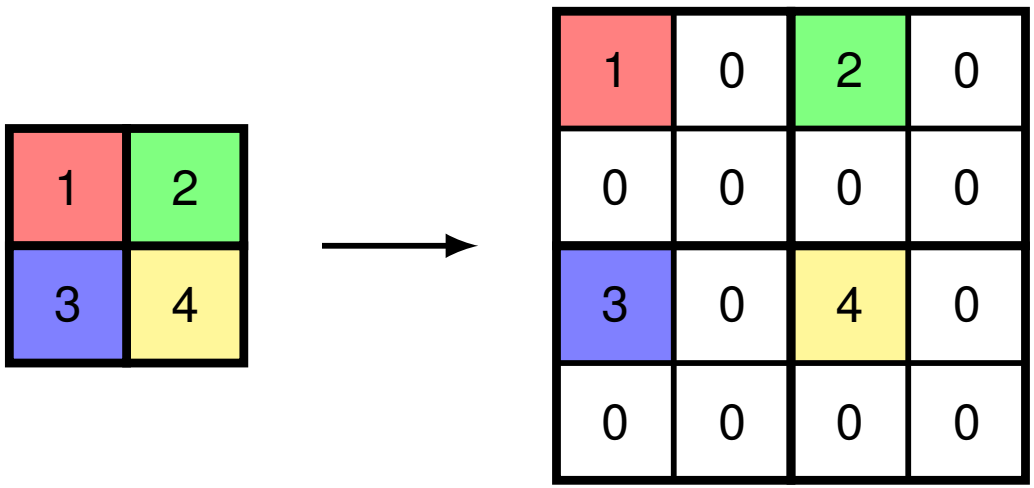
Figure: Unpooling operation for average pooling



Up-sampling

To reverse the down-sampling effects of max-pooling, we can copy the input value into the first unit of the corresponding output block, and the remaining values in each block are set to zero.

Figure: Unpooling operation for max-pooling



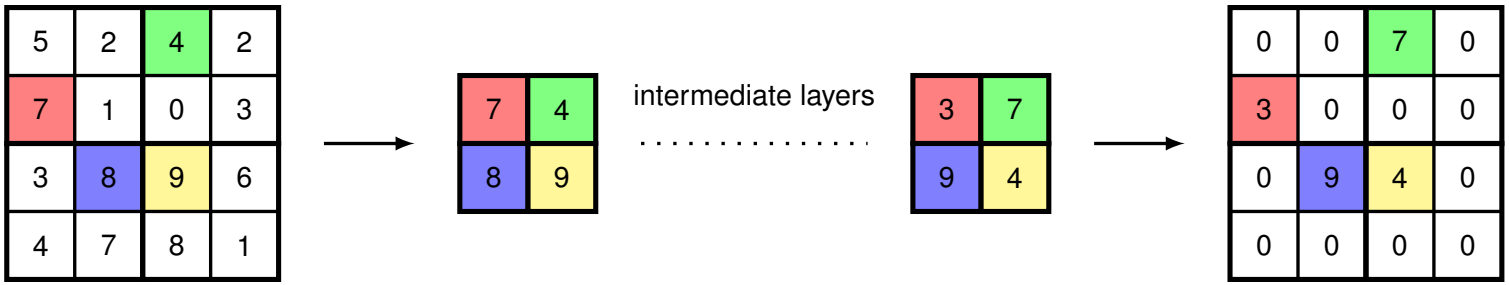
Up-sampling

Or we can:

- Choose a network architecture in which each max-pooling down-sampling layer has a corresponding up-sampling layer later in the network.
- During down-sampling, a record is kept of which element in each block had the maximum value.
- In the corresponding up-sampling layer, the non-zero element is chosen to have the same location.

