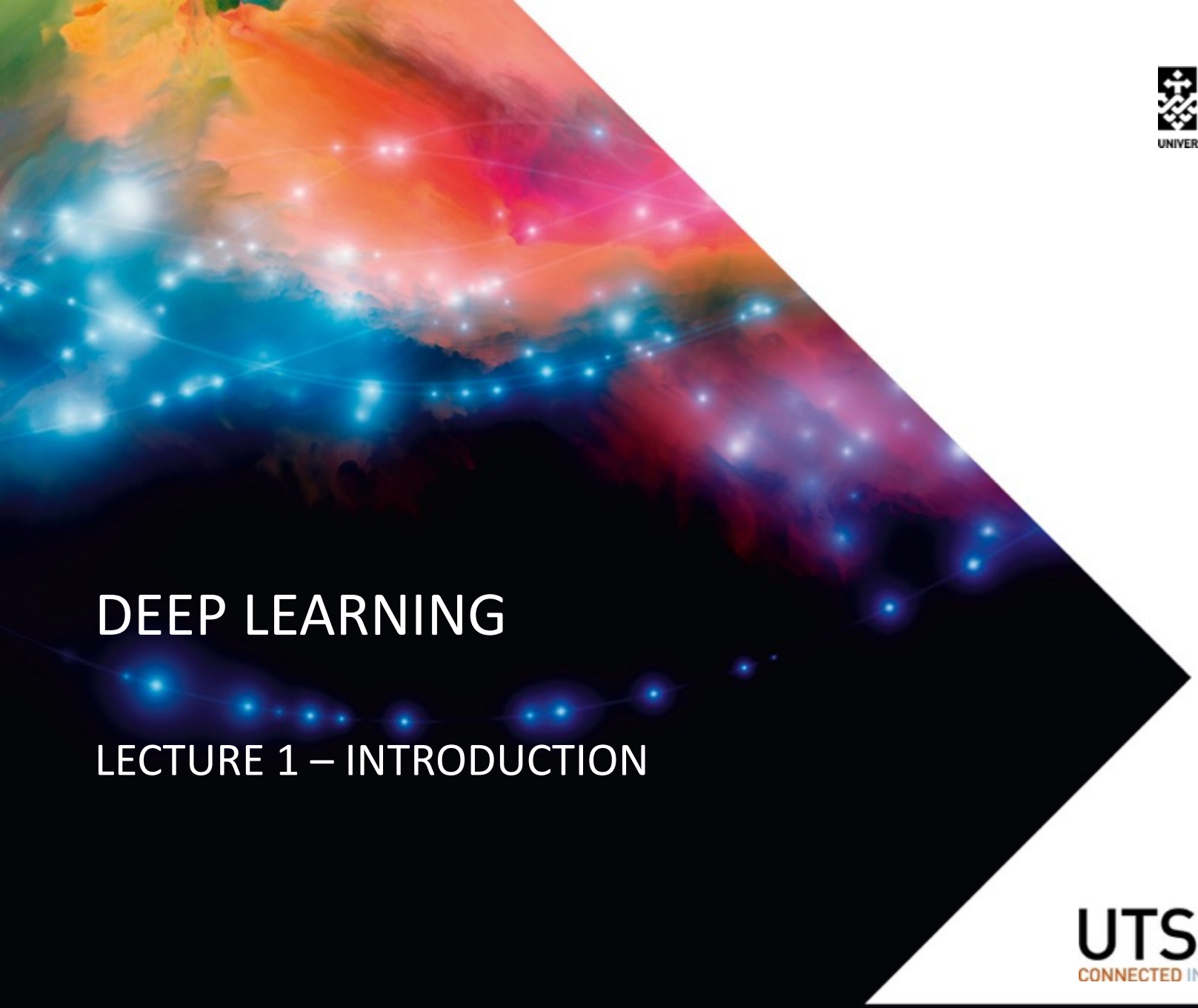


UTS
UNIVERSITY OF TECHNOLOGY SYDNEY

DEEP LEARNING

LECTURE 1 – INTRODUCTION

UTS:CIC
CONNECTED INTELLIGENCE CENTRE

OVERVIEW OF TODAY

Schedule:

> AT1A Feedback

> History of CNNs

> CNN Architecture

> Famous CNN Architectures

> Computer Vision Tasks

Lab:

> Introduction to CNNs in Keras and Tf

OVERVIEW OF TODAY

Schedule:

> AT1A Feedback

> History of CNNs

> CNN Architecture

> Famous CNN Architectures

> Computer Vision Tasks

Lab:

> Introduction to CNNs in Keras and Tf

AT1A FEEDBACK

- The good, the bad and the ugly
  - The story of this assignment

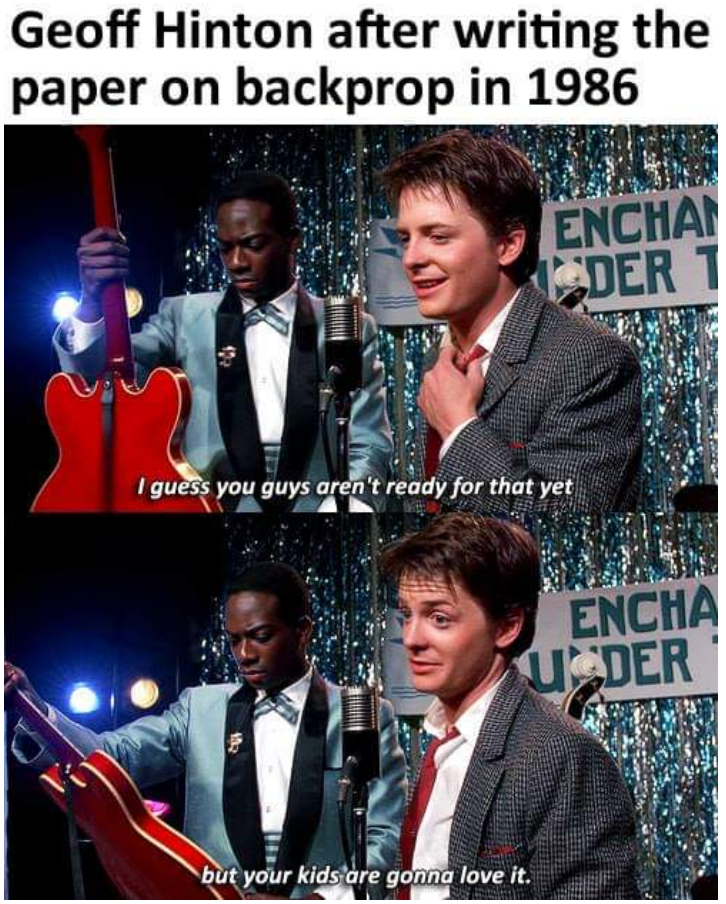
- Generally done exceptionally well

  - Particularly impressive experimental work generally
  - Reach out early for help

- Compiled answer notebook from the class available.

