Intro to Neural Network

- Logistic regression was invented in the early 1700s, XG boosting and random forest are from early 2000s
- It took two major technological advances for this to be relevant since 2008
  - 1. Science had to advance a lot to create statistical model that change the way we thought about this model that had been not very useful.
  - 2. They created revolutionary ways to train models using real life data
  - These created deep learning

Machine Learning and Artificial Intelligence

- AI
  - Is the effort to automate intellectual tasks normally performed by humans
    - It is to replace human in terms of reasoning at some level
- Machine learning is only a piece of AI
  - AI includes philosophy, neurology, and a bunch of science
  - Machine learning is the states and the science part of it
- Machine learning is all about turning computing on its head
  - When computers first used in early 1900s, the whole point of computing is a sum of things in which you give it data and rules and it give you answers
  - Machine learning is turning this around, the question now is not getting an answer, but to find rules that generate consistency across a set of situations
    - So taking data and answers, and turn that into rules
    - Random forest might have millions of cuts, a logistic regression gives you parameters to multiply by the value of variables.

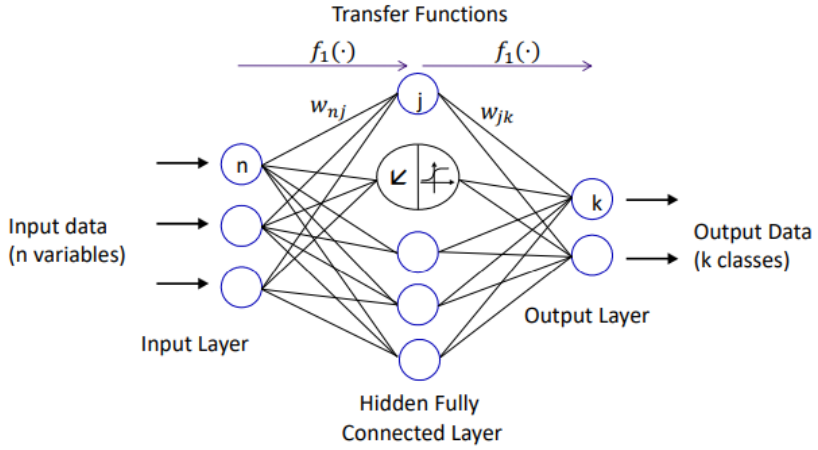
- Two key theorems
  - Universal approximation theorem
    - Under some very general conditions, a neural network can approximate any function with any level of tolerance that you want, just need to increase the complexity of the neural network
    - There are only a few models can achieve that
      - Fourier transformation is one, they need to approximate over a complex space for them to work
  - But approximating a function that is shown in data, but doesn't really translate to your ability to applying over a test set, it doesn't allow for generalization.
    - Babnik and Turban proved that if you have a test and train sample that comes from the same population, and you train a statistical model using functions that follow certain generalized properties and rules, those functions trained over data will be able to predict on test set

- For tree based methods and neural network, are not engineering model itself, we engineer methods that engineering models

Neural Network

- What is it?
  - ➢ It is a massively parallel distributed processor that has a natural propensity for storing experiential knowledge and making it available for use
  - ➢ It resembles the brain in two aspects
    - ✧ Knowledge is acquired by the network through a learning process
    - ✧ Inter-neuron connection strengths known as synaptic weights are used to store the knowledge
  - ➢ But it is a horrible way to simulate the brain
    - ✧ Simulating a worm with 10 neurons already takes a supercomputer
    - ✧ Besides, we are still finding new functions in the brain. We know that there is learning information and there is feedback

- Basic shape of neural network



  - ➢ You start with an input layer, it will have each of the variables entering each neuron
    - ✧ We would not normally use structured data for this
  - ➢ Then we are going to use a transfer function that takes the input and transfers them to the next neuron (i.e. to the hidden layer)
    - ✧ Here is where we encode the learning
    - ✧ We are going to weight the inputs (e.g. $x_i, \forall i$) by the weight and sum them (e.g. $y_1 = \sum w_{1,1}x_1 + \cdots w_{p,1}x_p$), and apply the transformation using the transfer function (e.g. $f_1(y_1) = \frac{1}{1+e^{-y_1}}$).
    - ✧ Then the first neuron of the hidden layer (i.e. j) will be $f_1(y_1)$
    - ✧ For the first neuron of the output layer (i.e. k), it will take the weighted sum of all the neurons of the hidden layer (e.g. $z_1 = \sum w_{2,1}f_1(y_1) + \cdots w_{2,p}f_1(y_p)$), then apply the second

transformation function, so we have the following at k (assume $f_2(z_1) = \frac{1}{1+e^{-z_1}}$)

$$f_2(z_1) = \frac{1}{1+e^{-z_1}} = \frac{1}{1+e^{-\sum w_{2,1} f_1(y_1) + \cdots w_{2,p} f_1(y_p)}} = \frac{1}{1+e^{-\sum w_{2,1} * \left(\frac{1}{1+e^{-y_1}}\right) + \cdots w_{2,p}\left(\frac{1}{1+e^{-y_p}}\right)}}$$

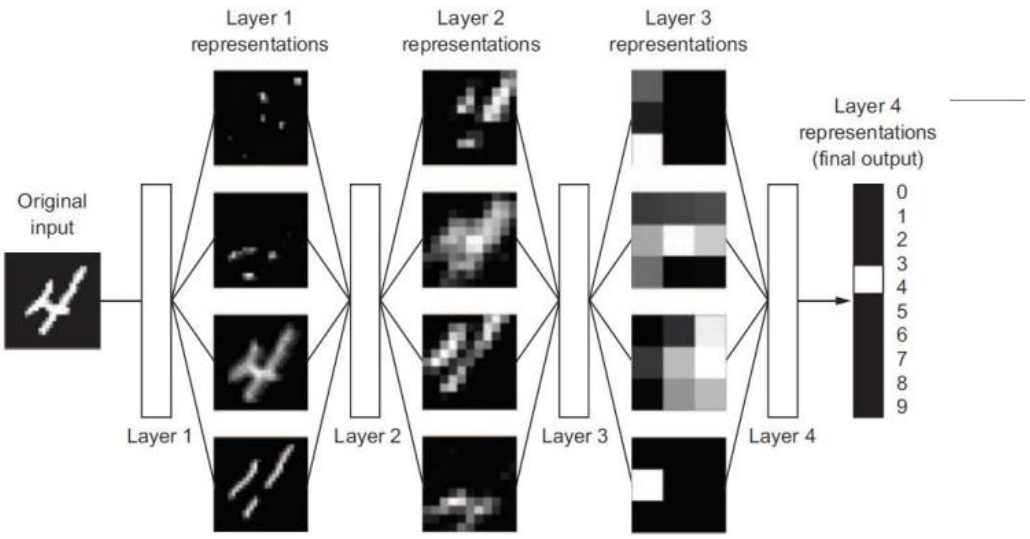
  - ✧ Lastly, there would be an output function which reshape the z_is into outputs with another set of weights (i.e. pretending k is not the output layer, this is another hidden layer before the output layer)
- ➢ This process is called feed forward
- ➢ The shape of this is going to be standard in terms of what functions to use
  - ✧ We're going to use certain logic arguments and identify which functions to go where
- ➢ How many layers to use how many inputs to use, and what transfer function and output function to use is our decision.
  - ✧ We are designing an architecture
- ➢ In the diagram, how many parameters we need to optimize?
  - ✧ Note that we add bias to hidden layers
  - ✧ So we have 3 * 10 * 2 + 10 + 2 = 72

