# Topic 4
# Deep Learning for Image Recognition

AI HUMAN INTERFACE

## Learning Outcomes

- Convolutional Neural Networks (CNN)
- Using Deep Learning for Images

## Convolutional Neural Networks (CNN)

## Convolutional Neural Networks (CNNs / ConvNets)

- Convolutional Neural Networks are very similar to ordinary Neural Networks: they are made up of neurons that have learnable weights and biases. Each neuron receives some inputs, performs a dot product and optionally follows it with a non-linearity.
- The whole network still expresses a single differentiable score function: from the raw image pixels on one end to class scores at the other. And they still have a loss function (e.g. SVM/Softmax) on the last (fully-connected) layer and all the tips/tricks we developed for learning regular Neural Networks still apply.
- So what does change? ConvNet architectures make the explicit assumption that the inputs are images, which allows us to encode certain properties into the architecture. These then make the forward function more efficient to implement and vastly reduce the amount of parameters in the network.

source http://cs231n.github.io/convolutional-networks/

## CNN: Architecture Overview

- Regular Neural Nets don’t scale well to full images.
  - In CIFAR-10, images are only of size 32x32x3 (32 wide, 32 high, 3 colour channels), so a single fully-connected neuron in a first hidden layer of a regular Neural Network would have 32*32*3 = 3072 weights. This amount still seems manageable, but clearly this fully-connected structure does not scale to larger images.
  - For example, an image of more respectable size, e.g. 200x200x3, would lead to neurons that have 200*200*3 = 120,000 weights.
  - Moreover, we would almost certainly want to have several such neurons, so the parameters would add up quickly! Clearly, this full connectivity is wasteful and the huge number of parameters would quickly lead to overfitting.

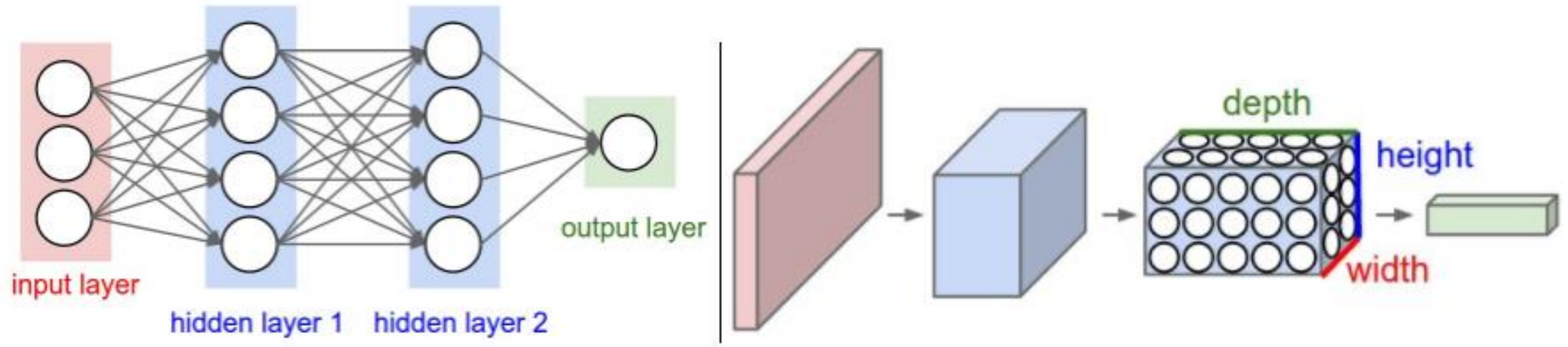
## CNN: Architecture Overview

- Regular Neural Nets don’t scale well to full images.
  - In CIFAR-10, images are only of size 32x32x3 (32 wide, 32 high, 3 colour channels), so a single fully-connected neuron in a first hidden layer of a regular Neural Network would have 32*32*3 = 3072 weights. This amount still seems manageable, but clearly this fully-connected structure does not scale to larger images.
  - For example, an image of more respectable size, e.g. 200x200x3, would lead to neurons that have 200*200*3 = 120,000 weights.
  - Moreover, we would almost certainly want to have several such neurons, so the parameters would add up quickly! Clearly, this full connectivity is wasteful and the huge number of parameters would quickly lead to overfitting.

## CNN: Architecture Overview

- 3D volumes of neurons. Convolutional Neural Networks take advantage of the fact that the input consists of images and they constrain the architecture in a more sensible way. In particular, unlike a regular Neural Network, the layers of a ConvNet have neurons arranged in 3 dimensions: width, height, depth (e.g. w x h x colours).
- As we will soon see, the neurons in a layer will only be connected to a small region of the layer before it, instead of all of the neurons in a fully-connected manner. Moreover, the final output layer would for CIFAR-10 have dimensions 1x1x10, because by the end of the ConvNet architecture we will reduce the full image into a single vector of class scores, arranged along the depth dimension.

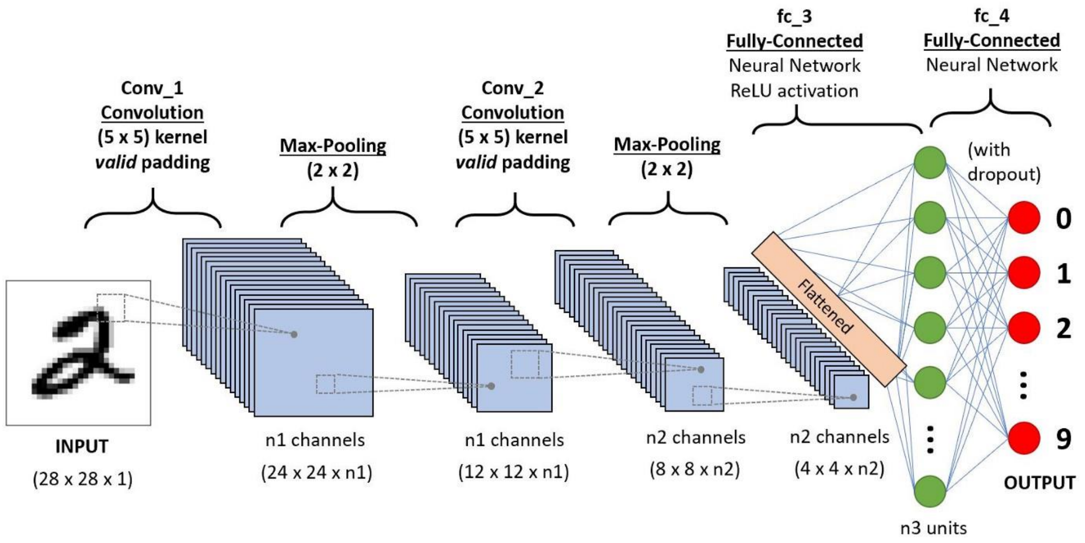
# CNN: Architecture Overview



Left: A regular 3-layer Neural Network. Right: A ConvNet arranges its neurons in three dimensions (width, height, depth), as visualized in one of the layers. Every layer of a ConvNet transforms the 3D input volume to a 3D output volume of neuron activations. In this example, the red input layer holds the image, so its width and height would be the dimensions of the image, and the depth would be 3 (Red, Green, Blue channels).