SOME HISTORY OF COMPUTER VISION

- (Video)
- Let's start in the late 1980's
  - Hinton was working on back-prop algorithm (Source, 1986)
    - *'learning representations by back-propagating errors'*
      - *Some 17,145 citations…*
  - Yann LeCun independently as well (Source, 1989)
    - *'backpropagation applied to handwritten zip code recognition' ('Pre-Lenet 1')*
      - *Some 4,500 citations*



SOME HISTORY OF COMPUTER VISION

- Then improvements, released LeNet. Various versions. LeNet-5. (LeNet-5)
  - *'Gradient-based learning applied to document recognition'* (*Source, 1998*)
    - *Some 17,895 citations…*
  - Same Bengio as in *Bengio & Bergstra* (2012) random search paper

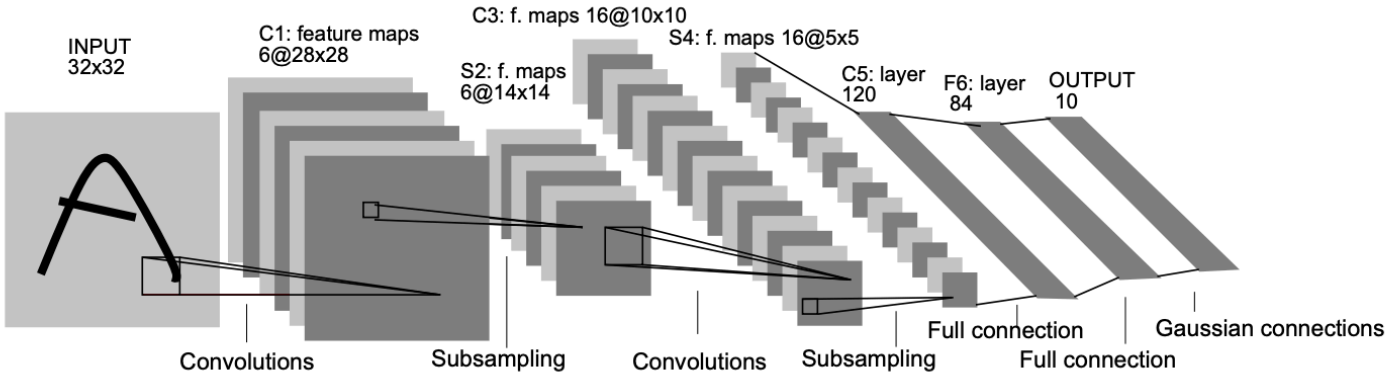
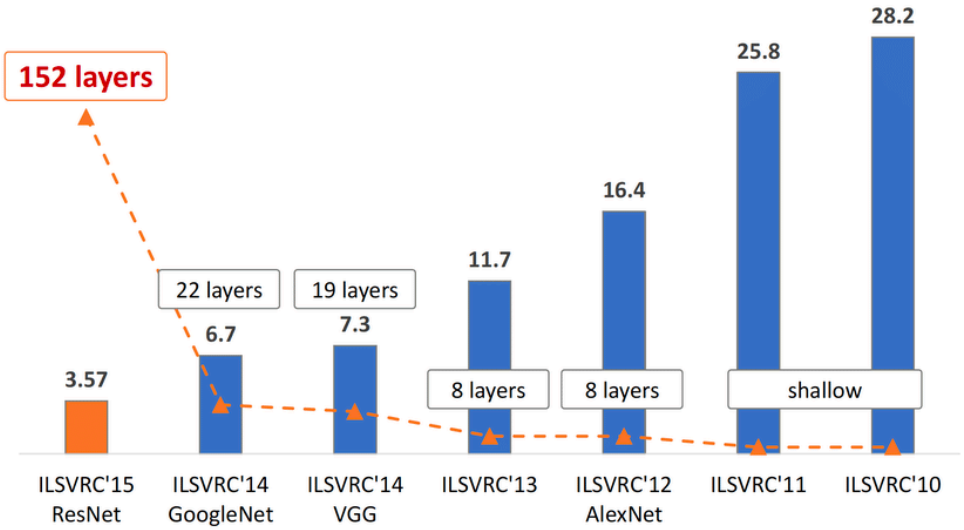


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

THEN NOTHING…..

THEN IMAGENET

- In 2009, Fei-Fei Li and others began working on putting together imagenet using mechanical turk.
  - *1,00 categories. 14M images.*
- Competition (**ILSVRC**) run in 2010 first (Stopped in 2017)
- Huge drop in 2012 with AlexNet, presented at NIPS 2012 (Source)
  - *38,561 citations*

