CITY
UNIVERSITY OF LONDON
EST 1894

Introduction to AI

Lecture 8: Convolutional Neural Networks and Image Classification

School of Mathematics, Computer Science and Engineering
Department of Computer Science

Overview of today's lecture

✓Images and MLPs

✓Convolution Neural Networks

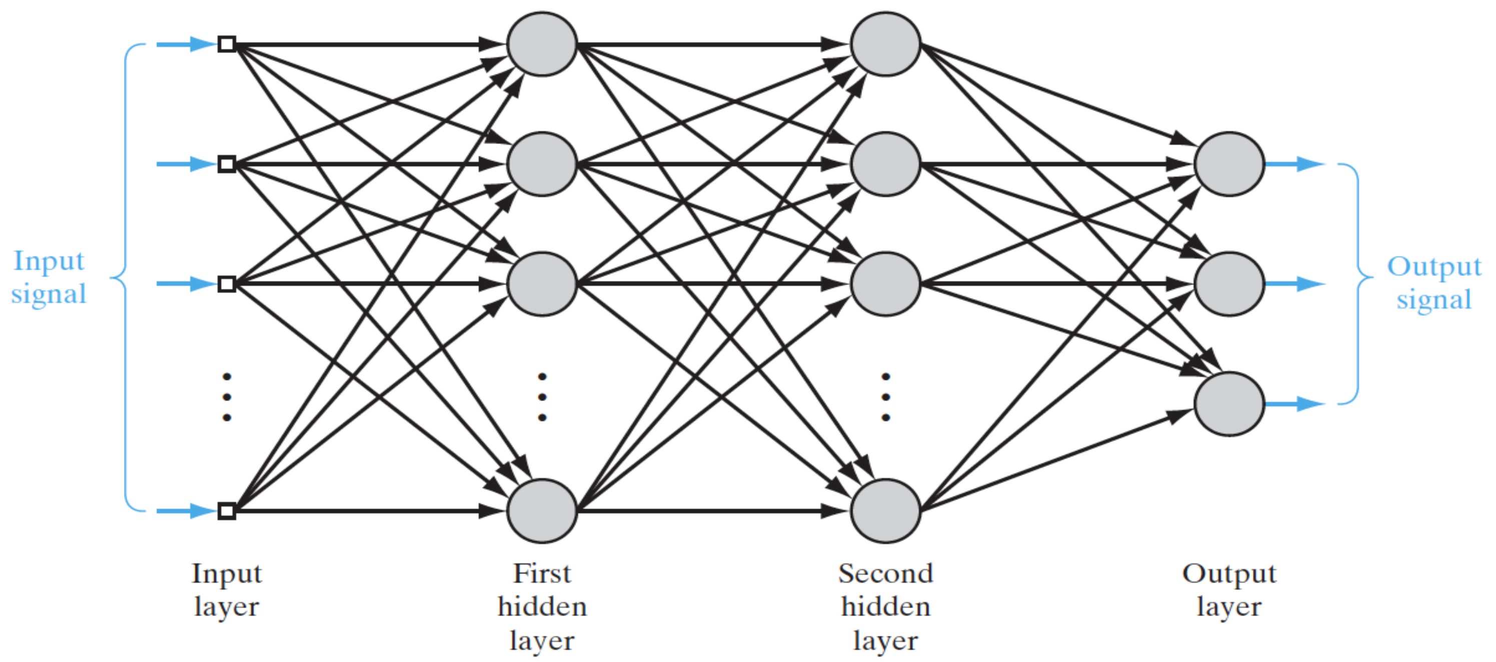
✓Data Augmentation

Images

- ❑Image classification is a key task in Artificial Intelligence
- ❑If you want a self-driving cars, then they need to be able to take in input, including visual input and work out (in some sense) what it is that they are looking at
- ❑Or (as we have seen repeatedly) it’s a useful computational task to be able to automatically read characters
- ❑Images, from a computer’s point of view, are just a bunch of pixels, or rather, just a structured collection of numbers

Neural Networks recap (again)

A fully connected multilayer perceptron with two hidden layers and an output layer:



Images and MLPs

- ❖Images can be fed into MLPs as input vectors
- ❖Take the 2d input from a greyscale image (or the 3d input from a colour image), and flatten it to a vector. Then train as usual
- ❖This will probably work for small examples
- ❖But… it doesn't scale very well
- ❖Importantly, this approach seems to have lost something of what an image is. The pixels aren't just independent dots, they are related to the pixels around them, usually this correlation is higher locally.
- ❖In MLPs each weight is a priori independent and used once. If an image has similar low-level structures that appear in different locations (e.g. edges), the MLP would need to repeatedly tune the weights corresponding to processing in different locations to capture the instances of the low-level structures

