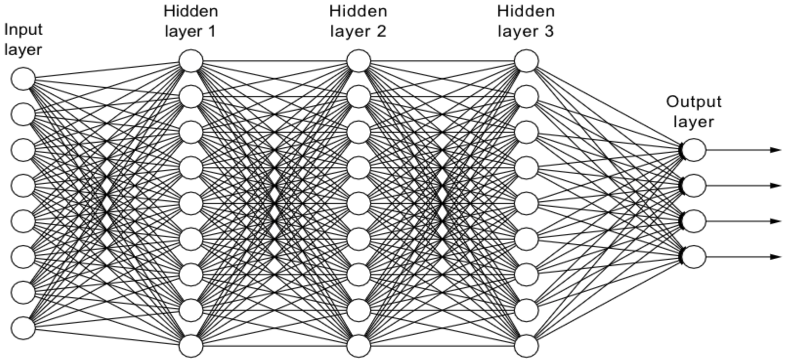
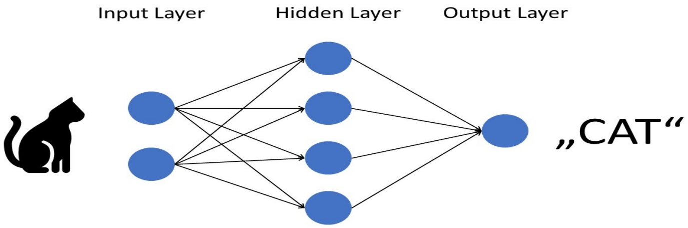
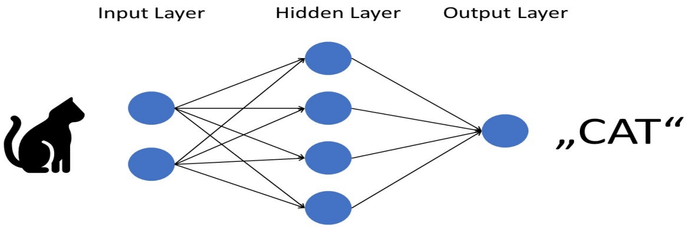
Machine Learning 5: Convolutional Neural Networks



Recap

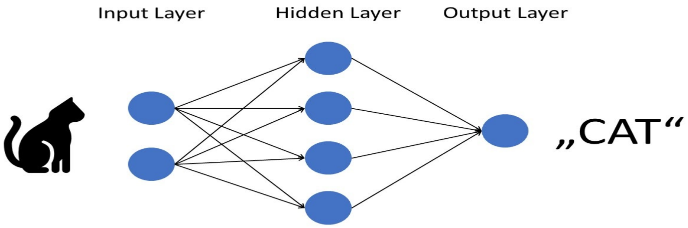


Recap



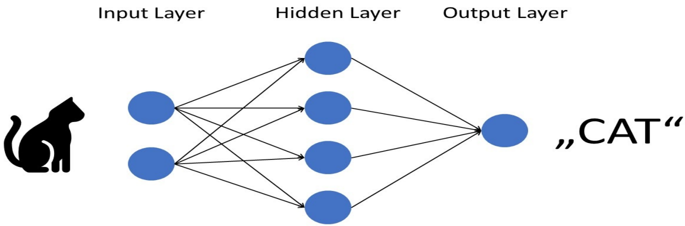
- A neural network is made up of a series of layers with artificially created ‘features’ or **activations**.

Recap



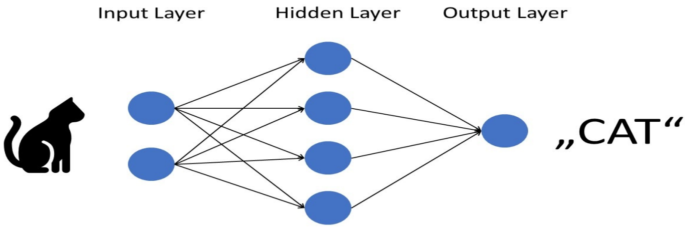
- A neural network is made up of a series of layers with artificially created ‘features’ or **activations**.
- These features are created through **non-linearly transforming** (e.g. sigmoid function) a bf weighted combination of the input features.