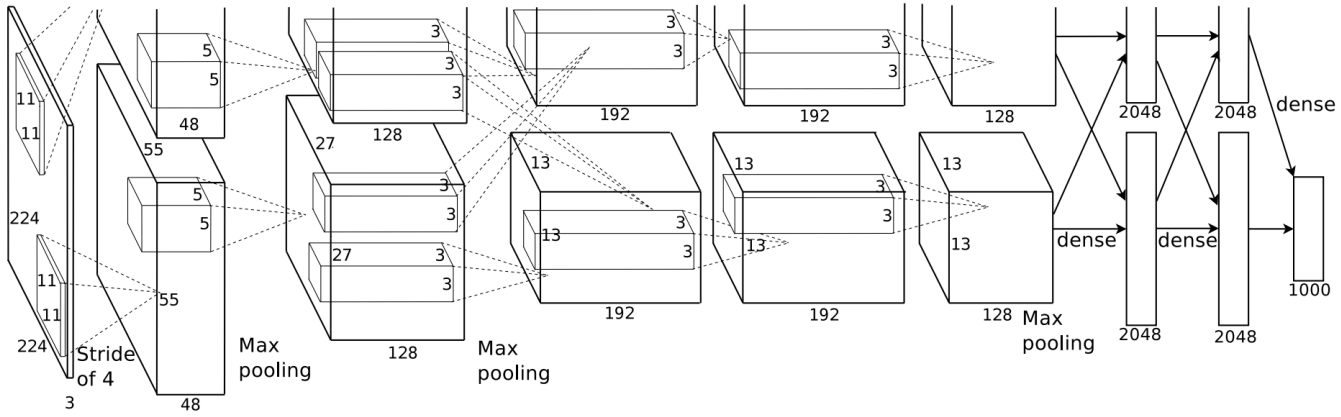


A NOTE ON ILSVRC

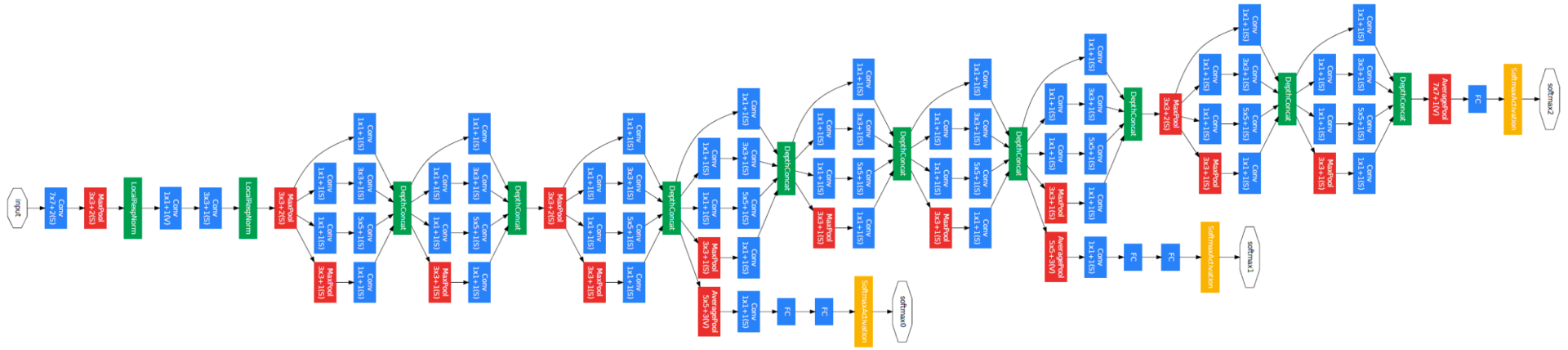
Year	Winner	People	Company/Uni
2012	AlexNet	• Alex Krizhevsky • Ilya Sutskever • Geoffrey Hinton	University of Toronto
2013	Clarifai	• Mathew Zeiler (founder)	Clarifai
2014	VGGNet	• Karen Simonyan • Andrew Zisserman	Oxford University
2014	GoogleNet	• Christian Szegedy et al.	Google
2015	ResNet	• Kaiming He • Xiangyu Zhang • Shaoqing Ren • Jian Sun	Microsoft

WHAT IS A CNN?

- A CNN is constructed of a number of different layer types we have not seen before:
  - *Convolution layers*
  - *Activations - We know this ☺*
  - *Pooling Layers*
  - *Fully Connected layers*

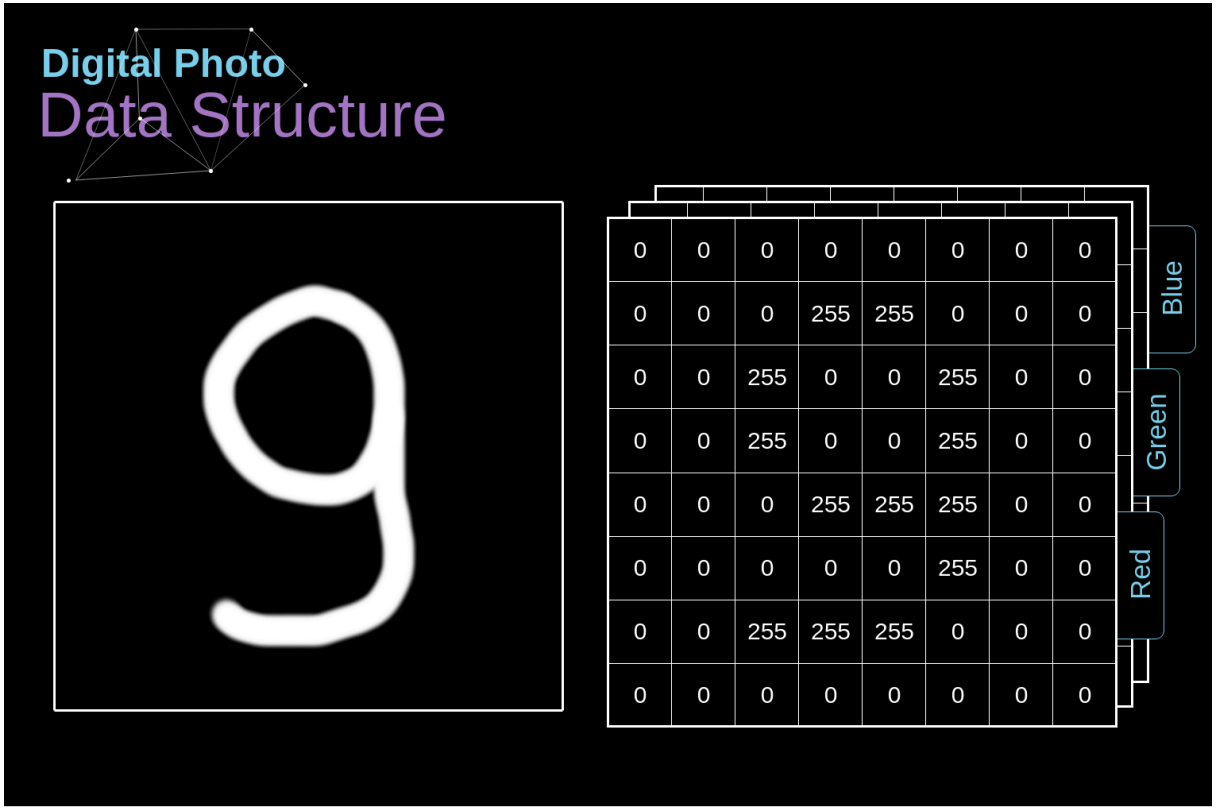


YOU WILL UNDERSTAND THIS ☺



SOME BACKGROUND

- There is a large ‘semantic gap’ (link) between an image as a computer sees it and the way a human sees it.

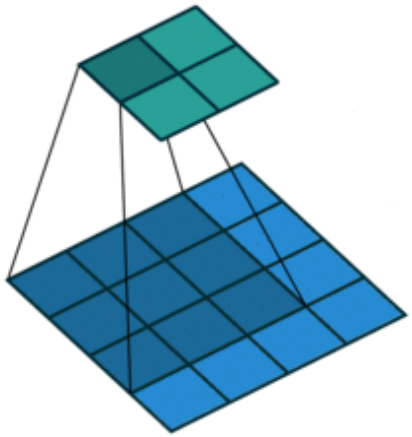


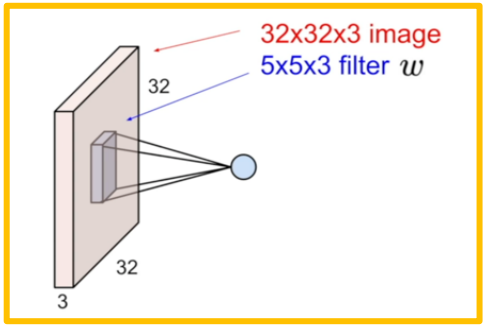
ADDITIONAL DIFFICULTIES

- Occlusion (Part of the object is visible)
- Illumination
- Class variation (‘What is a cat’)
- Background, other objects in the image