Up-sampling

Figure: Unpooling operation for max-pooling, with spatial information kept



Fully convolutional networks

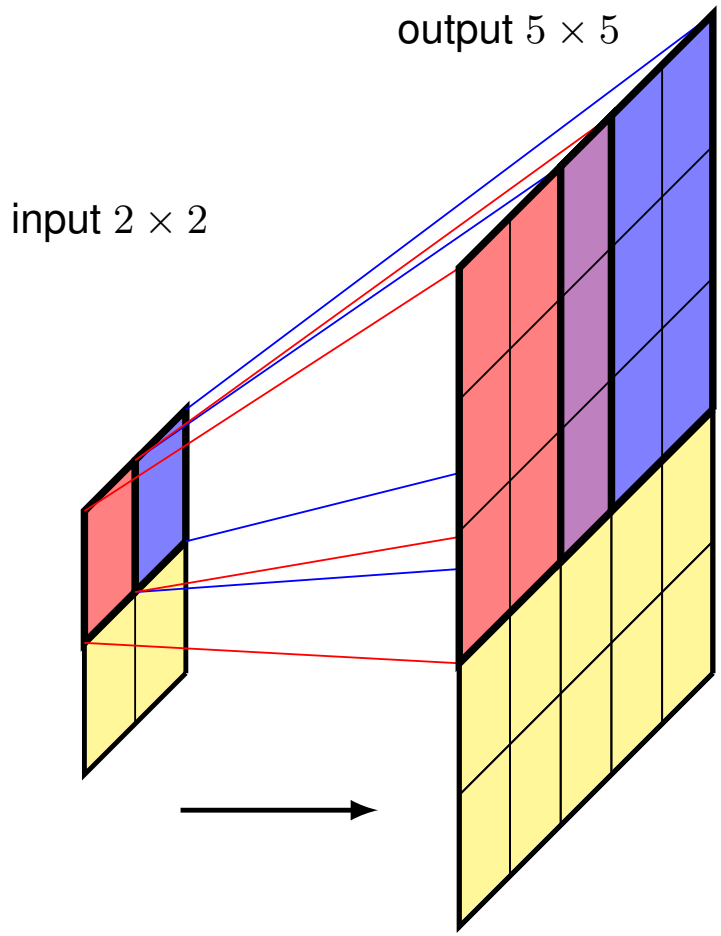
We can also use a learned up-sampling, which is the reverse of strided convolution:

- Use a filter that connects one pixel in the input array to a patch in the output array.
- Choose an architecture so that as we move one step across the input array, we move two or more steps across the output array.
- If there are output cells for which multiple filter positions overlap, we can calculate the output values by summing or by averaging the contributions from the individual filter positions.

If we have a network architecture with no pooling layers, so that the down-sampling and up-sampling are done purely using convolutions, then the architecture is known as a fully convolutional network.