- Two conditions for the input layer
  - ➢ It has to be small
  - ➢ It has to be normalized to the same range
- Neural networks could end up diverging for unknow reasons
  - ➢ It's optimization over an extremely complex space
  - ➢ It doesn't satisfy any of the modern rules of a well-designed problem
  - ➢ It will diverge, correlate, collapse (and all the weights will be 0)
    - ✧ When they collapse, can only retrain it again and hope it won't happen
    - ✧ Many papers try to see why it happens, but still unknown
    - ✧ To minimizing the chance of that happening, the inputs should be small
- A hidden layer doesn't really have a defined shape
  - ➢ We decide how many neurons there are in each layer, it can be more than the input, less than the input.
- If your data is structured, then you only need a neural network with two hidden layers to approximate any function
  - ➢ If the output function is bounded, then you need one hidden layer
  - ➢ But neural network is not strong for tabular data
    - ✧ Random forest and XG boosting are strong for that, and they are a lot simpler to train and they perform better
  - ➢ Neural network is strong for unstructured data, sound, images, text, and video

- Who is best for the job?
  - ➢ If you are doing predictive modelling, if you want to create something transparent, simple that can be easily interpreted, you go with logistic regression
  - ➢ If you have tabular data and you only care about the raw predive power, you go with random forest and XG Boosting
  - ➢ If you have unstructured data such as images, sounds, texts, then you go neural network

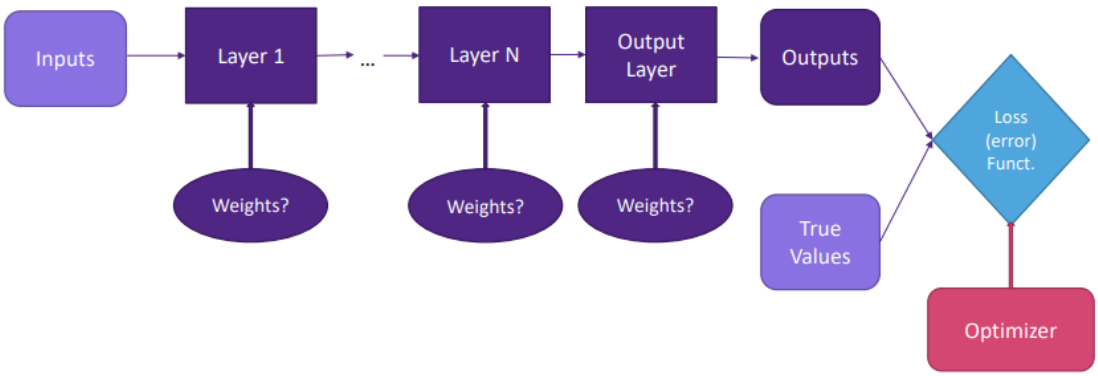
- The role of the output layer is to give the neural network output the shape it needs
  - If predicting price, it should be a function that's positive
  - If predicting probability, it should be a SoftMax or logistic
  - If performing unbounded regression, it should be linear

- Example



  - We take the input, and split it into pixels, fit those pixels into the model
    - Each neuron with the layer will learn a different part
    - In the next layer, we have very complex abstractions with activations that are difficult to identify with respect to the first layer
      - So it is a more complex abstraction of the abstractions.
    - After all the hidden layers, you feed it into an output that makes sense
      - In this case, a SoftMax function

Learning in Neural Networks



- How to design an architecture

- Other than weights, we have to decide everything else
  - E.g. How many layers to use?
  - What's the shape of the inputs?
- The neural network will train the weights automatically
- The output will depend on the problem
- The true values are given from the data
- Then combining output values with true values to see the error or loss
- The loss is then fed to the optimizer
  - Optimizers are defined as architectures because a lot of people are working on how to optimize over this complex spaces

Transfer/Activation functions

- Output functions: probability

Extension of the logistic function to multinomial classes. Assuming v variable, the softmax function for each class (target label) is j:

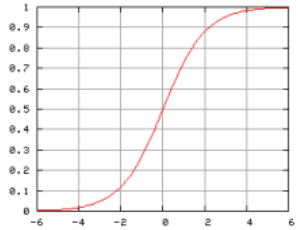
$$f_j(x) = \frac{\exp\left(\sum_v \beta_v^j \cdot x_v + \beta_0^j\right)}{\sum_k \exp\left(\sum_v \beta_v^k \cdot x_v + \beta_0^k\right)}$$

- Each class will have its own weight vector β^j.

When there are two classes, it reduces to the logistic (sigmoid!) function.

$$f(x) = \frac{1}{\exp(\sum_v \beta_v \cdot x_v + \beta_0)}$$

Common as output function of probability estimations!



- Output functions: hyperbolic tangent activation

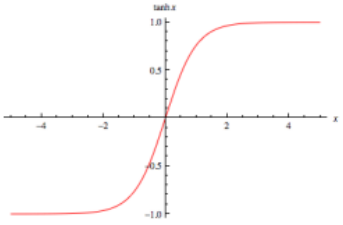
Very common in shallow networks, specially for (bounded) regression!

$$f(x) = \tanh(\beta x) = \frac{e^{\beta x} - e^{-\beta x}}{e^{\beta x} + e^{-\beta x}}$$

It is deemed less plausible than others, but it is very helpful when the activations both positive and negative are likely.

- A regression.

Complex to optimize.



- Transfer/output function: Rectified Linear Unit (ReLU) Activation

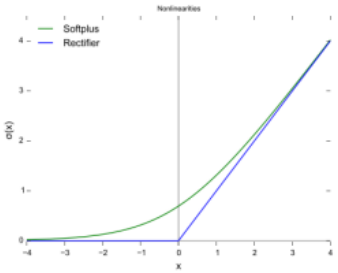
Very popular as a transfer function in Deep models!

- Reason: Easy to calculate, and in complex networks regularization occurs at a model level.

$$f(x) = \max(0, \beta x)$$

Many extensions!

- Softplus: $f(x) = \log(1 + e^{\beta x})$
  - Closer to a logistic regression (derivative is logistic).
- Exponential Linear Units $f(x) = \begin{cases} \beta x & if\ \beta x \geq 0 \\ a(e^{\beta x} - 1) & if\ \beta x < 0 \end{cases}$
  - More complex, but speed up long-term learning. May lead to better results.



- Some doctors started to think: how do people react to stimuli, how do neurons react to stimuli
  - They realized that most neurons actually fire using that shape. They are not continuous.
  - This allows you to approximate any functions
  - Like approximating a circle with dotted lines, by combining, adding up, and subtracting different shapes by traces, if I zoom in, I am going to see the straight lines and spaces, but if I zoom out, it's pretty rounded circle
- This is the first transfer function to try
- As long as you are outside the zero parts, very easy to optimize
- One problem, and this is why the model collapses
  - If you are unlucky, all your neurons go to zero, computer could be looking forever to try to get the weights out of zero.
- So, ReLU is extremely useful as long as you are lucky enough to initialize it well
  - Luckily, a whole bunch of smart people have designed initialization algorithms that come prepackage
  - But still can be unlucky sometimes

Loss Function

- Every optimization problem is solved in the same way: starting from a solution and moving towards the optimum
  - A lot of smart people are working on making them more efficient
  - Stochastic gradient descent: taking samples little by little that will, with some luck, take us to the solutions
    - But this is not robust if you have a long up trend before down trend

- Basic guideline
  - If modelling probabilities, using entropy.
    - May have different shapes
      - Binary: binary cross entropy
      - Multinomial: categorical cross entropy
  - If modelling regressions, using mean squared error.