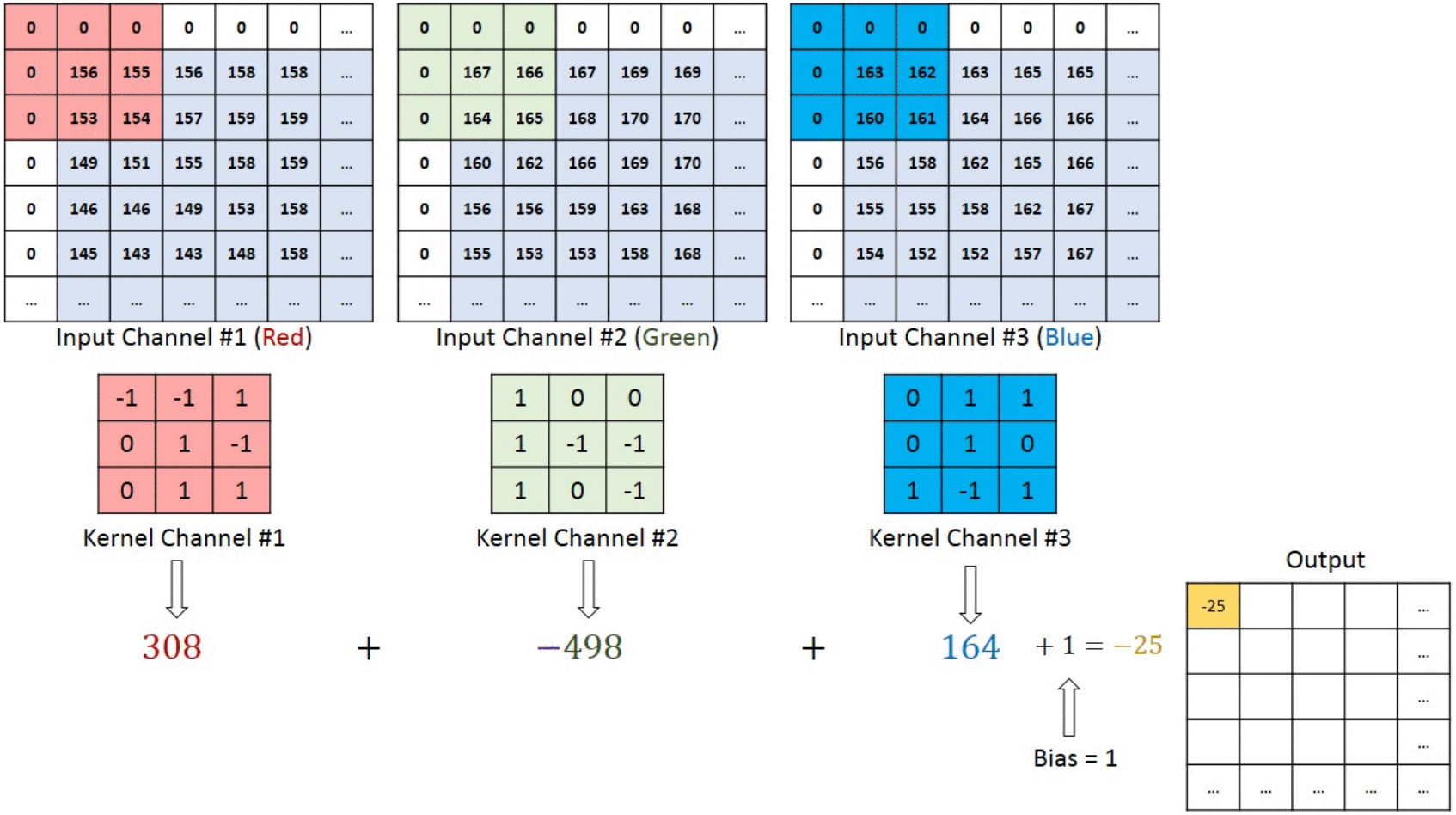
CONVOLUTION

- We want to preserve spatial structure so employ a clever trick of matrix multiplication to build '**Feature (Activation) Maps**' of our image.
- We define a Filter/Kernel that undertakes multiplications, slides across, continues.
- Each multiplication makes 1 number on our 'Activation Map'
- However, we do this in 3-dimensions rather than just a single dimension.
  - This is called a **Tensor Contraction**
- The filter/Kernel must have the same depth as the input