# CNN: Architecture Overview

- A Convolutional Neural Network (ConvNet) can **assign learnable weights and biases** to various objects in the image.
- ConvNet can capture **spatial and temporal dependencies** in an image through the application of relevant filters.
- ConvNet contains: Input, Convolutional Layers, Pooling Layers, Fully-Connected Layers.

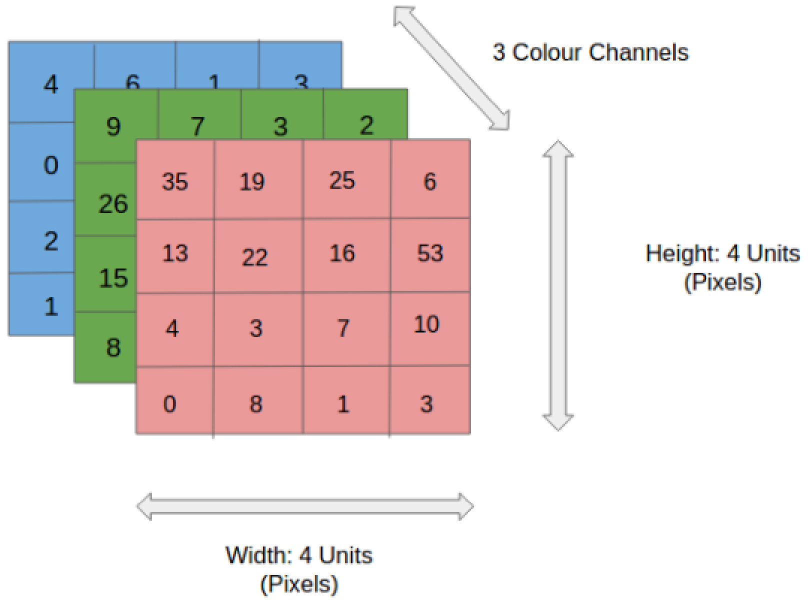


## CNN: Layers used to build ConvNets

- A simple ConvNet is a sequence of layers, and every layer of a ConvNet transforms one volume of activations to another through a differentiable function.
- We use three main types of layers to build ConvNet architectures:
  - Input Image
  - Convolutional Layer
  - Pooling Layer
  - Fully-Connected Layer (exactly as seen in regular Neural Networks)

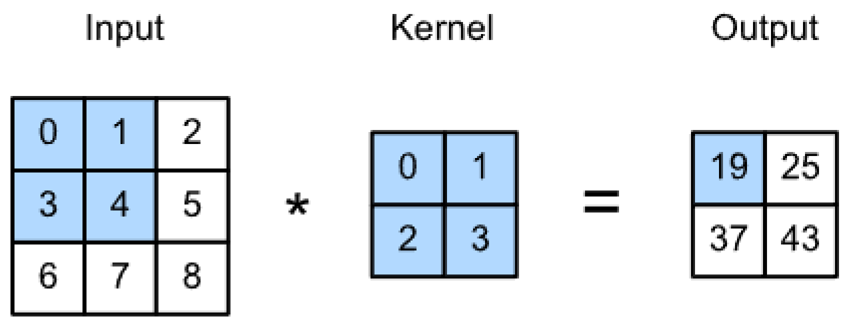
# Input Image

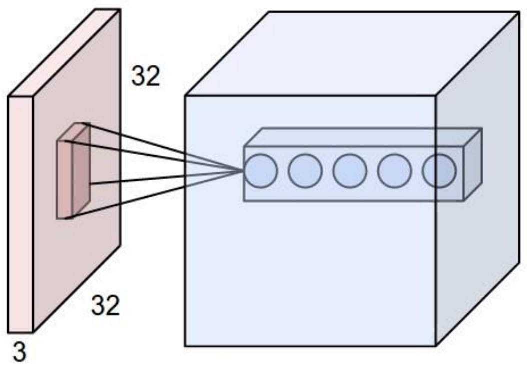
- An image can be separated into different color channels: grayscale, RGB, HSV, CMYK, etc.
- It will become computationally-intensive once the images reach higher dimensions, i.e. 8K images.
- ConvNet can reduce image dimensions without losing features that are critical for good predictions.



## Convolutional Layer

- Convolutional layers usually have filter (kernel) size of 3*3 or 5*5.
- The objective of the convolution operation is to extract **the high-level features** such as edges, from the input image.
- The number of parameters in a convolutional layer equals to width x height x channel, which is much less than fully-connected layers.

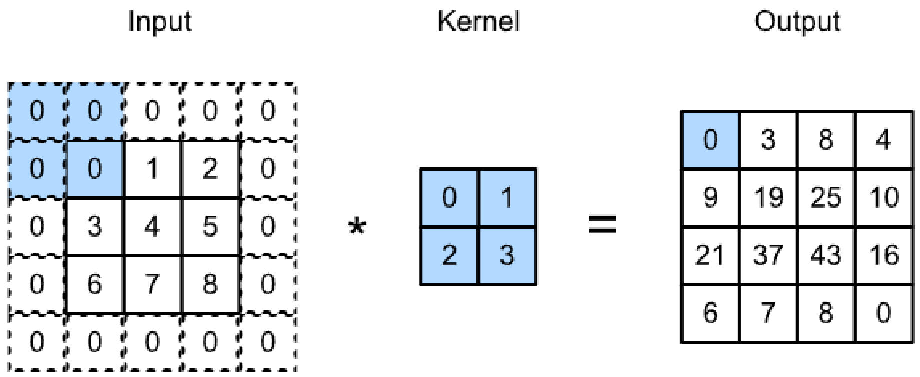




# Padding in Convolutional Layer

- Convolution operation can shrink the original image size, and it will lose information in the **corner pixels**.
- This issue can accumulate when applying many successive convolutional layers.
- **Solution**: zero padding can preserve the size of the original image and allow for a more accurate analysis of images.