- Cross-entropy

  We start from a one-hot vector y_{true} of size k (classes) that has a 1 in the correct class, and 0 in the others.

  The model will give us a probability y_{pred}, also a vector with the probabilities.

  Multinomial cross-entropy:

$$l(y_{true}, y_{pred}) = -\sum_{k} y_{true}^{k} \log(y_{pred}^{k})$$

  Entropy measure: Appropriate for probabilities.

  Binary version:

$$l(y_{true}, y_{pred}) = -y_{\text{true}} \log(y_{\text{pred}}) - (1 - y_{true}) \log(1 - y_{pred})$$

- Mean squared error

  Also known as $\mathcal{L}_2$ loss, or quadratic loss.

$$l(y_{true}, y_{pred}) = (y_{true} - y_{pred})^2$$

  Appropriate for regressions.

  Does not punish extrems too much.

  "Leads to higher robustness to the outliers/noise, as we try to maximise the expected probability of good classification as opposed to the probability of completely correct labelling" (Janocha & Czarnecki, 2017)

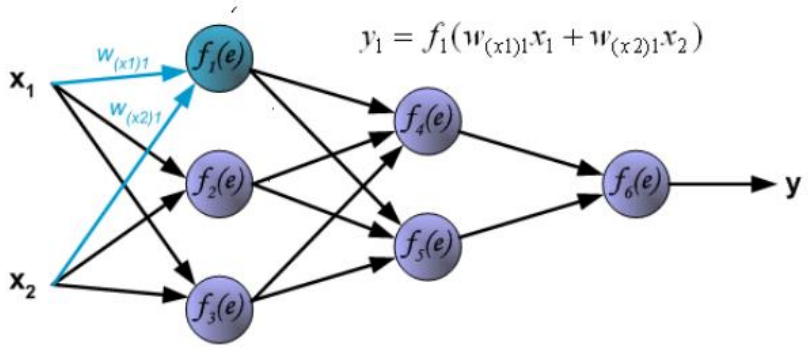
  - Cross-entropy punishes a lot if you make a mistake whereas MSE tolerate that.
  - It maximizes the expected probability of good classification, the average, not global thing.

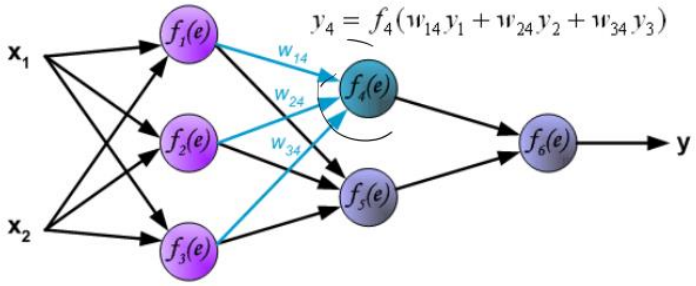
Backpropagation

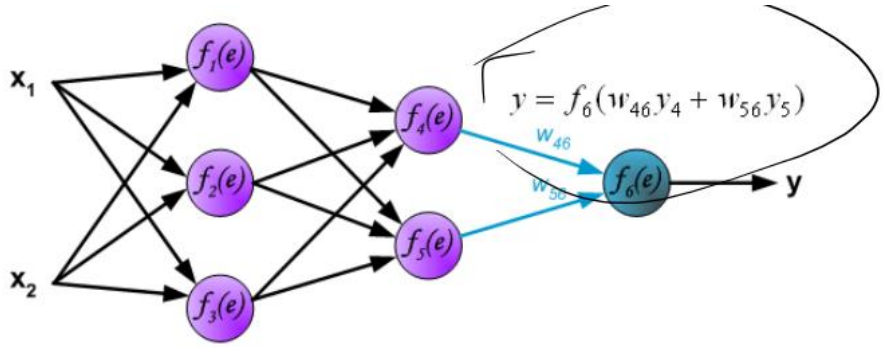
- Current best optimizer is Adam, it's fast but no one knows why it works
  - We know the issues with Adam, but no one knows how to fix it
  - Tried to fix it, but create a whole new set of problems

- So how do we actually optimize these loss functions
  - We will propagate the error back in the neural networks so that we make little tweaks in the parameters
    - That little tweak is called the learning rate
    - So we will change every parameter proportionally to how much they contributed to the error.
    - So if the error was 10, we're going to split ten across every weight
  - Note that the learning rate in neural networks should be very small (e.g. 10^(-6))