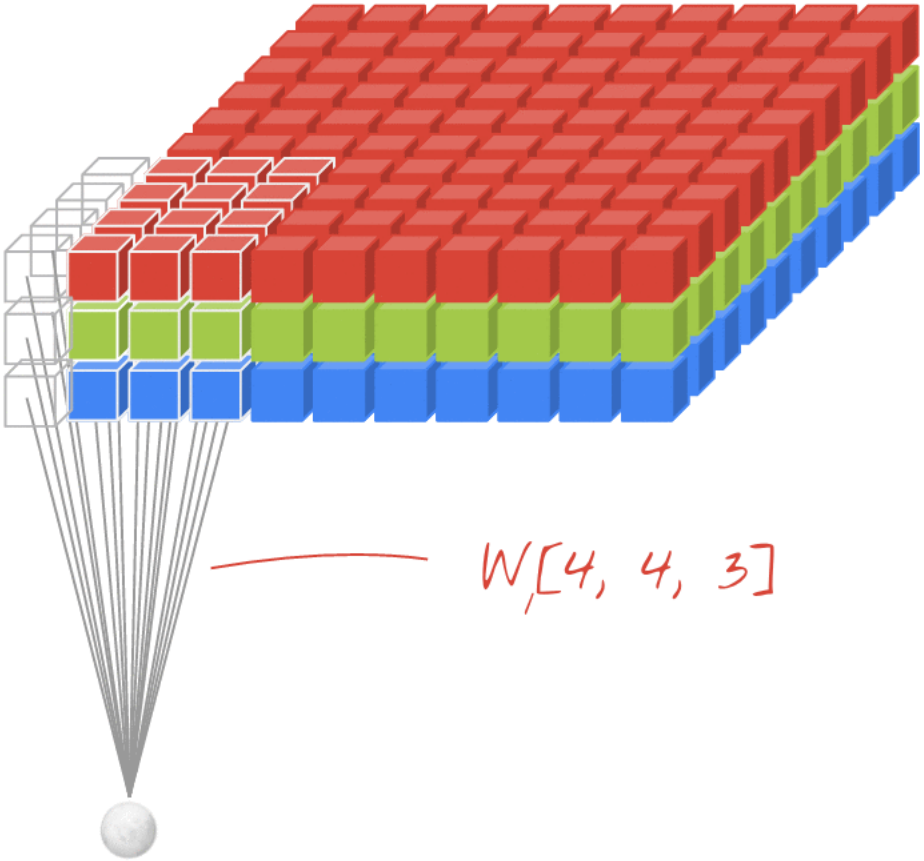




CONVOLUTION IN 3D



CONVOLUTION IN 3D



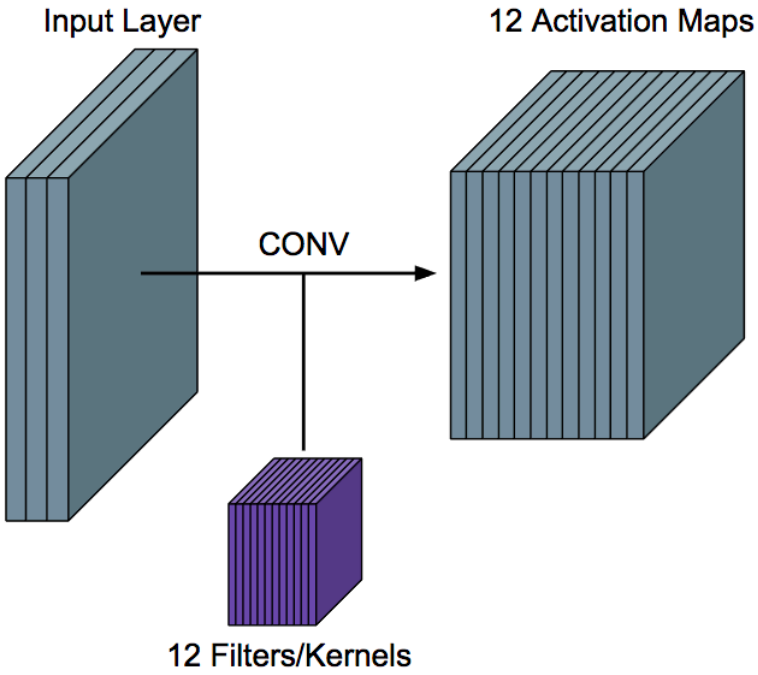
Source

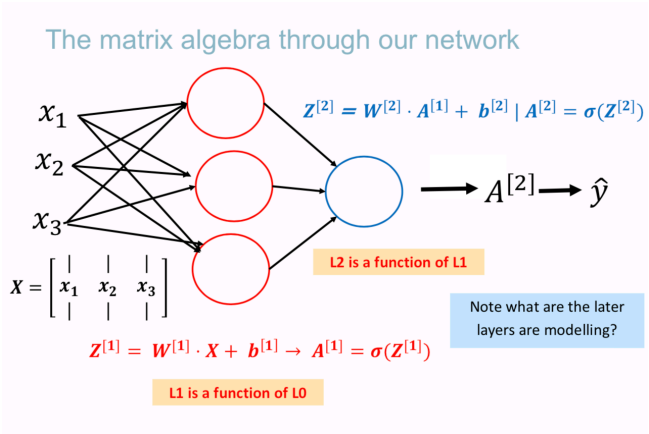
CONVOLUTION

- The output of this 3-d matrix operation is only 1D, so we can stack as many of these Filters/Kernels as we like
  - Therefore stacking activation maps.

- So we make ‘New Images’ which are fed into the next layer, building higher order features.

- This sounds familiar…..





CONVOLUTIONS & RECEPTIVE FIELDS

- You will often here the term ‘Receptive Field’ around CNNS
  - The filter (3*3 or 5*5) can be considered a *‘Receptive field’* for each Neuron
- A *neuron* in CNNs is a point on the activation map.
  - Each neuron in the activation map shares the same parameters (same filter)
  - But each neuron in the same *position* across activation maps will be looking at the **same region** in the input but for **different things.**

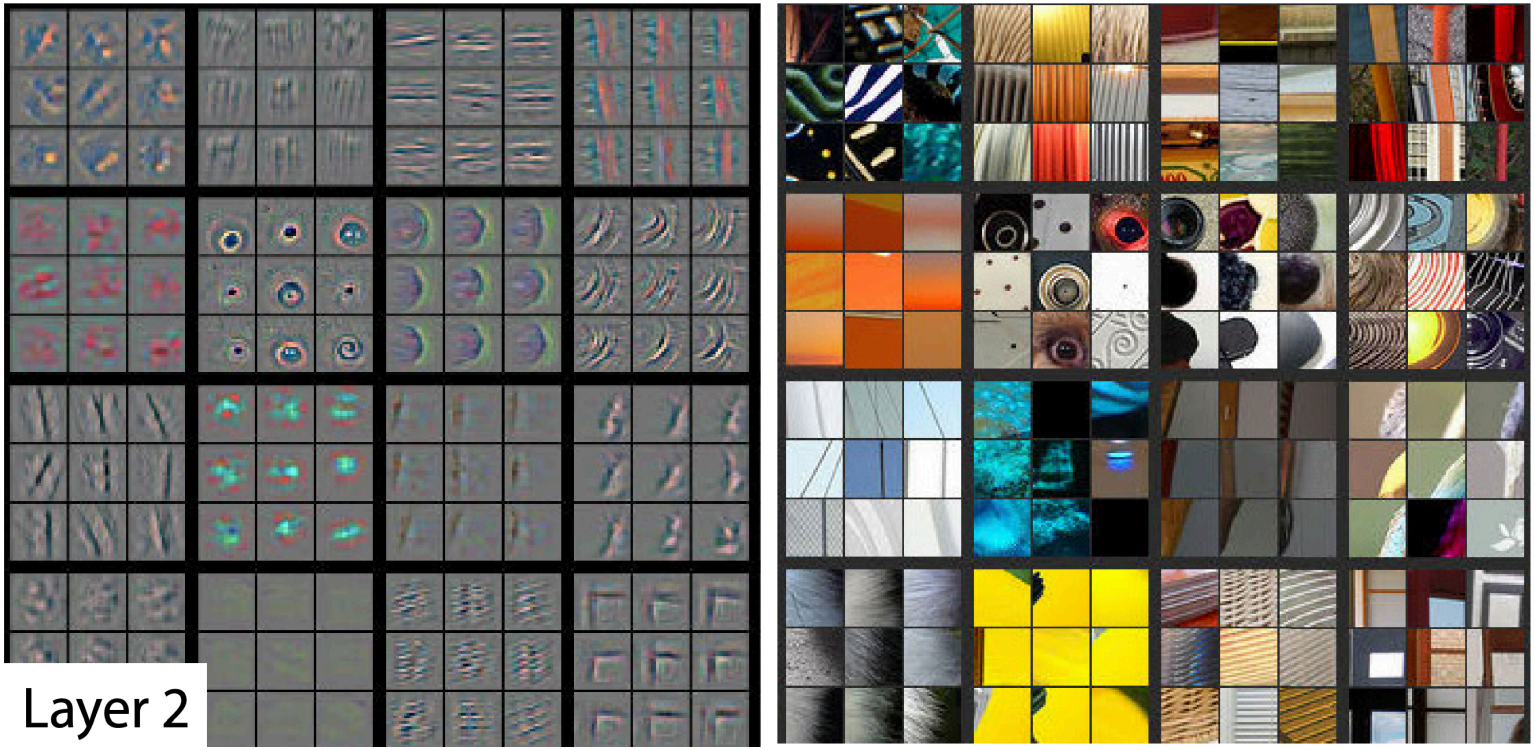
VISUALISING INTERMEDIARY LAYERS

- A lot of work has been done to assist visualising what is going on at intermediary layers.