CITY UNIVERSITY OF LONDON EST 1894

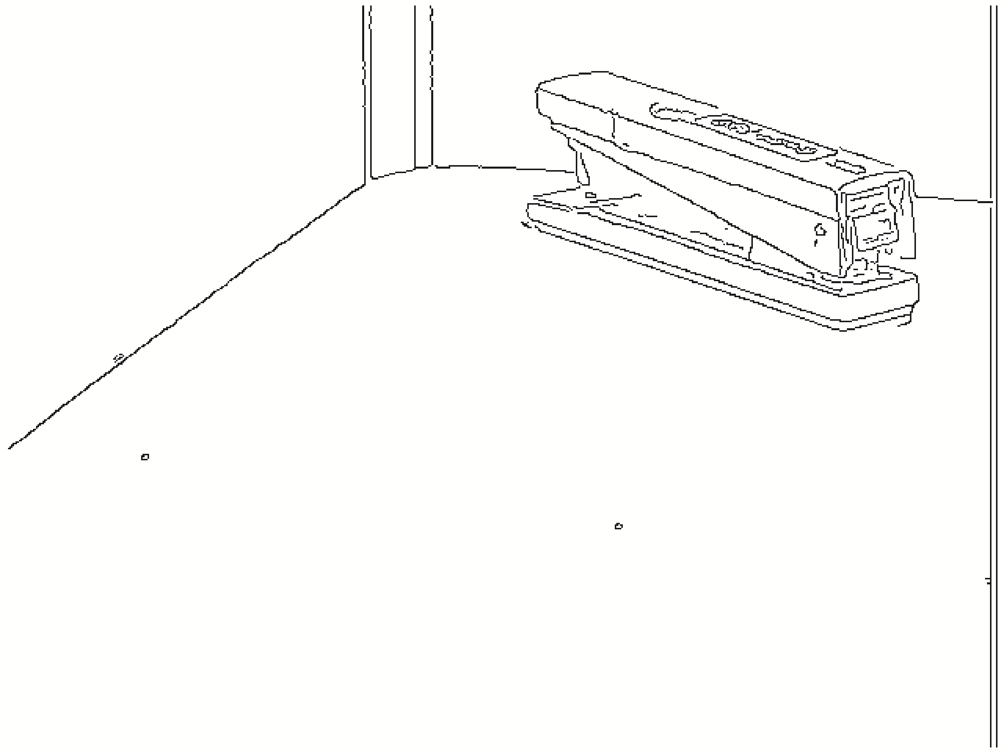
Images and machine learning

Relational information between pixels:

- ➢Images can be translated
- ➢Images can be rotated
- ➢Images can be scaled
- ➢… and so on

In many cases, we want the identity of the image to still be recognized after these transformations

Traditional Image Analysis



Edge detection

Edge Detection

- One approach to edge detection is to apply the Sobel filter
- Consider the image represented as a matrix
- Also, consider the following two matrices, examples of convolution kernels:

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \qquad G_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

Edge Detection

- These are applied to the image matrix pixel by pixel
- They are applied by matching the kernel matrix against the image matrix A, so that the centre of G_x is over a pixel, and the output is the result multiplying the elements of G_x and the corresponding elements of A.
- If the pixels matched against are uniform, then the result is 0. Otherwise, a gradient is produced.

Edge Detection

- The same is done for G_y
- The results are then combined, $\sqrt{G_x^2 + G_y^2}$
- The result of applying the filters pixel by pixel across the image is the kind of edge detection seen above