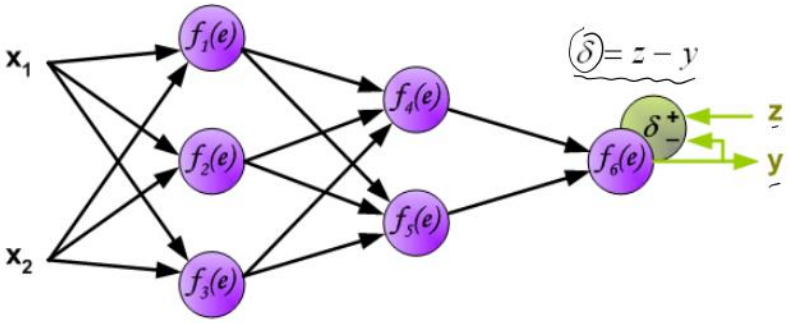
- Consider the process

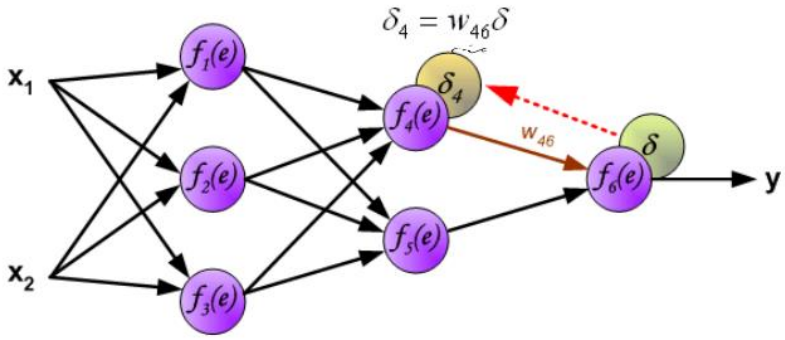


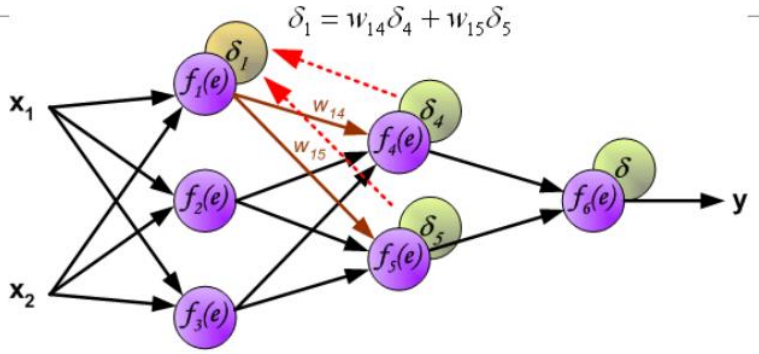


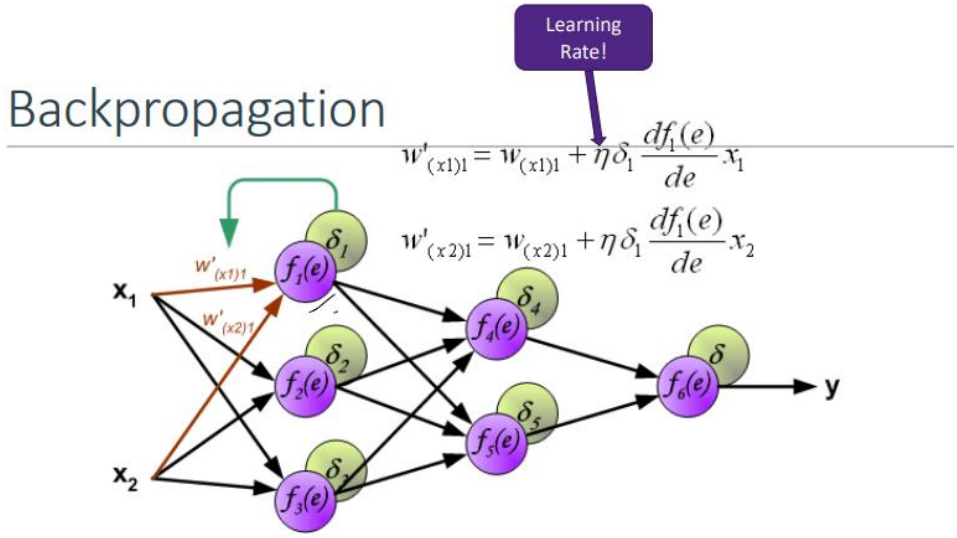


- The backpropagation process

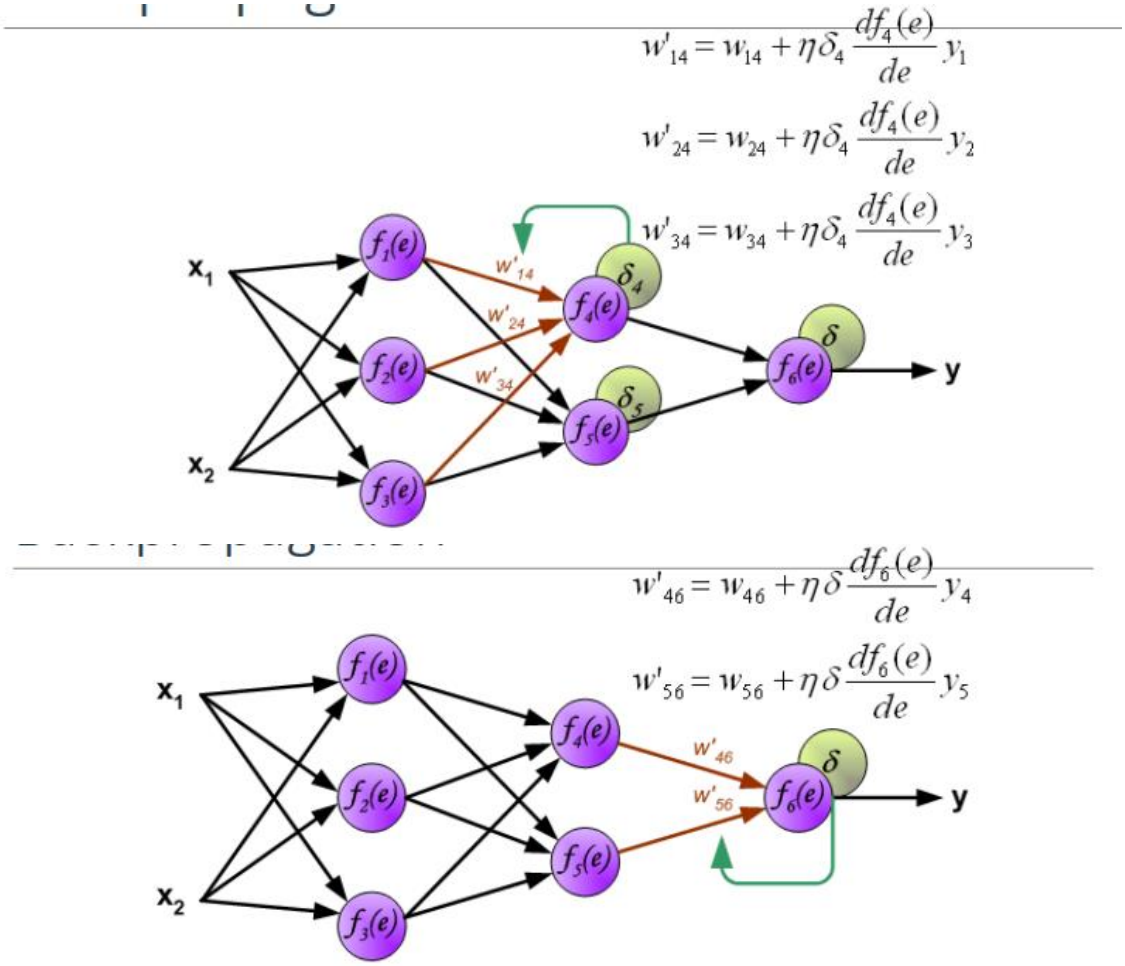








- So take the old weight, plus the learning rate times something that depends on how big the direction of growth is (that's way we calculate the derivative).
  - In ReLU, that's just going to be the original way
  - The learning rate must be small, if it is too high, that large movement in weights will result in an unstable neural network
  - Extremely important because we have too many weights
  - If we stuck with weighting the negative values, that would be 0 in ReLU, the derivative would be 0 and the model collapse. Then restart.



Convolutional Neural Networks

- This is the third wave of artificial intelligence is built upon a convolutional neural network
  - Everything else are just variations and different combinations, stacking layers in a different way. But they are all focusing on the same principles. They will try to mimic this one specific part of how we perceive things, sight. How we understand and perceive the world around us.
  - They were created with image classification in mind. They have been adapted to pretty much everything else out there, text, audio, signals, time series, video, translation, graphs.
  - Whenever you have unstructured data (that can't be copied and pasted into Excel), then deep learning is the top model out there.

Perceiving the World

- We want to determine, what is the shape of this data?

Input Data

- This is a few lines, a couple of circles with certain shape
- Why do you think this is a mouse?
  - Because of how the lines and circles are put together
  - You focus on a set of variables that you identify just by looking at the image
  - Immediately, by mixing those association, you decide that this is a mouse
- The question is: how did you know which parts of that image were actually the ones that you needed to make that decision
  - We have seen mouse before, and leant to identify the key features.