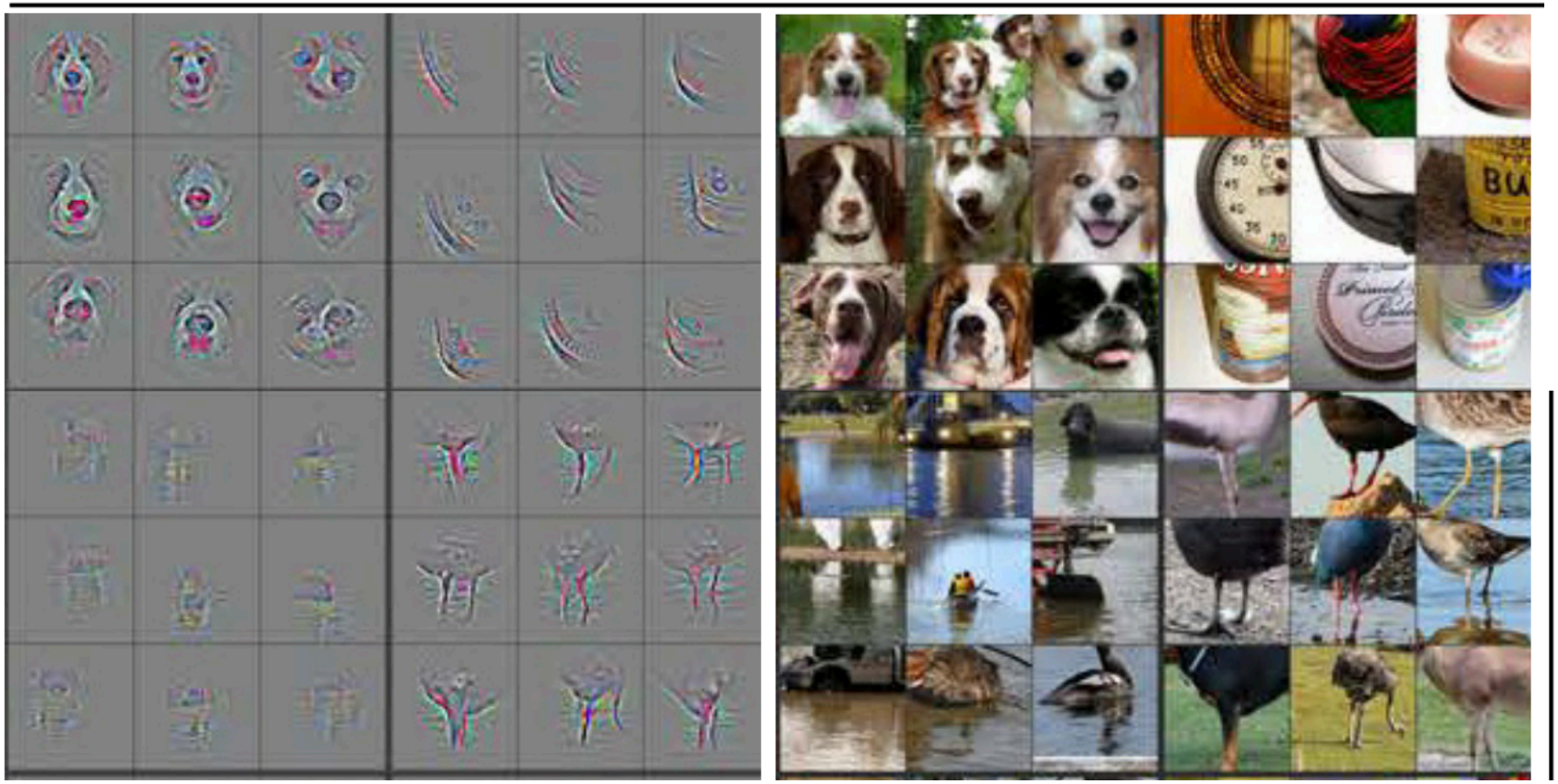
  - Since computer vision is ‘visual’, perhaps we can ascribe some human meaning to these layers and features?

- The classic paper (link)
  - *For a given feature map, we show the top 9 activations, each projected separately down to pixel space, revealing the different structures that excite that map and showing its invariance to input deformations. Alongside these visualizations we show the corresponding image patches*

VISUALISING INTERMEDIARY LAYERS



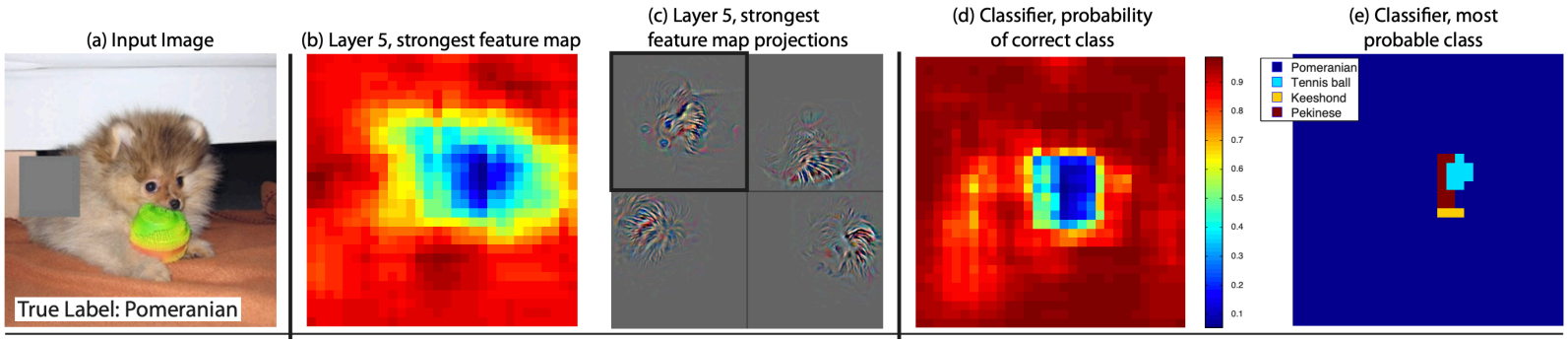
VISUALISING INTERMEDIARY LAYERS



Layer 4

VISUALISING INTERMEDIARY LAYERS

- Occlusion Sensitivity:
  - *With image classification approaches, a natural question is if the model is truly identifying the location of the object in the image, or just using the surrounding context. Fig. 6 attempts to answer this question by systematically occluding different portions of the input image with a grey square, and monitoring the output of the classifier. The examples clearly show the model is localizing the objects within the scene, as the probability of the correct class drops significantly when the object is occluded.*