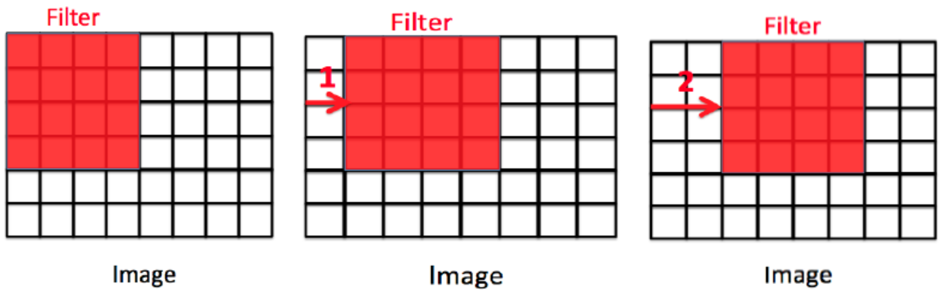
**Padding image convolved with 2*2 kernel**



# Stride in Convolutional Layer

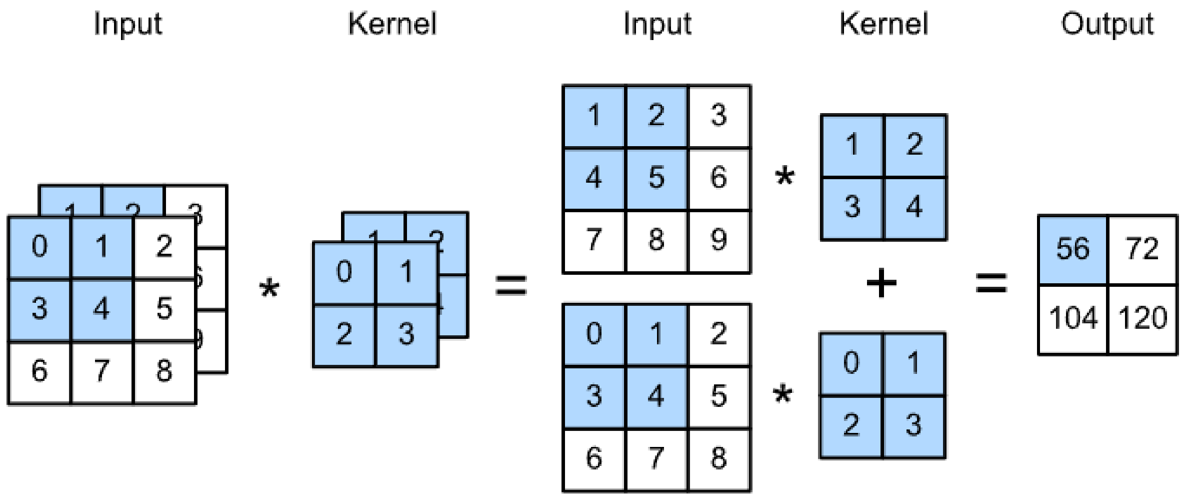
- Stride controls how the filter convolves through the input layer, and stride = 1 by default.
- A larger stride is used to increase computational efficiency or **downsample** the feature size.
- Stride and padding can be used together to adjust the dimensionality of the images.

**Stride with 1 step and 2 steps**



# Multiple Input and Output Channels

- During convolution operations, the channels of each kernel / filter must equal to the channels of input layer.
- In most neural network architectures, we can increase the number of filters as we go higher up in the network, so that our network is able to extract more abstractions from the images.



## CNN: Convolutional Layer

The CONV layer’s parameters consist of a set of learnable filters. Every filter is small spatially (along width and height), but extends through the full depth of the input volume. For example, a typical filter on a first layer of a ConvNet might have size 5x5x3 (i.e. 5 pixels width and height, and 3 because images have depth 3, the colour channels).

During the forward pass, we slide (more precisely, convolve) each filter across the width and height of the input volume and compute dot products between the entries of the filter and the input at any position. As we slide the filter over the width and height of the input volume we will produce a 2-dimensional activation map that gives the responses of that filter at every spatial position.

## CNN: Convolutional Layer

Intuitively, the network will learn filters that activate when they see some type of visual feature such as an edge of some orientation or a blotch of some colour on the first layer, or eventually entire honeycomb or wheel-like patterns on higher layers of the network. Now, we will have an entire set of filters in each CONV layer (e.g. 12 filters), and each of them will produce a separate 2-dimensional activation map. We will stack these activation maps along the depth dimension and produce the output volume.

# CNN: Convolutional Layer

**Local Connectivity**. When dealing with high-dimensional inputs such as images, as we saw above it is impractical to connect neurons to all neurons in the previous volume. Instead, we will connect each neuron to only a local region of the input volume.

The spatial extent of this connectivity is a hyperparameter called the **receptive field** of the neuron (equivalently this is the filter size).

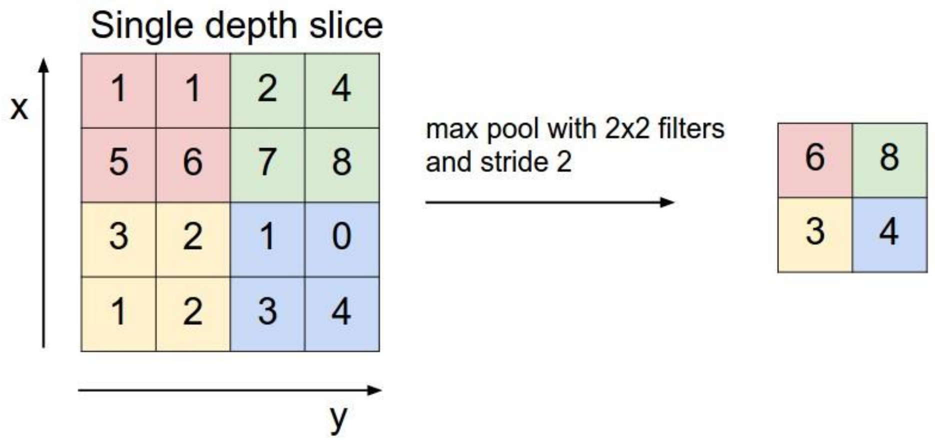
The extent of the connectivity along the depth axis is always equal to the depth of the input volume. It is important to emphasize again this asymmetry in how we treat the spatial dimensions (width and height) and the depth dimension: The connections are local in space (along width and height), but always full along the entire depth of the input volume.