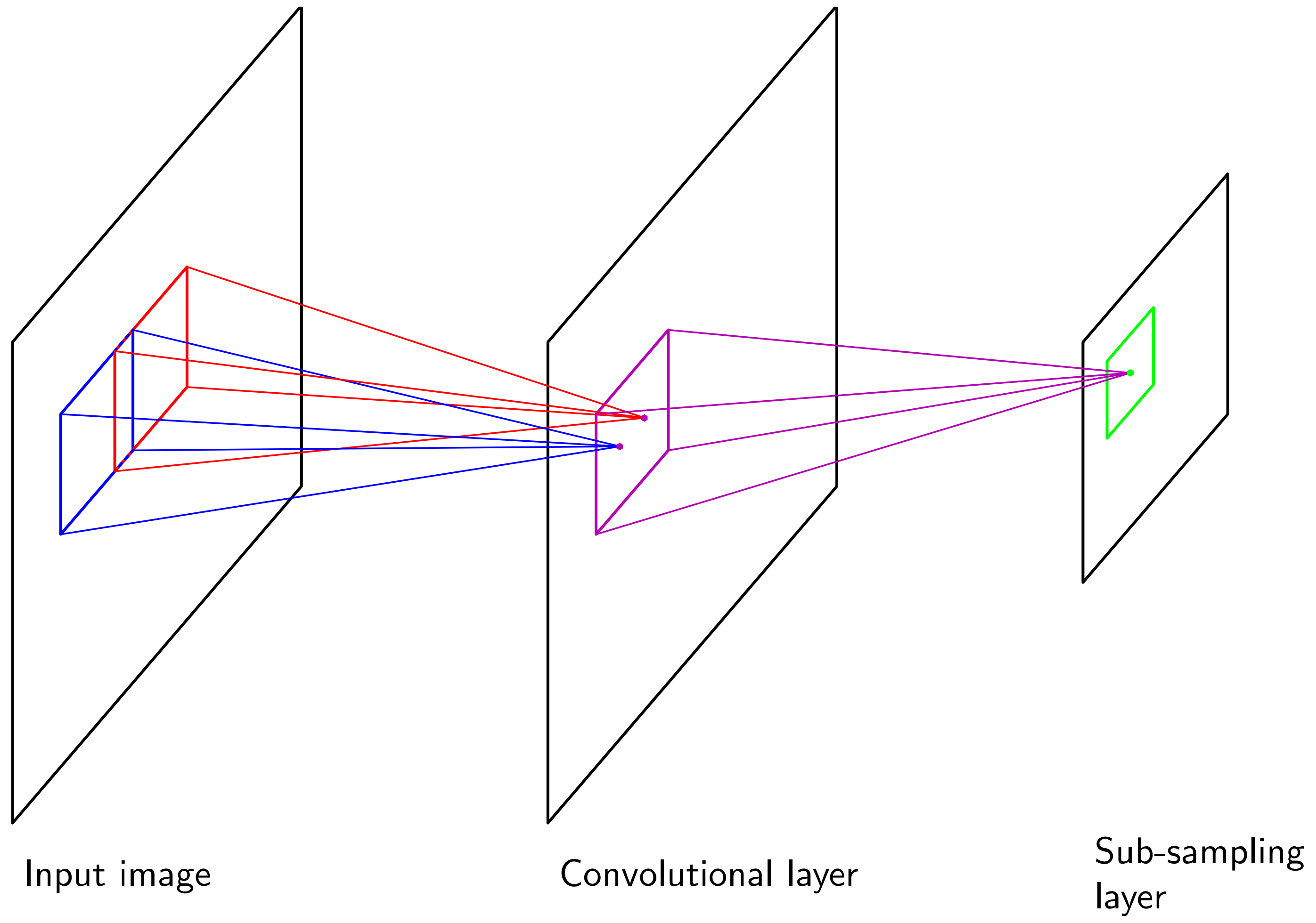
Convolutional Neural Networks

❖The feed-forward neural networks we have looked at so far are typically fully connected

- Convolutional neural networks (we'll write CNN) provide a way to specialize neural networks to work with grid-structured data
- This allows these models to scale to a large size and still work
- That is, we are not looking at fully connected networks here
- CNNs impose a priori structure to the weight connections, instead of expecting that this structure is learned during training: *sparse local* connectivity and *parameter sharing*.

Convolutional Neural Networks

- Small regions of the image data will be connected to a neuron
- There will be weight sharing across neurons
- There will be pooling layers
- There might be lots of layers



Overview of CNNs

The main components of a CNN are the following:

- Filters, which provide the structure of the maps in the convolutional layers. Important: in contrast to the use of filters such as Sobel filters, these filters are learned during training
- Pooling of neurons (down-sampling)
- Flattening of a layer
- Dense, fully connected layer(s)
- Output layer
- CNNs are deep networks and there might be several of these layers

What’s a convolution?

- ❖The convolution in CNNs refers to a linear kernel operation which slides about the input
- ❖We have just seen one of these for the Sobel filter used with edge detection

$$Y(i,j) = \sum_{k=-a}^{a} \sum_{k'=-a}^{a} X(i-k, j-k')K(k,k') \qquad X * K = \begin{pmatrix} x_{11} & x_{12} & x_{13} & x_{14} \\ x_{21} & x_{22} & x_{23} & x_{24} \\ x_{31} & x_{32} & x_{33} & x_{34} \\ x_{41} & x_{42} & x_{43} & x_{44} \end{pmatrix} * \begin{pmatrix} K_{11} & K_{12} \\ K_{21} & K_{22} \end{pmatrix}$$

$$y_{11} = K_{11}\, x_{11} + K_{21}x_{21} + K_{12}x_{12} + K_{22}x_{22} \qquad y_{23} = K_{11}\, x_{23} + K_{21}x_{33} + K_{12}x_{24} + K_{22}x_{34}$$

Note: the exact mapping of indexes between X and Y depends on the implementation, e.g. kernel (-a,a) or (0, 2a).

Note: The borders require special treatment: e.g. how to calculate y_{44}? We will see use of ‘padding’

Note: Here the kernel is slid with a 1 by 1 step, more generally we will see that the parameter *stride* determines this