When the dog's face is obscured, the probability for "pomeranian" drops significantly. …(e)….but if the dog's face is obscured but not the ball, then it predicts "tennis ball"

VISUALISING INTERMEDIARY LAYERS

Try it yourself ☺

- https://towardsdatascience.com/how-to-visualize-convolutional-features-in-40-lines-of-code-70b7d87b0030
- https://towardsdatascience.com/understanding-your-convolution-network-with-visualizations-a4883441533b
- https://towardsdatascience.com/https-medium-com-rishabh-grg-the-ultimate-nanobook-to-understand-deep-learning-based-image-classifier-33f43fea8327
- https://tensorspace.org/index.html

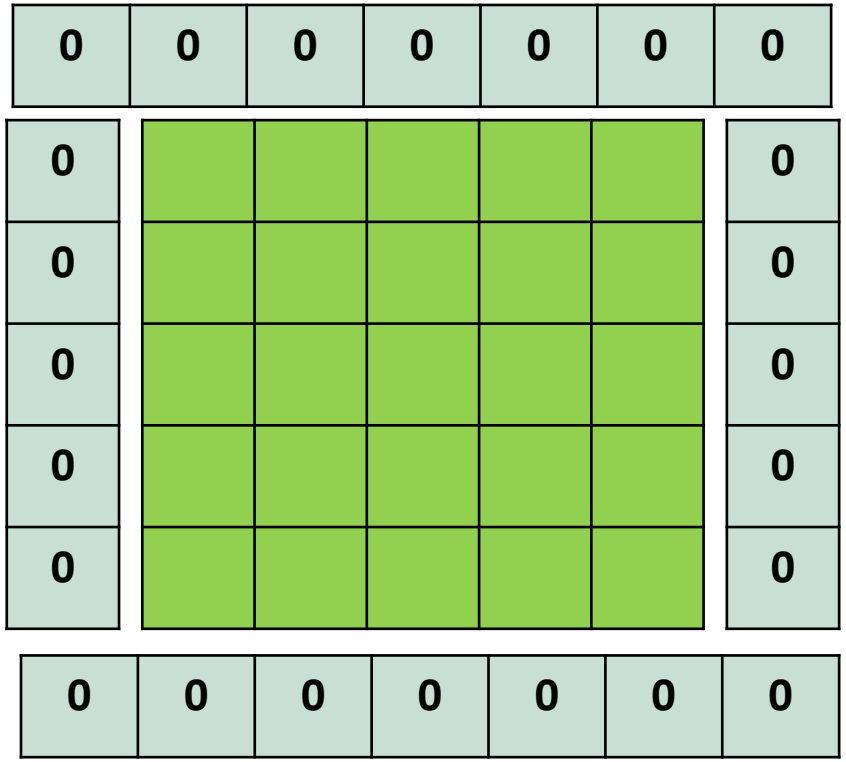
CONVOLUTION

- We saw that the filter/kernel slides along the input. We call this the **step size**
  - Typically 1 or 2.
- If this won't fit nicely we use **zero padding**
  - This technique also assists us to not lose information at the edges of our image.
- The formula to work out our output size is:

$$n_{out} = \left\lfloor \frac{n_{in} + 2p - f}{s} + 1 \right\rfloor$$

CONVOLUTION EXAMPLE

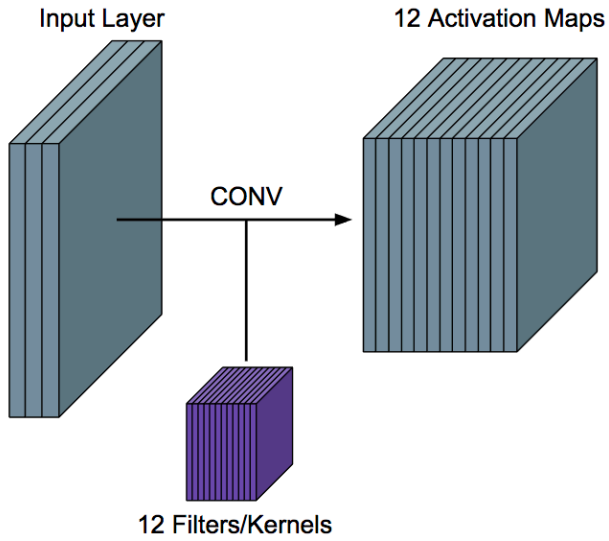
$$n_{out} = \left\lfloor \frac{n_{in} + 2p - f}{s} + 1 \right\rfloor$$



- With no padding, stride 2, filter 3*3:
  - 5/2 + 1
  - That's not nice…
- Let's add some zero padding:
  - (5 + 2 – 3) / 2 + 1 = 3
  - Much better!

A SUMMARY OF THE MATHS:

- We have the following:
  - K filters of F*F size
  - S stride
  - P padding
  - W1 * H1 * D1 input
    - Let's assume W1=H1
- Out output is:
  - $\left(\frac{W_1+2P-F}{S}+1, \frac{W_1+2P-F}{S}+1\right), K$



- Let's say 32*32*3 image, 12 filters (5*5), 1 Padding, Stride 2
- Out output is:
  - $\left(\frac{32+2-12}{2}+1, \frac{32+2-12}{2}+1\right), 12 = 12 * 12 * 12$

CONVOLUTIONS & VANILLA NN?

- The 'Z' of any given layer, L, can be found with the following formula:
  - $Z^{[L]} = W^{[L]}.A^{[L-1]} + b^{[L]}$
  - **W is now our filters**
- And the 'A' (activation matrix) of any given layer, L, is found by:
  - $A^{[L]} = g^{[L]}\left(Z^{[L]}\right)$
  - **This is the same**

STANDARD SIZES:

- Typical Stride Values:
  - 1 or 2 (more often 2) typically.
  - Larger strides will reduce network (information loss) faster

- Typical Kernel/Filter Sizes:
  - 3, 5, 7

- Typical numbers of filters:
  - 128, 256, 512…
    - Different architectures different approaches. VGG doubled each conv layer.