# CNN: ReLU Layer

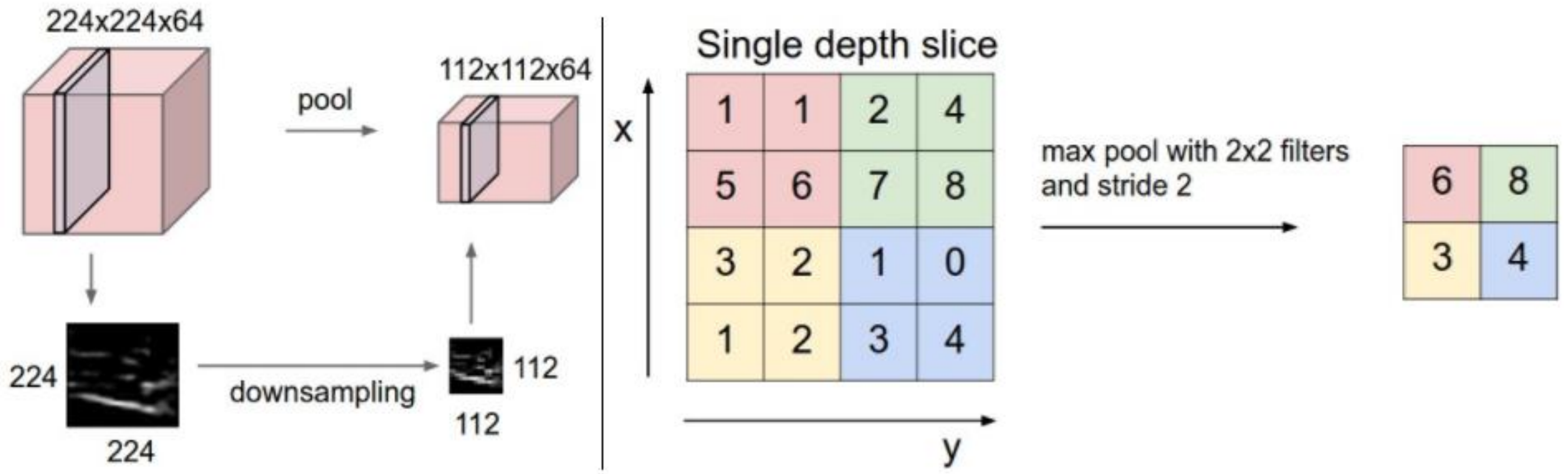
- ReLU Layer is not really an independent neuron layer like the CONV and POOL layers.
- We can explicitly write the RELU activation function as a layer, which applies elementwise non-linearity.
- The most common form of a ConvNet architecture stacks a few CONV-RELU layers, follows them with POOL layers, and repeats this pattern until the image has been merged spatially to a small size. At some point, it is common to transition to fully-connected layers.

## Pooling Layer

- Pooling layers are used to **reduce the dimensions** of the feature maps. Thus, it reduces the number of parameters to learn and the amount of computation performed in the network.
- The pooling layer summarises the features present in a region of the feature map generated by a convolution layer. So, further operations are performed on summarised features instead of precisely positioned features generated by the convolution layer. This makes the model more **robust to variations in the position** of the features in the input image.



# CNN: Pooling Layer



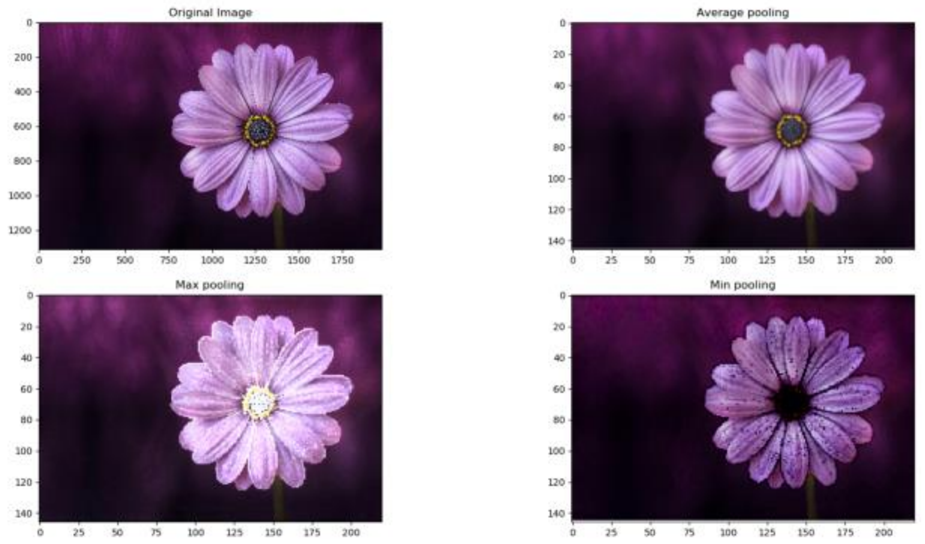
Pooling layer downsamples the volume spatially, independently in each depth slice of the input volume. **Left:** In this example, the input volume of size [224x224x64] is pooled with filter size 2, stride 2 into output volume of size [112x112x64]. Notice that the volume depth is preserved. **Right:** The most common downsampling operation is max, giving rise to **max pooling**, here shown with a stride of 2. That is, each max is taken over 4 numbers (little 2x2 square).

## Why We Like Max Pooling

Max pooling: The maximum pixel value of the batch is selected.

Min pooling: The minimum pixel value of the batch is selected.

Average pooling: The average value of all the pixels in the batch is selected.