Input Tensors

❖The input to a CNN is a tensor

❖Tensors are the higher dimensional generalisation of vectors and matrices

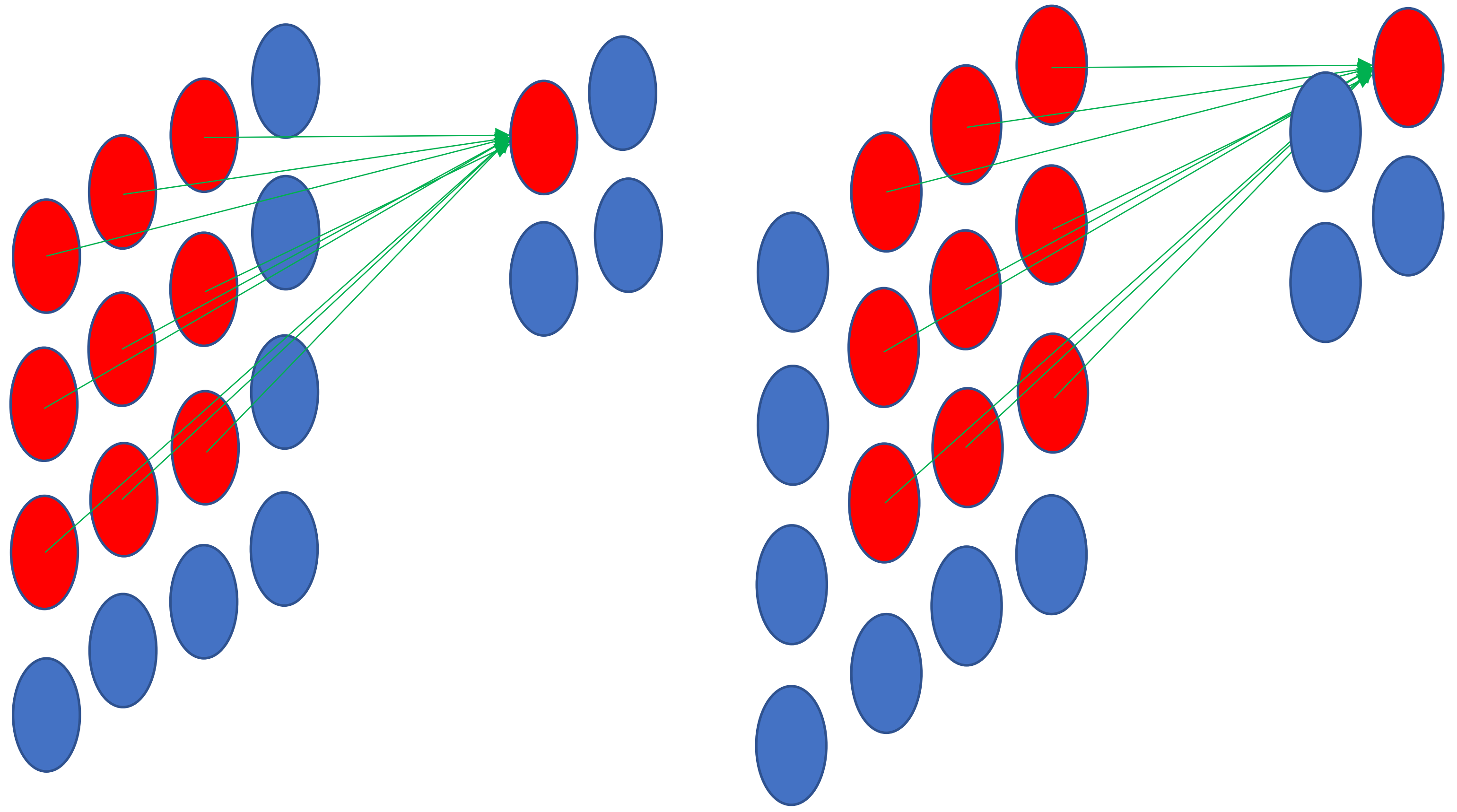
❖For processing images, they will be of shape *x y z.*

❖For colour images, the *z* dimension will be three deep for RGB. That is, the *x* and *y* define the coordinates of the pixel and the *z* the colour

❖For greyscale images, the *z* dimensional will be a single value

Convolutional Layers

- ❖In a CNN convolutional layer, incoming edges come from a small number of neurons in the previous layer, namely a small subgrid of neurons.



The units of the previous layer *k-1* (or pixels if it's the first layer) that are combined with the kernel to determine the activity of unit *u* in layer *k,* are called **the *receptive field*** of *u*
The output of applying the convolution operation to the whole layer is called a ***feature map***

Note: no padding, the dimension is reduced because convolution is not applied to rows and columns 3-4

Convolutional Layers

- ❖With this structure in place, there are three parameters
- ❖The first is the size of the subgrid (the kernel size) that links to an output neuron.
- ❖The kernel size is typically chosen to be small (say, 3 x 3, 4 x 4, 5 x 5), except maybe in the first layer (say 5 x 5, 7 x 7, 11 x 11)
- ❖The second is the step size (the stride). This is the offset from one subgrid to the next.
- ❖Again, this is typically small (say, 1 or 2). Usually, you want the subgrids to overlap