- The whole point about the convolutional neural network is to teach a computer to train itself so it identifies what is relevant to identify these patterns.
  - This is the difference between a convolutional neural network and pretty much every other models
  - These models are looking for local patterns
    - It does not only have to tell that there are ears, it has to tell that there are ears in a certain position
  - So a convolutional neural network is this model that allows the computer to engineer and identify what patterns are relevant, like we do
    - It ignores may other detailed things: textures, colors, these are not necessary for making a decision
  - Comparing this to multiplying age by 5, you can see the difference in finding the type of patterns that we are looking for.
    - In a regression, we are looking for one number that on average affects everyone on the data set.
    - This doesn't happen in a convolutional neural network, we are looking for maps that when they appear, no matter where, they are signaling you the occurrence of a certain phenomenon
    - A map is going to be a variance, a shape or something that hints towards an event when it occurs in sequence with a whole other bunch of maps
  - The convolutional neural network, if training mouses vs. non-mouses, is it will collect the features that tell you that this is a mouse.
    - Those features would be collected (encoded) in weights, then use them in sequence that when it hits all the boxes, then conclusion: you have a mouse
  - There is not much intelligence here, we are simply going to brute force the problem

- How are we going to create maps?
  - We simplify this by assigning numbers taking a look at a block
    - The whole image transferred to numbers is called a map

- We can represent this 3x3 segment as a matrix

$$\begin{bmatrix} -1 & -1 & 1 \\ -1 & 1 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

- This will be called a **filter** and will be a neuron in our CNN.

$$\begin{bmatrix} -1 & 1 & -1 \\ -1 & 1 & -1 \\ 1 & -1 & 1 \end{bmatrix}$$

- We divide the mouse into grids, and see what color is inside here
  - If there is white, it's a -1, if the majority is black, then a 1
  - This is a filter that can activate when a feature is detected
  - Every filter is looking for a local pattern
- The computer will move the filter through the image trying to see if it activates or not, but how do you know whether the filter is activated in that particular part?
  - Simply multiplying (point by point) and adding them up.
  - We call that multiplication (and summation) a convolution
  - So in our example, either filter would be activated against the other

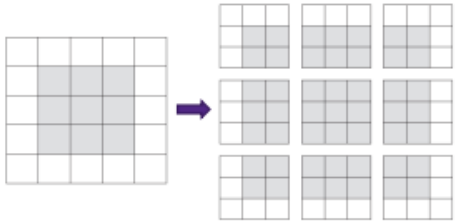
$$\begin{bmatrix} -1 & -1 & 1 \\ -1 & 1 & -1 \\ -1 & -1 & -1 \end{bmatrix} * \begin{bmatrix} -1 & 1 & -1 \\ -1 & 1 & -1 \\ 1 & -1 & 1 \end{bmatrix}$$
$$= -1 * -1 + -1 * 1 + 1 * -1 + -1 * -1 + 1 * 1 + -1 * -1 + -1 * 1 + -1 * -1 + -1 * 1 = 1$$

- Once the filter is compared to the whole image, we have a whole matrix with all the activations, how do I build the maps?
  - You don't, the optimization process would look for the "beta parameters"
  - Note that we don't have beta parameters, here the filters are going to be the unknown, and look for the weights
  - It will look at patterns, and it will only keep the ones that are useful to make the prediction
  - And the first layer (with many filters) would identity and save many basic features, an ear, a nose, …. The second layer will look for combinations of ear, noses and eyes. The third layer will see combinations of combinations
- In general, images, audio, text, time series has spatial hierarchy (initial layers create basic blocks, subsequent create complex ones from theses)
  - Neural network is good with them, but it is not designed for traditional data (structured data)
  - The structured data (income, age, …) can only give you global patterns
  - Neural network can be used with mixed datatype because it can combine different sources of data very well
- When can we use deep learning?
  - Whenever inverting 2 columns leading to a completely different example
    - Pictures of two people in the opposite position results in a completely different input matrix
    - Inverting columns of age and income still results in the same information.

Defining a convolutional layer

- When we are building a neural network, we are deciding the shape of the model (e.g. what size would it have?)
  - ➢ We would decide the size of the kernel (the filter), most are squares
  - ➢ The second thing we need to design, and this would define the convolutional layer, is how many filters to use in that particular image?
    - ✧ How many things are relevant? Maybe eyes, ears,…
    - ✧ The number of maps is important to keep as power of two so it could fit into memory more effectively.
  - ➢ These two things thing would compose one convolutional layer
    - ✧ It input one combination in and have one combination (variable) out
    - ✧ One filter will be stored for every neuron in the convolutional layer

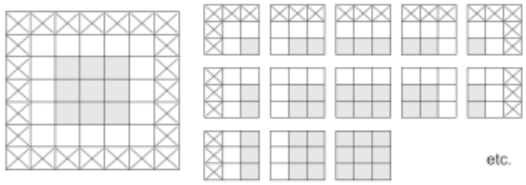
- Padding



We can only create 9 different features of size 3 x 3 in this 5 x 5 image!

Solution: Add **padding,** or a sequence of 0's to get to our desired size of the feature map.

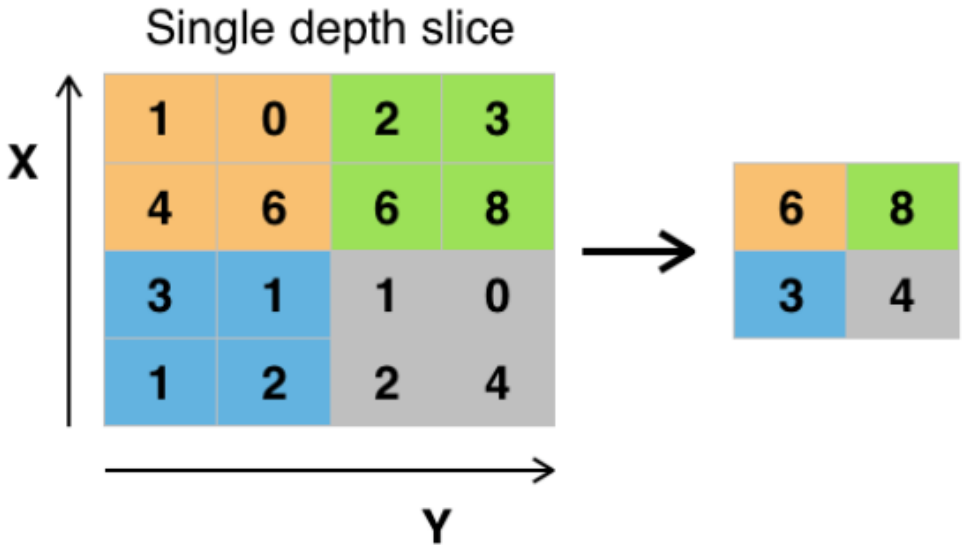
For example, to build a 5 x 5 feature map we need to pad the image like this:



  - ➢ Adding 0s on top of original image so that we can capture features in the corner of the image