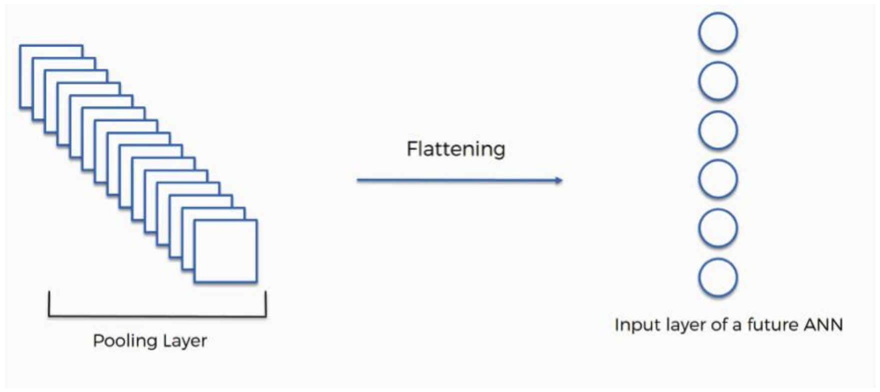


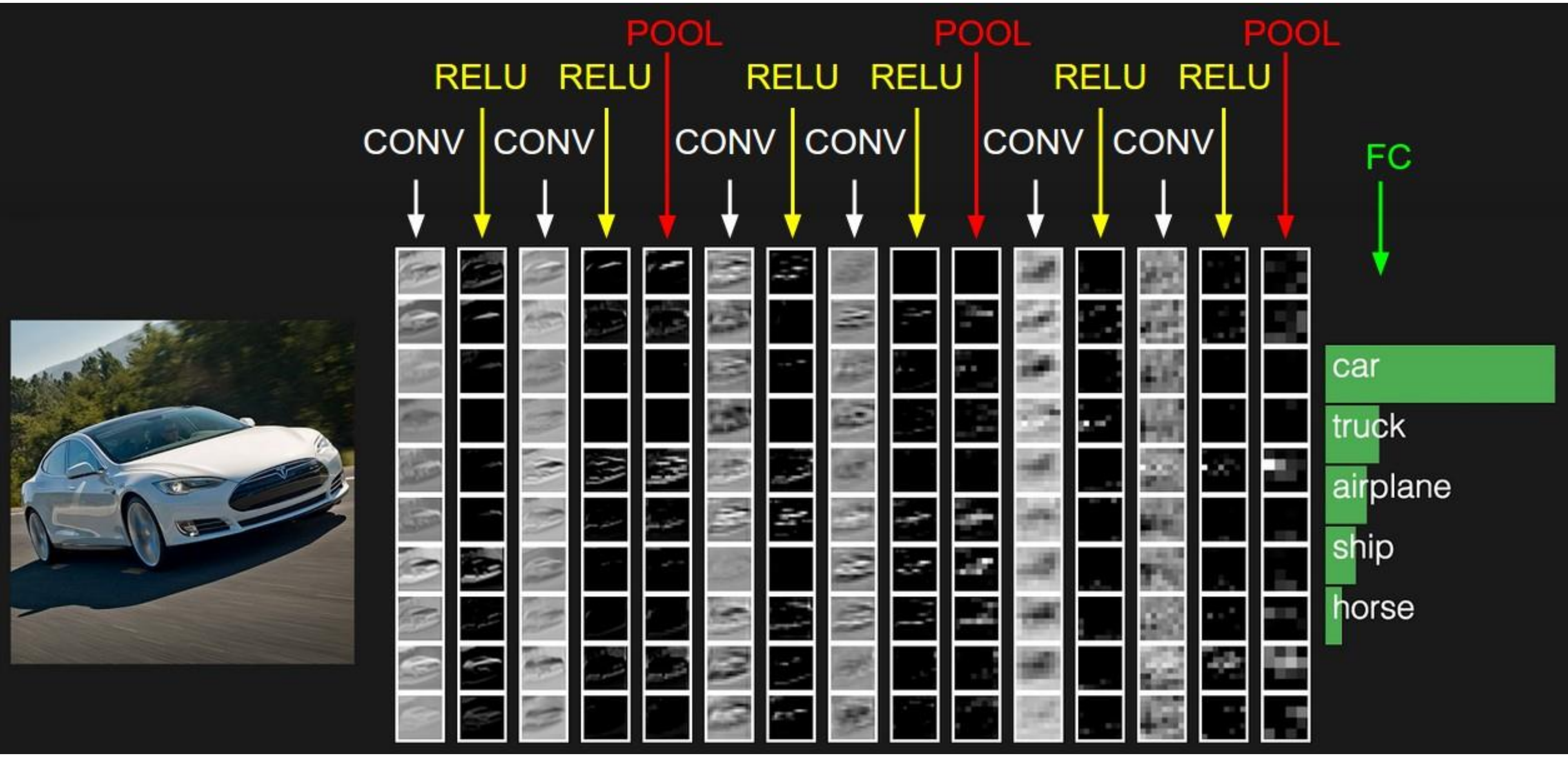
## Fully-Connected Layer

In a modern CNN architecture, fully-connected layers form the last few layers in the network.

The output from the last convolutional / pooling layer will be **flattened** and fed into the fully-connected layer.

The final layer will use **sigmoid / softmax** activation function to calculate the probabilities of different classes.




POOL
POOL
POOL
RELU
RELU
RELU
RELU
RELU
RELU
CONV
CONV
CONV
CONV
CONV
CONV
FC
car
truck
airplane
ship
horse

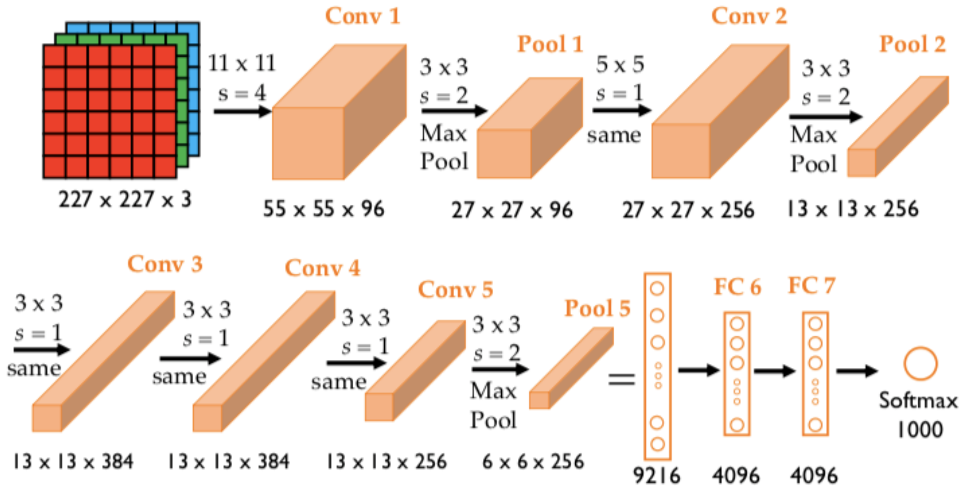
## Modern CNN Architectures

# Modern CNN Architectures: AlexNet

AlexNet architecture marked the first success of applying convolutional neural network on image recognition task.

AlexNet used **large filter size** (11 × 11) in its first convolutional layer and reduced the filter size in the subsequent convolutional layers.

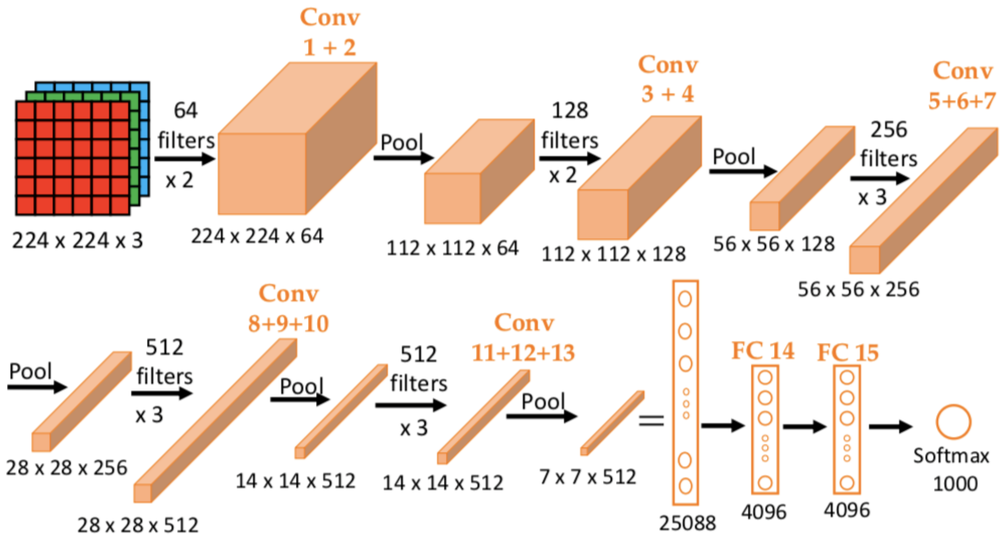
AlexNet used ReLU activation function to simplify the computation and become more robust to different parameter initialization.



## Modern CNN Architectures: VGG

VGG are very deep convolutional network models with 16 layers or 19 layers, with approximately 138 million parameters.