Convolutional Layers

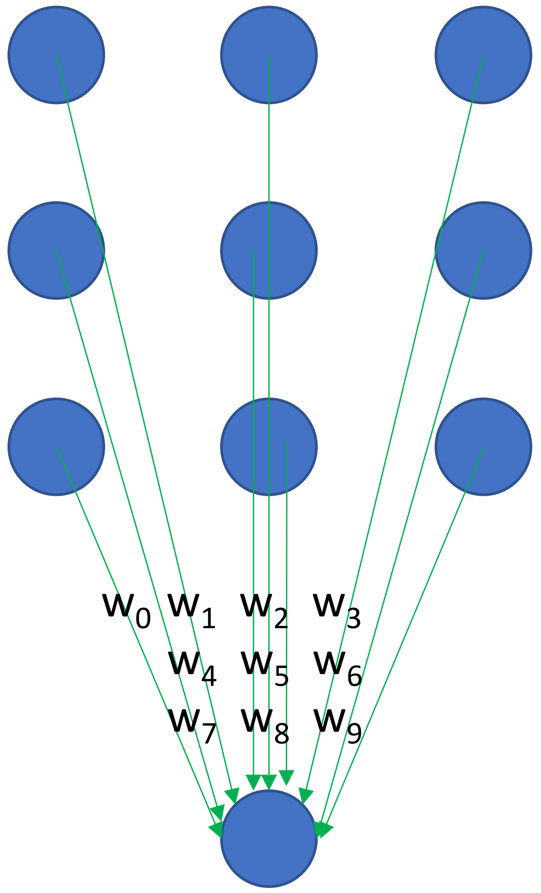
- ❖The third is whether or not to include padding
- ❖Padding adds extra pixels around the image (usually with value 0) so that with stride=1 the number of output neurons is the same as the number of input neurons (other padding options are wrap or mirror pixels, used less often)
- ❖Note: Keras padding = “same” adds 0s, padding=“valid” (default) uses no padding and reduces the size by the size of the kernel-1.

Sparse local connectivity

- ❖With a stride of 1, and using padding, the output layer has the same size as the input layer
- ❖With a fully connected network, this would lead to a problematic number of weight parameters to train (for a 64 x 64 greyscale image: flattened is 2^6 x 2^6 = 2^{12} 1D vector; connections between layers fully connected, with output layer of same size, 2^{12} x 2^{12} = 2^{24} ~ 16 million)
- ❖With a grid size of 3 x 3 and padding, with an output layer of same size, we get a rather better 10 x 64 x 64 = 36864 parameters (10 = 3 x 3 + 1, with +1 being the bias).
- ❖This is if there was *sparse connectivity* but the parameters of the grid at each location were independent; further reduction comes from *parameter sharing*.

Parameter sharing

- ❖In the fully connected MLP models, each weight is independent and is used only once
- ❖For CNNs, parameter sharing is used
- ❖For each unit in a convolutional layer, the input weights are the same
- ❖That is, the parameter weight is shared across each sparsely connected neuron in the layer



Filters

- Parameter sharing means that there are lots of neurons in a layer that behave in the same way
- However, these neurons are connected to *different* input nodes
- The effect is to apply a filter *across* the input image
- It is these filters that are being learned by the network
- And each filter then consists of tuning just 10 weights (assuming a 3 x 3 kernel size, plus the bias parameter)
- Note: ReLU is the standardly used activation function

Full Convolutional Layer: pile of feature maps

- ❖The above has described the structure of a single filter to be learned by the network (construction of a single feature map)
- ❖A full convolution layer will consist of a number of filters, stacking feature maps that represent different aspects of the image captured by each filter.
- ❖In this week's examples we use 32 or 64, but it can be hundreds, or more
- ❖A full convolutional layer, with 3 x 3 kernel size, stride of 1, 32 filters, and inputs of size 64 x 64 has (9+1)*32 = 320 weight parameters to learn, and 64 x 64 x 32=2^{17} outputs (the number of parameters is << number of units)

Output of a convolutional Layer from previous feature maps

❖The $a_{ijk}^{(p)} = b_k^{(p)} + \sum_{k'=0}^{K^{(p-1)}} \sum_{u=0}^{l_h^{(p)}-1} \sum_{v=0}^{l_w^{(p)}-1} z_{i'j'k'}^{(p-1)} w_{uvk'k}^{(p)}$ $\quad i'=is_h^{(p)}+u \quad j'=js_w^{(p)}+v$

$a_{ijk}^{(p)}$ activation of neuron of layer *p* in row *i* and column *j* in the feature map *k*

$b_k^{(p)}$ bias of input to neurons in the feature map *k* of layer *p*

$K^{(p-1)}$ number of feature maps in layer *p*-1

$l_h^{(p)}, l_w^{(p)}$ vertical and horizontal size of filters of the input to layer *p*

$z_{i'j'k'}^{(p-1)}$ output of neuron of layer *p*-1 in row *i'* and column *j'* in the feature map *k'*