Recap



- A neural network is made up of a series of layers with artificially created ‘features’ or **activations**.
- These features are created through **non-linearly transforming** (e.g. sigmoid function) a bf weighted combination of the input features.
- The **training process** consists of a series of **forward propagations** (to determine the cost) and **back propagations** (to determine the cost derivatives) in order to optimise the weights.

Recap



- A neural network is made up of a series of layers with artificially created ‘features’ or **activations**.
- These features are created through **non-linearly transforming** (e.g. sigmoid function) a bf weighted combination of the input features.
- The **training process** consists of a series of **forward propagations** (to determine the cost) and **back propagations** (to determine the cost derivatives) in order to optimise the weights.
- Typically each hidden layer of a neural network is **fully connected**, i.e. each neuron in layer N is connected to each neuron in layer $N \pm 1$.

A short-coming of standard neural networks

In the context of image classification, a neural network with fully connected layers can end up over-fitting very easily.

A short-coming of standard neural networks

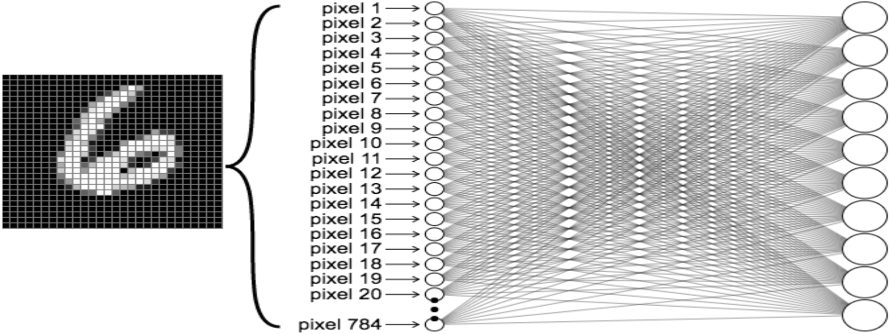
In the context of image classification, a neural network with fully connected layers can end up over-fitting very easily.

For example a 200×200 pixed color (RGB: $\times 3$) image produces $120,000$ weights for each neuron of our first hidden layer.

A short-coming of standard neural networks

In the context of image classification, a neural network with fully connected layers can end up over-fitting very easily.

For example a 200×200 pixed color (RGB: $\times 3$) image produces $120,000$ weights for each neuron of our first hidden layer.



Convolutions

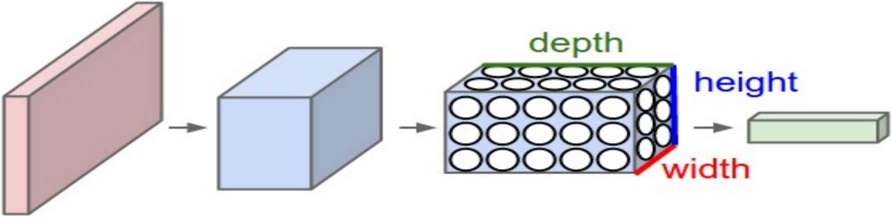
One way to solve this problem is to make use of the fact that our input is an image. This image can be represented as a volume, for example a $200 \times 200 \times 3$ cuboid.

Segments of this volume can then be convolved with weight arrays to produce a new volume. We can perform successive such convolutions to finally ending up with a reduced output volume, for example 1×2 for the face-not face classification.

Convolutions

One way to solve this problem is to make use of the fact that our input is an image. This image can be represented as a volume, for example a $200 \times 200 \times 3$ cuboid.

Segments of this volume can then be convolved with weight arrays to produce a new volume. We can perform successive such convolutions to finally ending up with a reduced output volume, for example 1×2 for the face-not face classification.



Credit: https://cs231n.github.io/convolutional-networks/

How does this work in practice?

How does this work in practice?

Each weight array, or **filter**, scans the image jumping N-pixels as a time (N=1 usually). N is called the stride.

How does this work in practice?