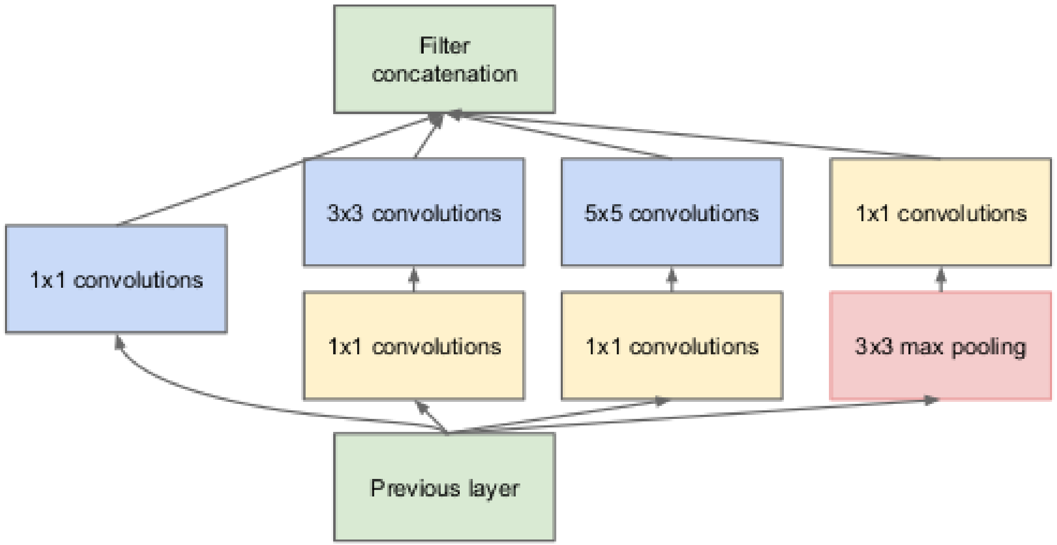
VGG architecture is constructed using various blocks, and each block contains a convolutional layer, ReLU activation, and a max pooling layer. The use of blocks allows for efficient design of complex networks.



# Modern CNN Architectures: GoogleNet

GoogleNet employs Inception blocks, which is a **combination of kernels** with different sizes.

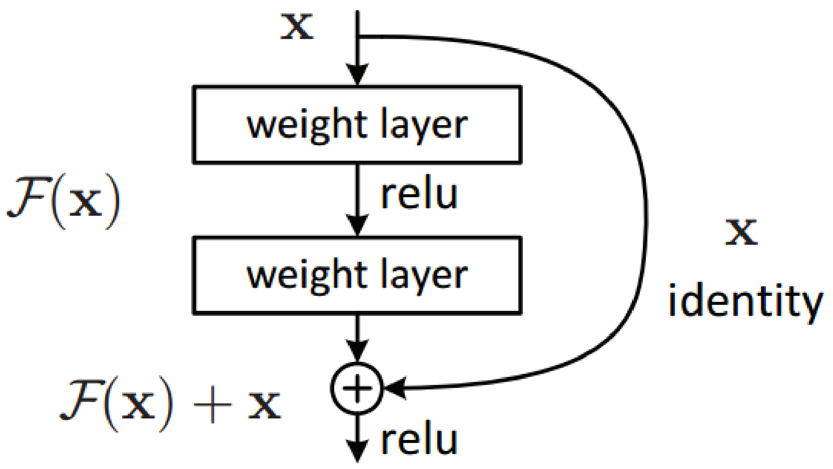
The inception architecture can explore the image in a variety of tiler sizes. This means that details at different extents can be recognized efficiently by filters of different sizes.

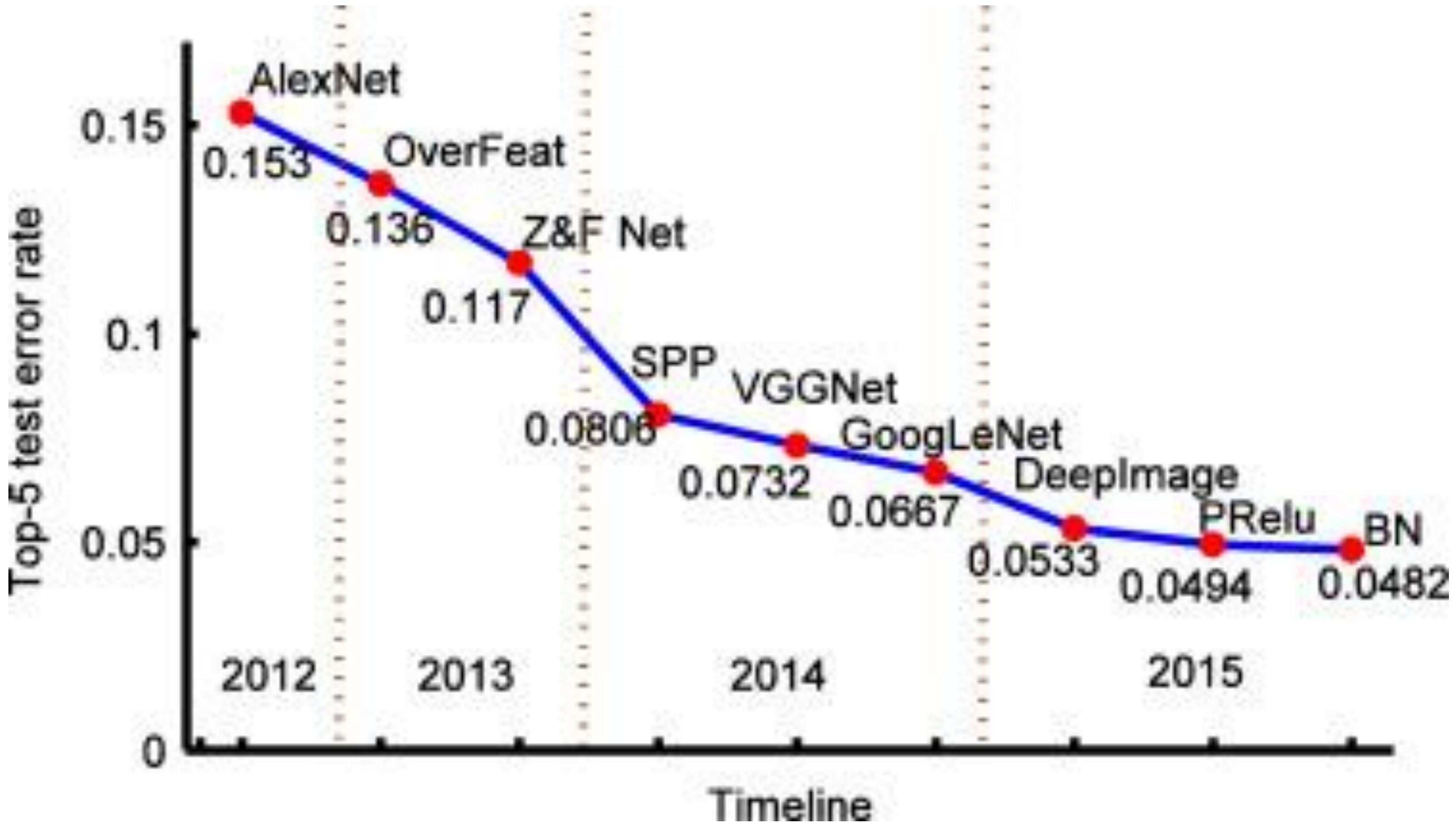


# Modern CNN Architectures: ResNet

Vanishing gradient is a common problem, which leads to failure of very deep convolutional neural networks.