$s_h^{(p)}, s_w^{(p)}$ vertical and horizontal strides on the input to layer *p*

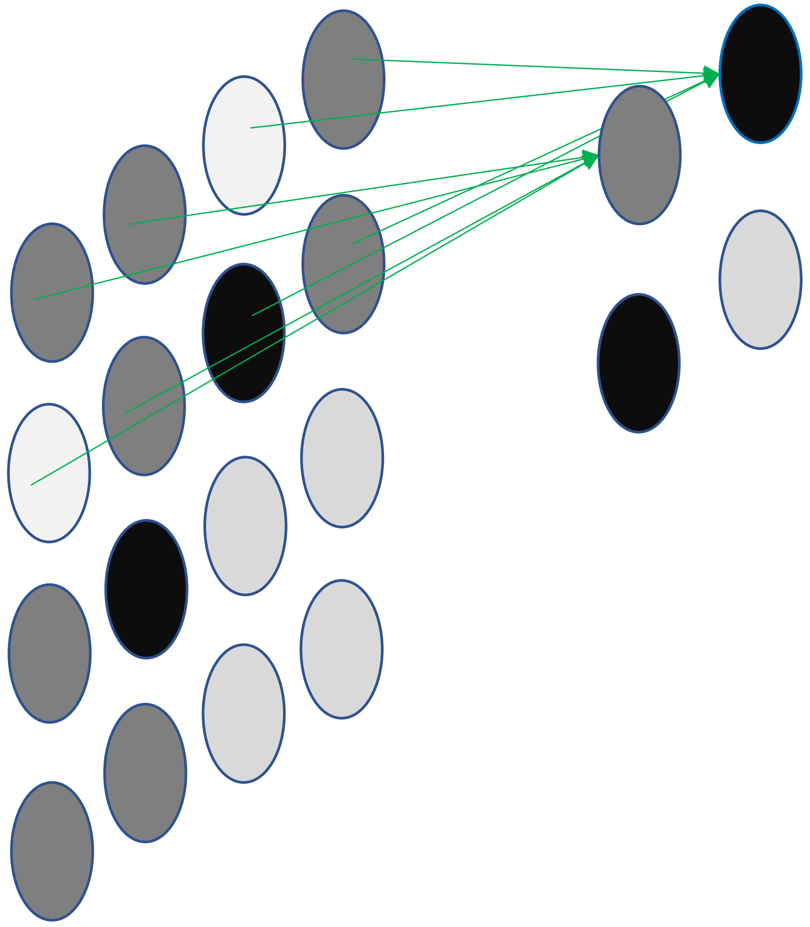
$w_{uvk'k}^{(p)}$ connection weight between feature map *k'* in layer *p*-1 and feature map *k* of layer *p* between the neuron in row *i'* and column *j'* in layer *p*-1 and neuron in row *i* and column *j* in layer *p*

Pooling

- ❖After one or more full convolutional layers, CNNs typically applying pooling.
- ❖Pooling reduces the resolution of the output filtered image
- ❖Typically, this is by small amount (say, 2 x 2)
- ❖That is, two-by-two subgrids are considered and combined into one output, applying the subsampling with a certain pooling stride (Default pooling stride in Keras is equal to pooling kernel size, e.g., 2x2 max pooling with stride 2x2 divides both dimensions by half).
- ❖Typically, this is calculated using the max value of the inputs, but other choices are possible (e.g. weighted average)

Pooling

- The pooling layer performs a fixed computation, there are no parameters to be tuned, nothing to learn
- The pooling operation reduces the number of neurons in the next layer
- Pooling creates invariance to small shifts
- Pooling reduces the dimension of representations, possibly making it affordable to use more feature maps in subsequent layers



Flattening

- ❖Throughout the CNN so far, inputs and output have been tensors
- ❖With the use of convolutional filters, the number of weights to be learned is relatively low
- ❖The final layer(s) of a CNNs are those of a standard fully connected MLP
- ❖Therefore, after filtering and pooling, flattening turns the grid into a vector for input into part of the feed-forward network

Fully Connected Layers

- ❖A fully connected layer or layers is the final component of a CNN
- ❖Unlike the convolutional layers, even after pooling this layer will have quite a lot of weights to be learned

Labels and output, one-hot encoding

- ❖CNNs are trained on labelled data
- ❖This data typically will be categorical (e.g. a digit, or a name)
- ❖Therefore, the labels need to be encoded numerical
- ❖This is achieved using a one-hot encoding (as we have seen)
- ❖Therefore, the output layer applies a softmax activation function and gives a probability distribution for the possible categories of output