Fully convolutional networks

Figure: Illustration of transpose convolution



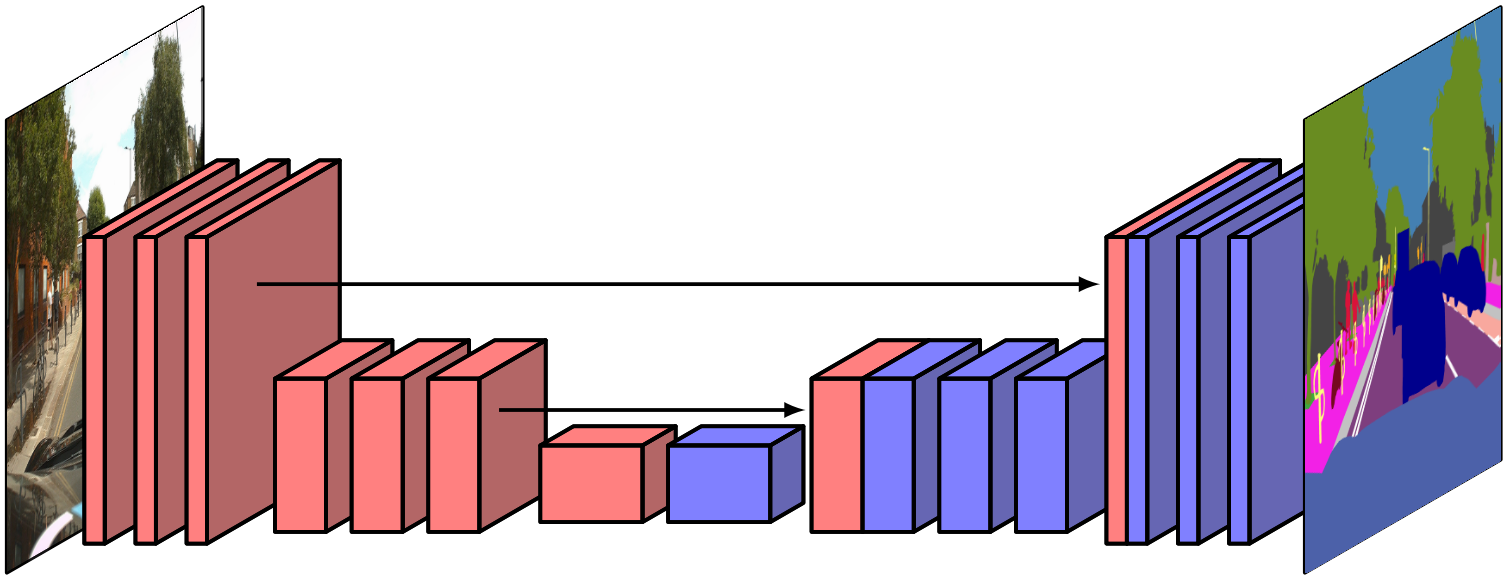
The U-net architecture

To maintain the spatial information discarded during down-sampling, we can use the U-net architecture:

- For each down-sampling layer there is a corresponding up-sampling layer.
- The final set of channel activations at each down-sampling layer is concatenated with the corresponding first set of channels in the up-sampling layer.
- 1×1 convolutions may be used in the final layer of a U-net to reduce the number of channels down to the number of classes.

The U-net architecture

Figure: The U-net architecture



Style transfer

Neural style transfer: Generate a synthetic image G whose content is defined by an image C and whose style is taken from some other image S. This is achieved by defining an error function $E(G)$ given by the sum of two terms:

$$E(G) = E_{\text{content}}(G, C) + E_{\text{style}}(G, S)$$

We can then find G by starting from a randomly initialized image and using gradient descent to minimize $E(G)$.

Style transfer

To define $E_{\text{content}}(G, C)$:

- Pick a particular convolutional layer in the network.
- Measure the activations of the units in that layer when image G is used as input and also when image C is used as input.

We can then encourage the corresponding pre-activations to be similar by using a sum-of-squares error function of the form:

$$E_{\text{content}}(G, C) = \sum_{i,j,k} (a_{ijk}(G) - a_{ijk}(C))^2$$

where a_{ijk} denotes the pre-activation of the unit at position (i, j) in channel k of that layer.

Style transfer

To define $E_{\text{style}}(G, S)$, the intuition is that style is determined by the co-occurrence of features from different channels within a convolutional layer. Again consider a particular convolutional layer, we can measure the extent to which a feature in channel k co-occurs with the corresponding feature in channel k':

$$F_{kk'}(G) = \sum_{i=1}^{I} \sum_{j=1}^{J} a_{ijk}(G) a_{ijk'}(G)$$

where I and J are the dimensions of the feature maps in this particular convolutional layer. If there are K channels in this layer, then $F_{kk'}$ form the elements of a $K \times K$ matrix, called the style matrix.

Style transfer

We can measure the extent to which the two images G and S have the same style by:

$$E_{\text{style}}(G, S) = \frac{1}{(2IJK)^2} \sum_{k=1}^{K} \sum_{k'=1}^{K} (F_{kk'}(G) - F_{kk'}(S))^2$$

We could obtain more pleasing results by using contributions from multiple layers:

$$E_{\text{style}}(G, S) = \sum_{l} \lambda_l E_{\text{style}}^{(l)}(G, S)$$

where the coefficients λ_l determine the relative weighting between the different layers.