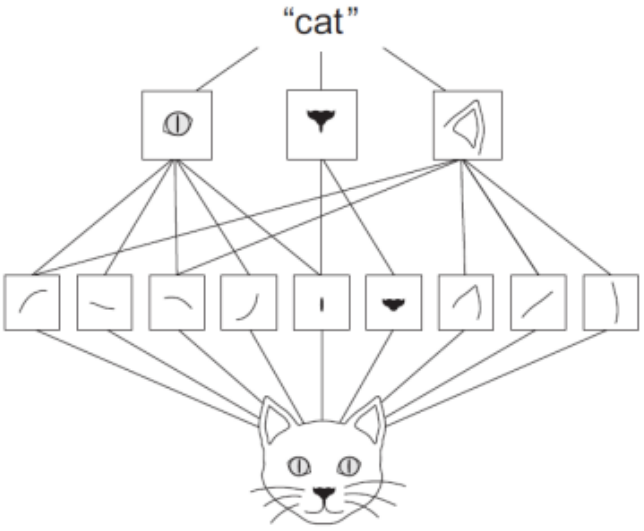
- Stride
  - ➢ If we move the filter through the image at every pixel, we may need to store a whole lot of weights, we will have a very big inefficient model
  - ➢ Maybe if we are looking for bigger patterns, we can use bigger filters. Or we can make smaller filters and just move them more (larger stride)

- Pooling
  - ➢ It is a very inefficient way by looking just at stacks, we can get rid of things that are not very useful, so we use pooling
  - ➢ We are going to take several parts of the map that we just built, and we're going to just keep the ones that are more important to our problem
  - ➢ This is a new layer that goes on top of the convolutional layer
    - ✧ So we might have, convolution, pooling, convolution
  - ➢ That's going to comprise the models
  - ➢ In general, we would use max pooling. Which means keep the most significant signals
    - ✧ However, we could also use average, minimum pooling for specific reasons.



  - ➢ This is usual because most of the times, we only care about max activation, why not just take the maximum of the whole matrix?
    - ✧ Because sometimes it's not just whether you have them or not, its whether you have the map or not in combination with where it is on the map, you would have all the convolutional layers on top, and they would be looking for where on the map you appear, not just whether you appear or not.
  - ➢ So pooling will compress the convolutions and make them more efficient, at the cost of potentially deleting things that are useful

- Stacking layers: learning complex patterns, if it wants to predict a cat



- Each feature map will store a very simple part of the cat first
- Then those would be combined into higher level abstractions
- Then those are combined again into learning

Tensors and Input Layers

Given that there is a lot of matrix operations in a neural network, we can generalize the way of thinking on them by the use of the mathematical definition “Tensor”.

Tensor: A generalization of matrices to undefined dimensions.

- Vector: a 1-dimensional tensor (1D for short).
- Matrix: a 2-dimensional tensor (2D for short).
- Matroid: a 3-dimensional tensor.
- Videos can use up to **5D tensors**!

Every layer will always be represented by a set of tensors of some dimensions.

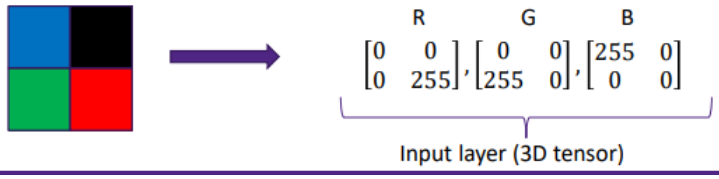
- Thus, Google’s name for their software “Tensorflow”.

- Tensor is basically a matrix, a vector, a sequence, so a representation in a certain number of fixed dimensions
  - If you have audio to the video, you will have up to 5 dimensional transfers, width, height, color, time, and audio track
  - Tensors are general representation of things, and whenever one layer is represented by a set of tensors of a certain fixed dimension, we’re going to apply this ReLU functions and transformations that turn them into a different tensor.
  - Thus, a neural network takes sensors and reshapes them as they flow through the neural network, thus the name TensorFlow.

- How do I turn a picture of something into actually a set of numbers that a computer can read?
  - This is representation embedding.
  - Embedding is a numerical representation of something
    - E.g. text, a set of words
  - The idea here is to use as many numbers as you need to create an accurate representation to your level of need.
  - For images:

For an image, we will use their channels.

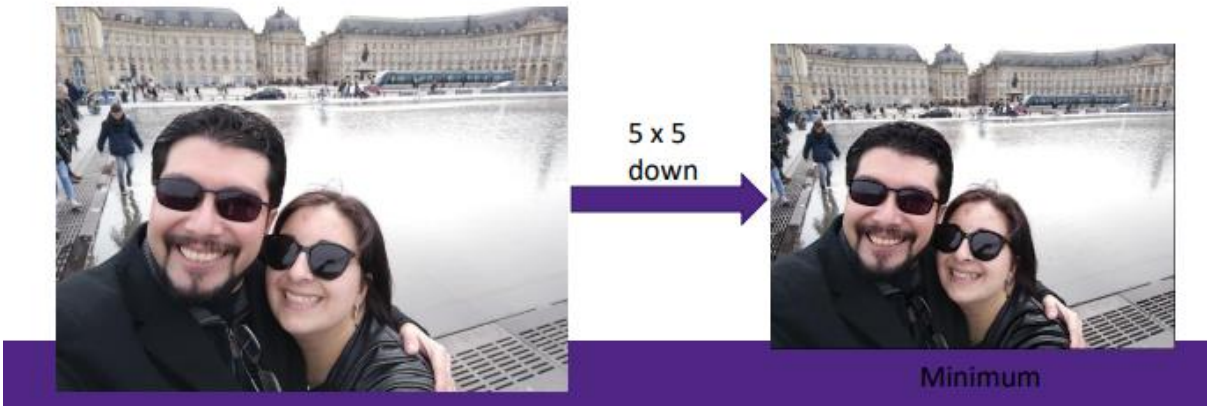
- Channels: Colours available. For example, Red-Green-Blue (RGB) represents each colour as a value between 0 and 255.
- Input results in a 3D-tensor $\{v_{hwc}\}_{h,w,c}$ such that.
  - h: Pixel height
  - w: Pixel width.
  - c: Channel.
  - v: Colour value between 0-255.



    - White is all colors set to 255, black is all colors set to 0

Image

- When you take an image, depending on what you want to do, and because the typical image is too big, you can do some downsampling
  - You will compress this image to get rid of the information that you don't need (downsampling)
  - How do we do this?
    - It can be down by taking a block, take the max, min, average pooling operation to the picture without any convolutional layers
  - If you want to ignore the details at the back of the image (ignore the level of details that preserved for whatever is in the background at the map),
    - You can do a minimum downsampling, and the image will be smaller and darker portions would be reserved



- I can also use max downsampling and average downsampling

  - For maximum pooling, the whites are going to be kept
  - Average downsampling is giving you this negative view because when you use average, you lose the information on the corner, so you get this negative distortion. This is very useful to differentiate things that are really different than the background

Time series

- Time series are two dimensional tensors
  - You have time and value
  - Neural networks can allow you to actually combine multiple time series.

Text Data

Text data has the particularity that it does not have a natural numeric representation, as in images.

Several have been invented.

- LSA: Replace each document by the first K LSA features. 1D tensor!
- One-hot: Replace each word by a dummy-coded vector. Each document will be a matrix with the stacked vectors. 2D tensor.

Neither one of these transform defines a **topology of language**.

- We want words with **similar meanings to have similar representations**.
- This requires **training an embedding**.