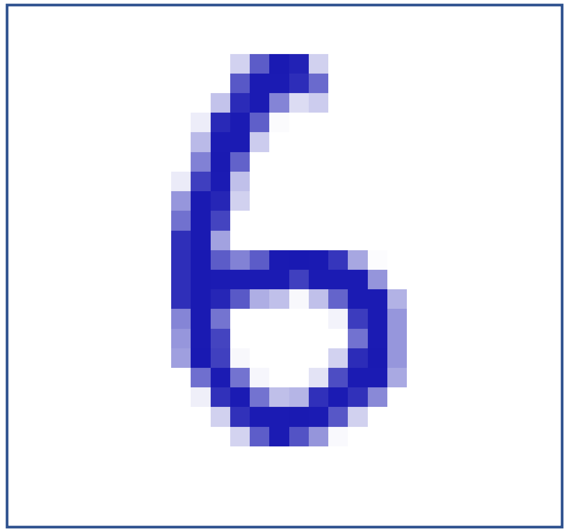
Data Augmentation

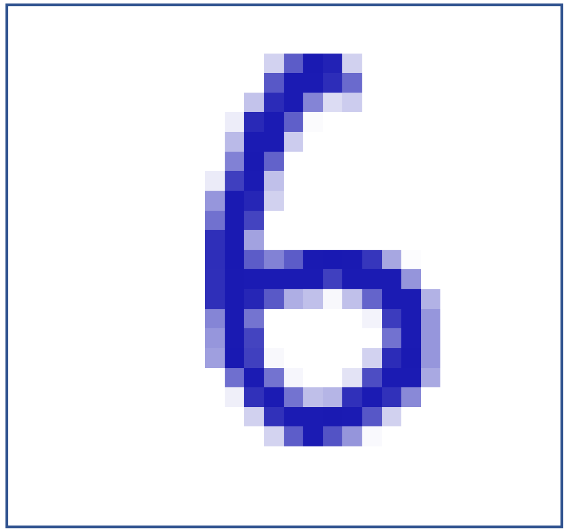
- Machine learning and pattern recognition algorithms work well when they have plenty of data to work with
- It might be that for image classification the available dataset is not quite as large as might be desired
- In addition, we want to be sure that small changes in data don’t change the way in which it is classified
- We don’t want to fail in classification because of a small rotation, or shift of position within an image, change of scale, or minor change of contrast, intensity, etc.

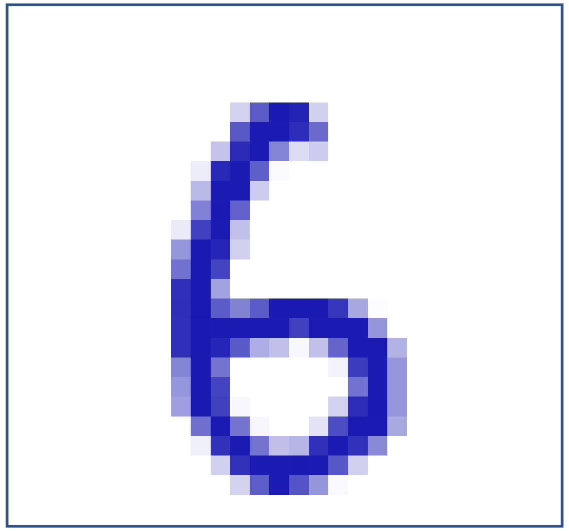
Data Augmentation

- Manipulating the input dataset in order to create additional artificial data can be a good way to improve the accuracy of a CNN classifier
- There are a number of way that this might be done:
  - Shift the image, up or down
  - Rotate the image (by how much depends on the problem and symmetries, e.g. ‘6’ and ‘9’, refrains from big rotations)
  - Flip the image (again problem dependent and existing symmetries in the image, e.g ‘b’ and ‘d’ refrain from flipping letters)
  - Scaling
  - More…