In ResNet architecture, the identify mapping solves the vanishing gradient problem by **providing alternate path** for the gradient to flow through.





Top-5 test error rate
0.15
0.1
0.05
0
AlexNet
0.153
OverFeat
0.136
Z&F Net
0.117
SPP
0.0806
VGGNet
0.0732
GoogLeNet
0.0667
DeepImage
0.0533
PRelu
0.0494
BN
0.0482
2012
2013
2014
2015
Timeline

## Techniques to Improve Deep Learning

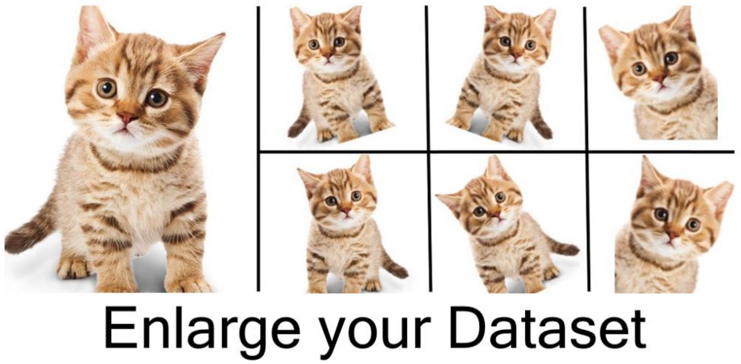
# Improve Performance with More Data

Get more data

- Deep learning model performance is usually proportional to amount to training data.

Invent more data -> **Data Augmentation**

- Rotation, cropping, shifting, adjusting brightness and exposure.



Semi-supervised learning

- Supplement training data with non-labelled data.

## Improve Performance with Larger Model

Add more layers and nodes to the deep neural network -> more complex model.

Reduce filter sizes -> **smaller filter** (3x3 or 5x5) usually perform better than large filters.

The maximum number of hidden nodes between fully-connected layers is restricted by the memory size (either GPU or RAM).

For very deep neural network, **skip connection** design is required to solve vanishing gradient problems (ResNet structure).

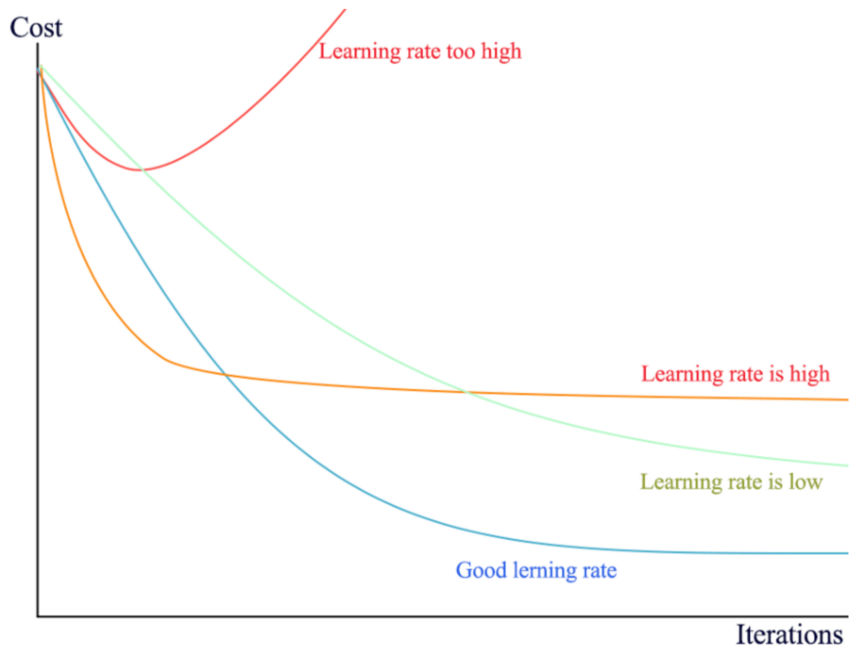
# Improve Performance with Tuning – Learning Rate

The first model hyperparameter to tune is the learning rate.

Typical learning rate is between 1 to 1e-7.

A high learning rate can cause loss to go up, and low learning rate will take a long time to converge.

Learning Rate Tuning



# Improve Performance with Tuning – Batch Size

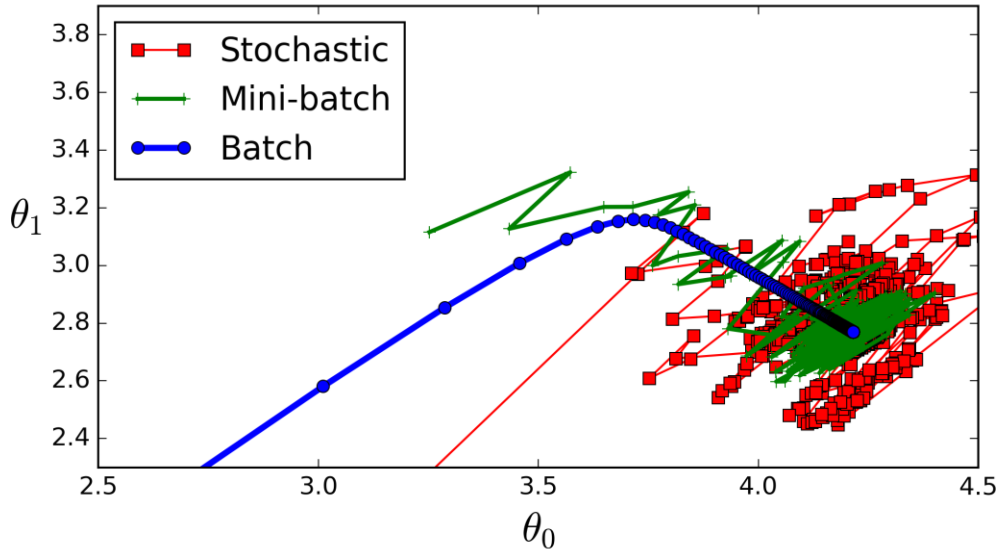
**batch size** defines the number of samples that will be propagated through the network.

Typical batch size is 8, 16, 32 or 64.

Too small batch size -> model learn slowly and it may oscillate

Too large batch size -> One iteration will be very long with small update.