- If you use Google Translate, there is a neural network behind
  - It turns text into numbers

Architecture

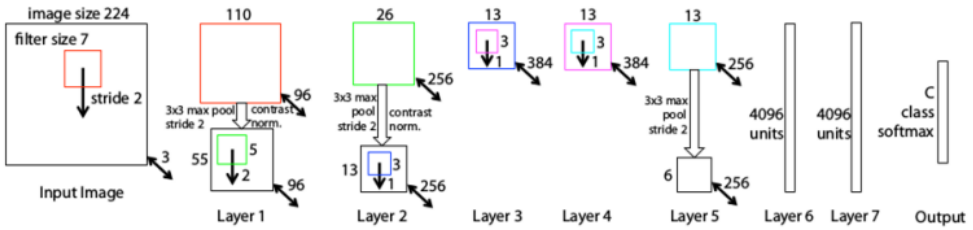
- Building an architecture is basically designing all the blocks
  - How many maps to create
  - It's more of an art than engineer, there is no particularly methodic way of using formulas to create, there is a lot of handwork here, going step by step by step, …
  - There is a tradeoff between the computational capacity that you have, how complex is your problem, what's your output
  - The thing about creating an architecture is that finding completely new ways of doing things is quite hard
    - Big classes: CNN, recurrent model, attentional models, generative adversarial networks, reinforcement learning

ZF Net (2013)

Derived from seminal AlexNet (Krizhevsky et al., 2012), first real CNN applied.

ZF Net (Zeiler & Fergus, 2013). Winner of the ImageNet Large-Scale Visual Recognition Challenge ILSVRC in 2013.

Goal: Image recognition with 1,000 different classes.



ZF Net Architecture

- In most models you will see that input data is a downsample that leaves the picture as 224*224 pixels
  - There are more models available that are able to actually scale this up by a factor of 3, 4, but rarely more than 1024 * 1024
  - Step 1: the first convolution will use a filter of size 7 by 7, and use a stride of 2, and there are 3 channels, one for red, one for green, and one for blue
    - What is the input size? 224 *224 * 3
  - Then, the first convolution generates a set of maps that are of size 110 * 110, and 96 maps (i.e. 96 features)
    - How to get the number? The answer is how many pieces of size 7 by 7 with a padding of one actually fit within a 224 * 224 image when you are moving it with a stride of two
  - Step 2: do a 3 by 3 max pooling with a stride of 2
    - Now the feature maps would be 55 by 55
  - Step 3: perform a second convolution of size 5 by 5 with a stride of 2, and build 256 feature maps
    - General architecture rule: start with bigger kernel (filter) sizes but less maps, as the model grows, you would reduce the kernel size, and increase the number of maps
  - …
  - In the final step, there are 384 feature maps and filter size of 3 with stride of 1
  - In the final phase before going into dense neurons, they reduced the feature maps to 256 and perform a 3 by 3 max pooling resulted in 256 maps with size of 6
    - Argued: is this a good idea? Maybe use drop out?
    - The neural network generates very high level abstractions of the original data. So we end up with 256 * 36 final abstractions
  - Now, we feed this to the traditional dense neural network to do the prediction
    - This is the top or head of the model
    - Everything before the dense layers is looking for local patterns, everything in the dense layers is looking for global patterns

VGG (2014)

Simonyan & Zisserman (2014)

Simple architecture:

- 3x3 maps, stride 1.
- 16 stacked layers.
- Three dense layers after final max pooling.
- Softmax output for final classification.

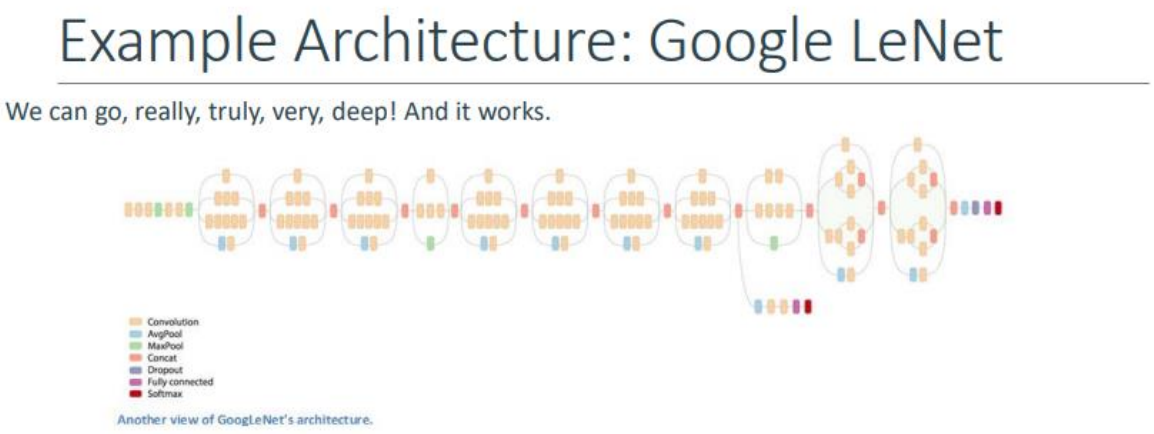
Key takeaway: Hierarchical representation is the best way to classify.

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB imag)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

The 6 different architecures of VGG Net. Configuration D produced the best results

- Instead of always having convolution, max pooling, convolution, max pooling, …, it will create blocks that will look at the same sizes, it created a whole bunch of things
  - At the top of things, it created 11 layers, 11, 13, 16, 16, 19 layers called VGG-16, …, VGG-19 (so D and E, nobody remembers the rest)
  - The innovation is that they created blocks so they have a sequence
    - With 224 * 224 inputs, they created 2 convolutions kernel size 3 with 64 feature maps
      - So they didn't start with large kernels, they stick to 3, but they increased the number of maps at every iteration
    - Then it follows by a max pooling, then another 2 convolutions, max pooling, then 3 convolutions in sequence, and then max pooling.
      - The theory was that they were creating abstractions within the same number of maps. They were still building on top of other, but without really trying to look for different segments. They are all at the same size and still creating complexity by stacking them on top of the other

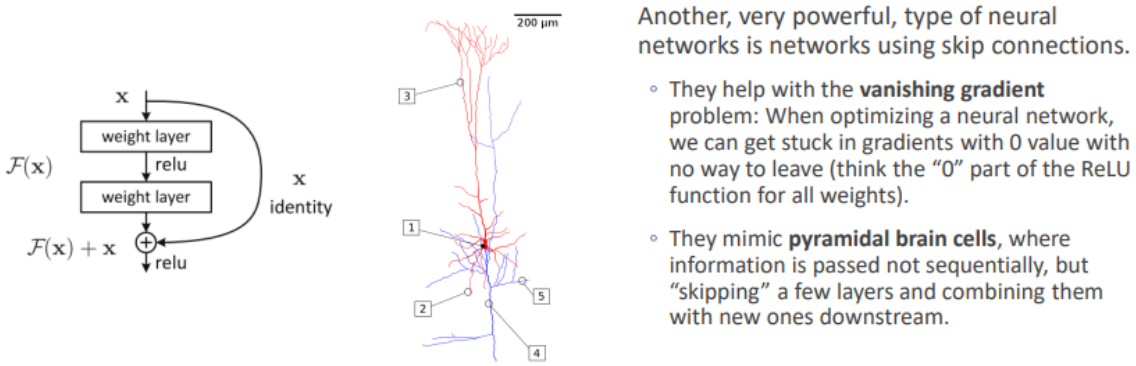
Google LeNet



https://adeshpande3.github.io/The-9-Deep-Learning-Papers-You-Need-To-Know-About.html