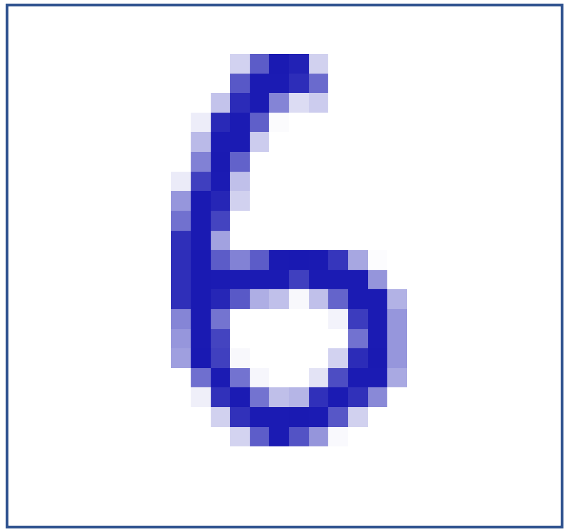
Data Augmentation

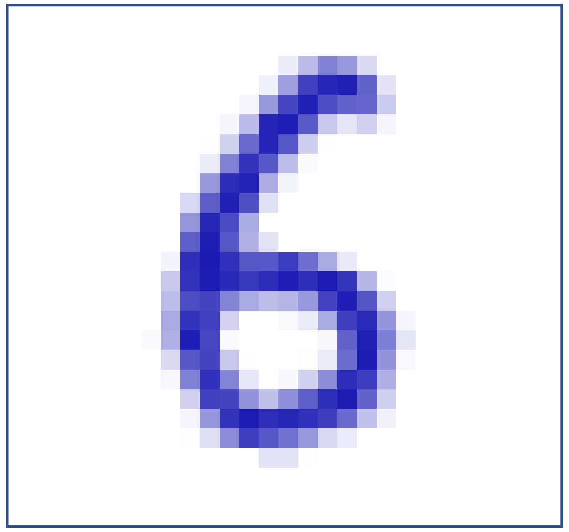

Original

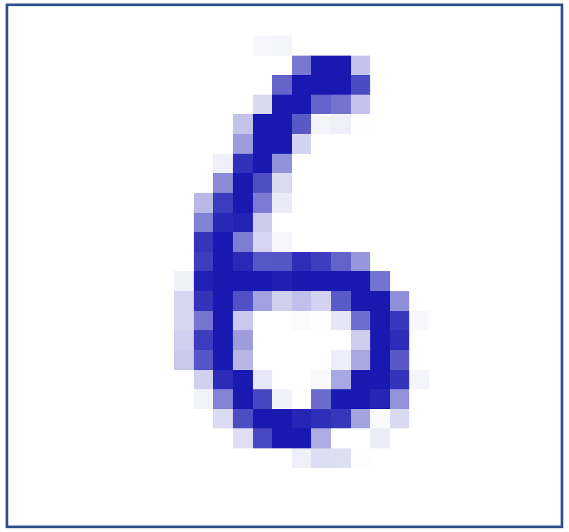

Moved right


Moved Down

Data Augmentation


Original


Rotated


Transformed
(small amount added noise)

Data Augmentation and Keras

- Generating new data from the input dataset is something that could be performed prior to training
- Keras has an ImageDataGenerator class which will generate new images using a randomised application of the parameters you set
- This can be incorporated into batch learning to improve the performance of a classification CNN

Design of CNNs

- In summary, you have a number of parameters to tune and choice to make when building a CNN:
  - The number of convolution layers
  - When to apply pooling
  - What activation functions to use (standardly ReLU)
  - How many fully connected layers to include
  - How many feature maps in each of the convolutional layers
  - What kernel size to use in each of the convolutional layers

Design of CNNs

- Continued…:
  - Whether to use padding
  - What stride to use
  - Which pooling method to use, and subgrid size
  - Whether to use data augmentation
  - How many nodes in the fully connected hidden layers

CNN code

```
model = Sequential()
model.add(Conv2D(64, kernel_size=(4, 4), activation='relu', strides=1,
                         padding='same', input_shape= X_train[0].shape))
model.add(Conv2D(32, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(64, (3, 3), activation='relu', padding='same'))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
```

CNN code

```
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dense(units = num_classes, activation = ‘softmax’))
model.compile(loss='categorical_crossentropy',optimizer='adam',
                    metrics=['accuracy'])
```

Summary

```
_________________________________________________________________
Layer (type)                        Output Shape           Param #
=================================================================
conv2d_1 (Conv2D)                   (None, 62, 47, 64)     1088     (4*4+1)64=1088
conv2d_2 (Conv2D)                   (None, 60, 45, 32)     18464    (3*3)64*32 + 1*32=18464
max_pooling2d_1 (MaxPooling 2D)     (None, 30, 22, 32)     0
conv2d_3 (Conv2D)                   (None, 30, 22, 64)     18496    (3*3)64*32 + 1*64=18496
conv2d_4 (Conv2D)                   (None, 28, 20, 64)     36928    (3*3)64*64 + 1*64=36928
max_pooling2d_2 (MaxPooling 2D)     (None, 14, 10, 64)     0
flatten_1 (Flatten)                 (None, 8960)           0
```

Imagenet

- Imagenet is a long running project to provide a database of images corresponding mostly to nouns (possibly across several layers of category: e.g. mammal, dog)
- It had been heavily used for machine learning/AI
- One particularly successful deep learning CNN architecture is VGG16
- This is partly because the model comes with Keras (and others ResNet, GoogLeNet, etc)

Example of existing CNN: VGG16

- VGG16 is built from 3d Convolutional Layers (since it is processing coloured images). Input is 256 x 256 x RGB. Trained 2-3 weeks on 4 GPUs…trained on Imagenet
  - Stride is one pixel, kernel size is 3 x 3, in all convolutional layers
  - 2 convolutional layers with 64 filters, followed by 2 x 2 max pooling
  - 2 convolutional layers with 128 filters, followed by 2 x 2 max pooling
  - 3 convolutional layers with 256 filters, followed by 2 x 2 max pooling
  - 3 convolutional layers with 512 filters, followed by 2 x 2 max pooling
  - 3 convolutional layers with 512 filters, followed by 2 x 2 max pooling
  - 2 x fully connected layers with 4096 nodes
  - Softmax over 1000 classes

Transfer learning

- Training good CNNs is a computationally expensive business
- Transfer learning is a general AI concept, where knowledge gained in one domain is reused in another related domain
- CNNs are good candidates for transfer learning
- The convolutional layers are learning filters, giving ways to capture features of the input images
- The fully connected layers are doing the classification
- To apply to a new problem, replace one or more fully connected layer with a fresh one, and training just this layer.

Summary

- ✓Image filtering
- ✓Images and neural networks
- ✓Convolution Neural Networks
- ✓Data Augmentation
- ✓Design and use of CNNs

Reading

- “Pattern Recognition and Machine Learning,” Bishop, Springer 2006. Chapter 5
- “Deep Learning,” Goodfellow, Bengio and Courville. MIT Press 2016, Chapter 9. http://www.deeplearningbook.org
- “Artificial Intelligence, 3rd ed.” Russell & Norvig, Pearson 2010. Chapter 24.
- “Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow”, Geron, A., O'Reilly Media, 2019. Chapter 14.