**The direction of small batch gradient fluctuates more in comparison to full batch gradient**



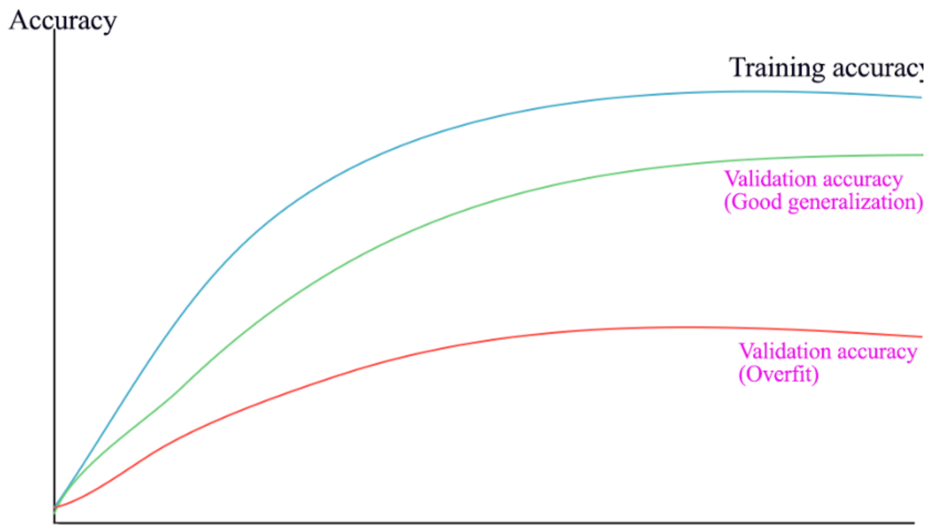
## Improve Performance with Tuning – Regularization

Regularization can help the neural network model avoid overfitting and generalize well on unseen data.

**Dropout**: skip random neurons during training.

**Learning rate decay**: reduce learning rate after certain iterations.

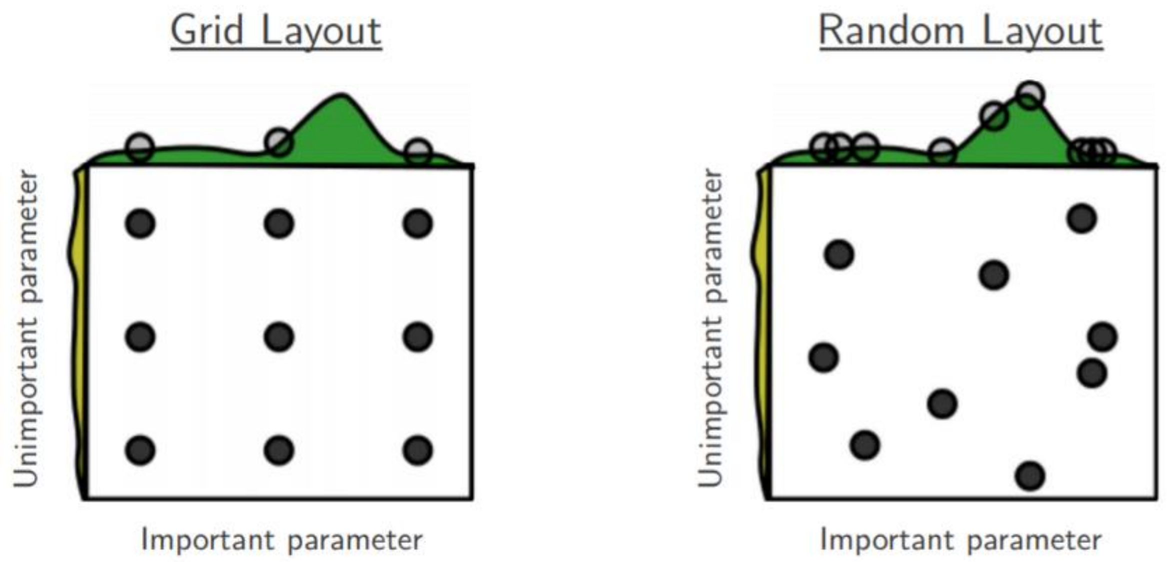
**Early stopping**: overfitting can be reduced by stopping the training when the validation errors increase persistently.



# Improve Performance with Tuning – Grid Search

Grid search is used to find the **optimal combination** of hyperparameters in a model which results in the most 'accurate' predictions.

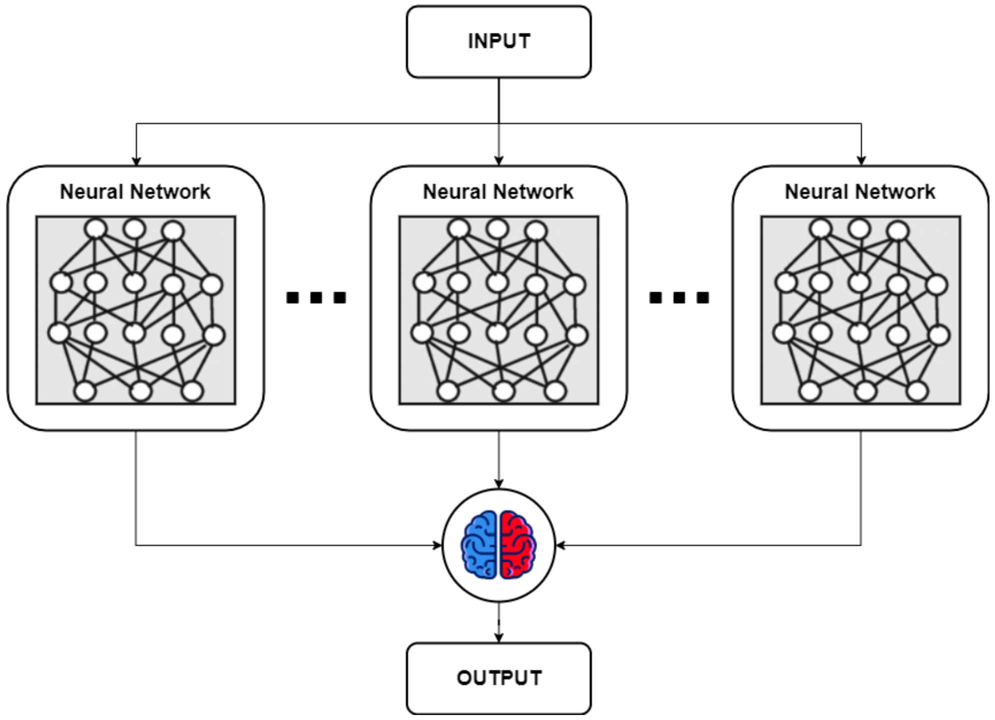
A grid search of hyperparameters is computationally intensive. We should start from coarse grain to fine tuning.



# Improve Performance with Ensemble Model

Ensemble Model: combines the predictions from multiple neural network models to reduce the variance of predictions and reduce generalization error.

The shortcomings of ensemble neural network is that 1) Increased complexity as there is a need to construct and combine more than one base model, and 2) Loss in interpretability of the model.



## Summary

We have learnt:

- What is a convolutional neural network (CNN) architecture
- How CNN can be used for image classification task
- Techniques to improve Deep Learning