- Every block is one layer, the most notable innovation is the serialization of layers that opens up inputs and it creates sequences that run in parallel
  - For example, the first serial block consists of 1 convolutional layer, 3 convolutional layer, 5 convolutional layer, 1 average pooling + 1 convolutional layer. And the output of those four get stack together to create a larger map, that's what the concatenation layer does.
- This is a completely new way of thinking about things, thinking about serializing convolutions.
- Another innovation, it has two outputs (two SoftMax)
- We can see that creating this gigantic deep model work
- It had a lot more experimenting than engineering

Residual Networks

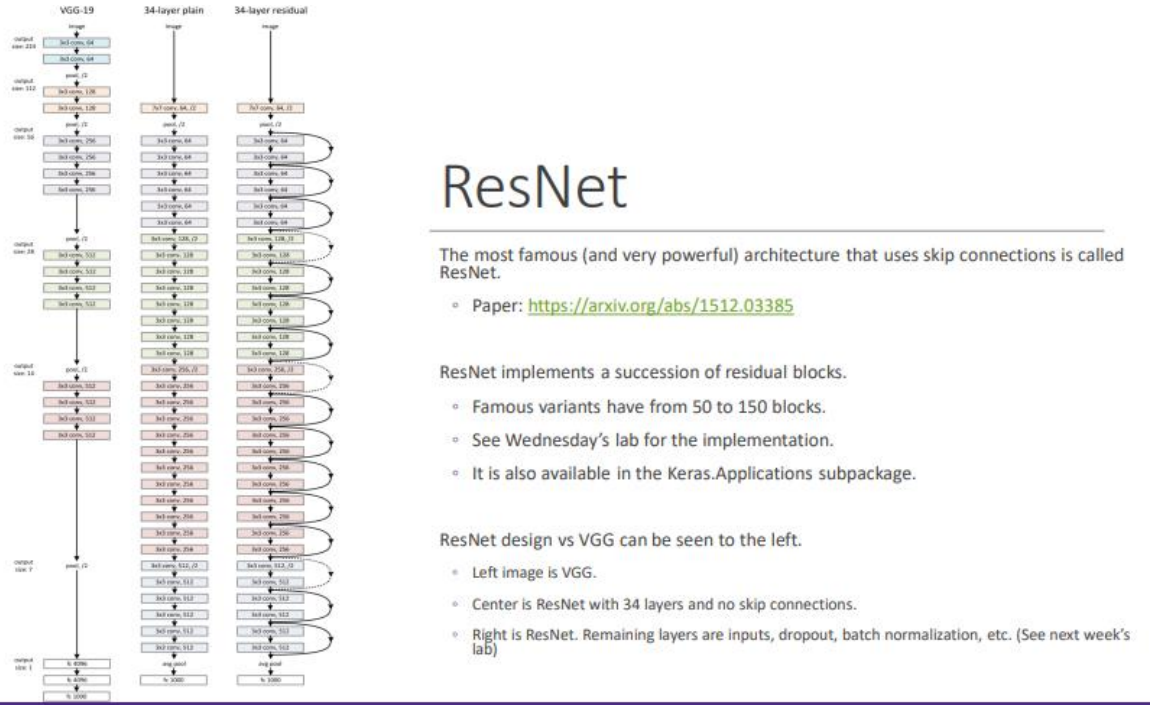


- They realized that looking at two neurons, they don't just connect serially, they don't just say like neuron A should store neuron B and neuron B catches the neurotransmitters and second the signal forward. They

realized that neurons send connections to the end of the neuron, skipping some neurons all together.

- So you are connected in such way that you skipped one connection, and you pass forward the connection to the next neuron. So it's not just sequentially, it also goes bypasses.
- Before, if you stuck at zero in the middle of ReLU, it's very hard to get out and go back to the linear part (vanishing gradient problem). They realized that this could be dealt with by skipping part of their original information in the flow
- So we have a residual block like the ones above
  - It has convolutions, but the inputs to the very fist layer of the block is also going to be passed directly to the end of the block
- A problem for previous architecture is that if you start stacking too many layers in VGG, (e.g. 34), you would have tiny kernels and huge amount of maps and everything is 0. Your gradient vanishes along the way. How do you minimize the risk of this? That's why this structure is useful

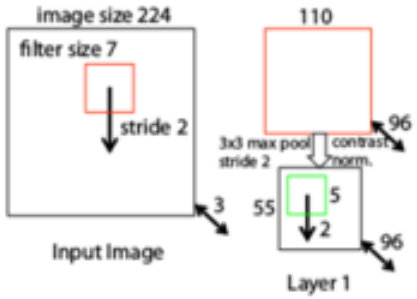
ResNet



- The third one is called ResNet50, but it only uses 34 layers
  - The reason is that there are max pooling layers in between and stuff like that get counted dropout layers. So there are 16 layers of other things that support the main convolutions
- ResNet can go all the way up to 150 layers
  - Compared to VGG that after 19 layers, the problems of vanishing gradient are getting very serious
  - So they took the VGG 19, they try to do the same with ResNet, put in 34 layers, and the gradient vanishes all the time because it was too much. You could not get to the layer of complexity. However, if you start to make skip connections with several residual blocks. (We call this residual because we add up the very first part, it's like $x + F_1(x) + F_2(x)$). What's why you need to make the convolutions match in size
- The resulting model will be huge with 30-100 layers so ResNet is commonly used for offline systems

Question

- If I have a filter for the tail and its long, do I have to have a large filter (e.g. 50*50 pixels) in order to see if it is activated? If so, the tail would like to be in different shapes in different pictures, how should I choose the shape of my filter?
- Do we have a threshold for the convolution (the multiplication and summation) to indicate whether the filter is activated? (e.g. if the number is greater than 5, then it is activated). I realize that the max pooling is taking the greatest number among 4 or maybe 9 blocks in the feature map, how do the computer decide what blocks is activated
- Starting from second layer, are we still using filters to identify higher level features? I imagine that would be extremely abstract to design
- Once have maybe 32 feature maps as the output of first convolutional layer (by applying 32 filters to the original image), how would I proceed with the second convolutional layer, do I apply several filter maps to each of the 32 feature maps? And result in maybe 64 feature maps (applying 2 filters each)? In the process, it identifies what feature is activated and where in the image is activated, but it didn't really combine the features (e.g. be able to tell the ears in the right position compared to the other features), is this applied in the later process?
- Three by three max pooling with a stride two, overlapping information?
- How does this result in size 110? What about the padding to the end of 224?
    - With 0 padding, we start from 6, 8, 10, …, 224, so $\frac{224-6+2}{2} = 110$



- For VGG, why use 2 convolutions before max pooling, what is the purpose of performing different convolutions, they each produced 64 feature maps, is the difference between the type of those feature maps, can we just create 1 convolution with 128 feature maps by applying the different filters in each of the two convolutions?
- For ResNet, I am adding the original input with two convolutional layers which perform feature extraction, the output from convolutional layers would be smaller than the original image, so in order to match the size, do I downsample the input image using the size of the filter used in the two convolutional layers?
- What is the parameters for convolutional layers? In a dense neural network where each neuron is connected with every other neuron preceding it, the parameters are the weights applied, but convolutional network only apply filters across the image resulting in a matrix representing the feature map, and feature maps are kept separately, so I am wondering what is the parameters in this case?