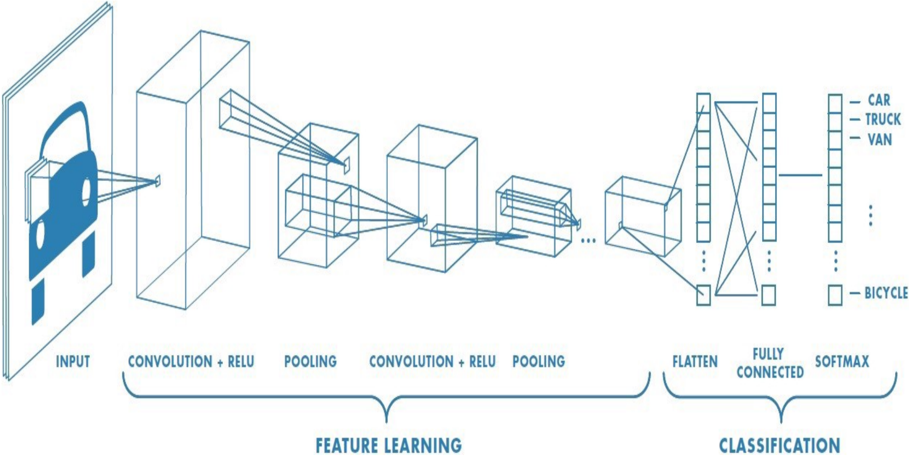
Each weight array, or **filter**, scans the image jumping N-pixels as a time (N=1 usually). N is called the stride.

When it is applied to a segment of the image an element-wise multiplication is performed and then the products are summed to give the output ‘pixel’.

See animation at at

https://cs231n.github.io/convolutional-networks/

The layers of a CNN



Credit: https://medium.com/@_sumitsaha_

The layers of a CNN

Conv layer: The filters to be applied to the input volume with a depth equal to the input volume depth (in our example 3 - RGB). These filters are applied to segments of the input volume sequentially, producing a new volume.

The layers of a CNN

Conv layer: The filters to be applied to the input volume with a depth equal to the input volume depth (in our example 3 - RGB). These filters are applied to segments of the input volume sequentially, producing a new volume.

Pooling layer: A pooling layer acts to reduce the volume and hence number of weights for a successive Conv layer. It does this by moving through the input volume at a stride greater than 1 pixel at a time. One example is max pooling which applies a `Max` function to select most important input features.

Flatten and Dense layer: Unrolls the volume into a vector of values and then the dense layer is a fully connected normal neural network hidden layer with neurons equal to the number of classes.

Flatten and Dense layer: Unrolls the volume into a vector of values and then the dense layer is a fully connected normal neural network hidden layer with neurons equal to the number of classes.

Softmax: Apply softmax function to generate probabilities for classes.

$$\sigma_i(\boldsymbol{z}) = \frac{e^{z_i}}{\sum_{j=1}^{j=K} e^{z_j}}. \tag{1}$$

Flatten and Dense layer: Unrolls the volume into a vector of values and then the dense layer is a fully connected normal neural network hidden layer with neurons equal to the number of classes.

Softmax: Apply softmax function to generate probabilities for classes.

$$\sigma_i(\boldsymbol{z}) = \frac{e^{z_i}}{\sum_{j=1}^{j=K} e^{z_j}}. \tag{1}$$

Dropout: Dropout is a technique which acts as a regularisation of the network. During training, random activations are not considered in the layer prior to a dropout operation. This creates 'noise' in the training process. By forcing the weights to account for this, we make each individual weight less important, and consequently we reduce over-fitting.

Flatten and Dense layer: Unrolls the volume into a vector of values and then the dense layer is a fully connected normal neural network hidden layer with neurons equal to the number of classes.

Softmax: Apply softmax function to generate probabilities for classes.

$$\sigma_i(\boldsymbol{z}) = \frac{e^{z_i}}{\sum_{j=1}^{j=K} e^{z_j}}. \tag{1}$$

Dropout: Dropout is a technique which acts as a regularisation of the network. During training, random activations are not considered in the layer prior to a dropout operation. This creates 'noise' in the training process. By forcing the weights to account for this, we make each individual weight less important, and consequently we reduce over-fitting.

You can now have a look at 'tutorials/NN_tutorial.ipynb' and BaCoN!