BACK TO LENET

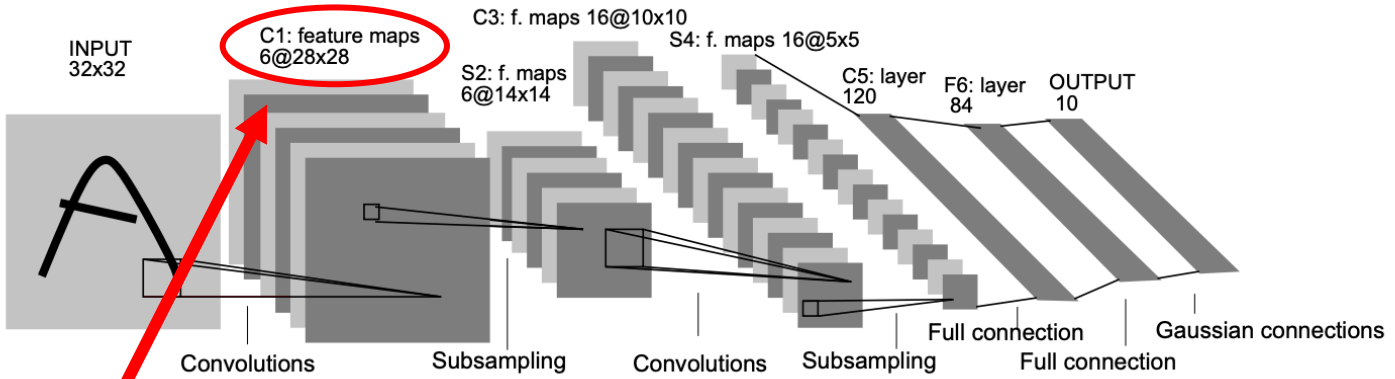


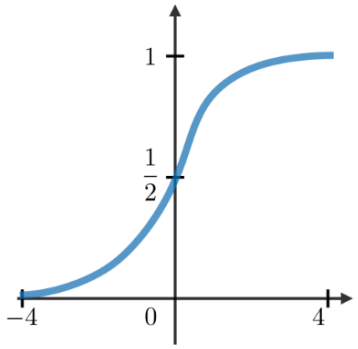
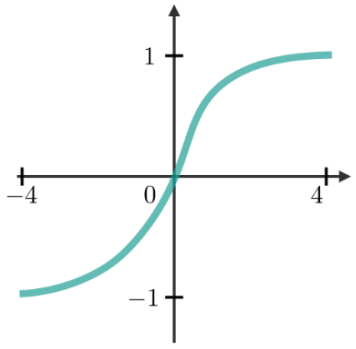
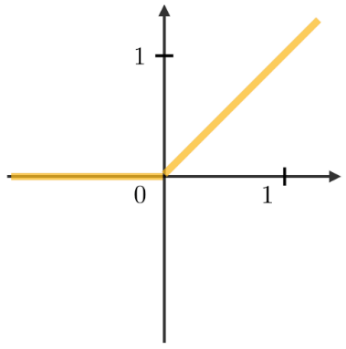
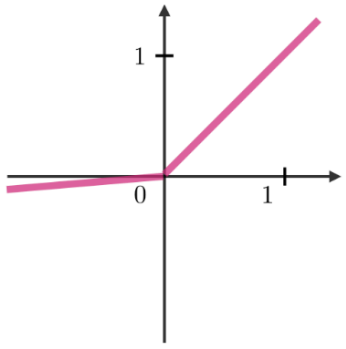
Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

- Input 32*32
  - 6 filters of 5*5, no padding, stride 1.
  - $\left(\frac{W_1+2P-F}{S}+1, \frac{W_1+2P-F}{S}+1\right), K$
  - $\left(\frac{32+0-5}{1}+1, \frac{32+0-5}{1}+1\right), 6$ = (28*28*6)

Fun Fact: Technically there was padding as digits were maximum 20*20

ACTIVATIONS

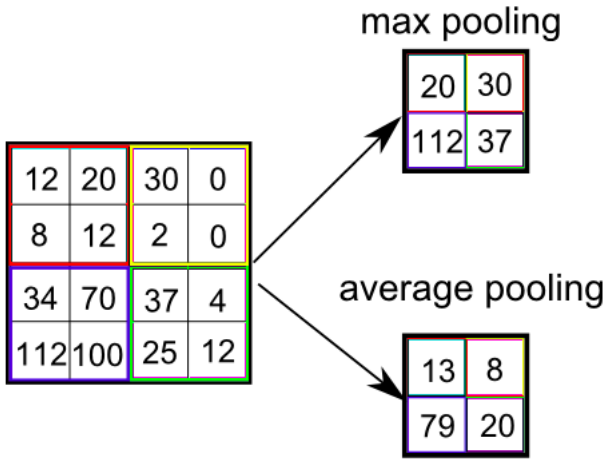
- Follow the convolution layer
  - Which makes sense considering our Vanilla NN flow

Sigmoid	Tanh	ReLU	Leaky ReLU
$g(z) = \frac{1}{1+e^{-z}}$	$g(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	$g(z) = \max(0, z)$	$g(z) = \max(\epsilon z, z)$ with $\epsilon \ll 1$
			

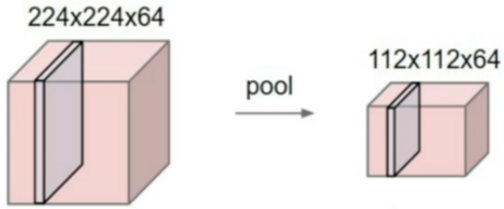
https://stanford.edu/~shervine/teaching/cs-229/cheatsheet-deep-learning

POOLING LAYER

- This is an operation to reduce the number of parameters (Down Sampling) to reduce parameters and compress representations.
  - We **down sample spatially**, not in the depth channel



- Max pooling is the most popular method where you take the maximum value in the region.



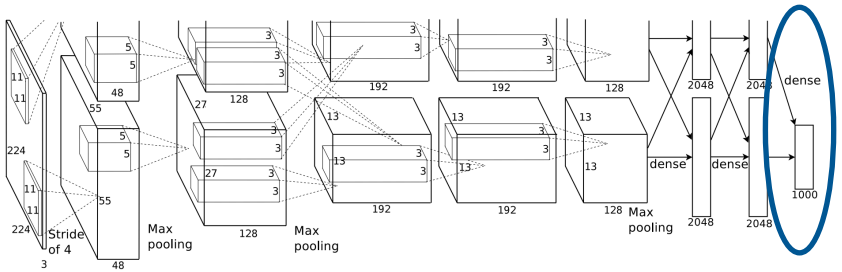
- We must also select a **Filter** and **Stride size** for this. (Typically 2/2 or 3/2)

FULLY CONNECTED (DENSE) LAYER

- Note that Yann LeCun has famously noted there is no such thing as a fully connected layer just
  - *"convolution layers with 1x1 convolution kernels and a full connection table"*

- In essence, we 'stretch out' our matrix to achieve a **1-dimensional vector output**. It can be thought of as the 'classification' component of our model.

  - Alexnet = last layer of size 1000
  - LeNet = last layer of size 10



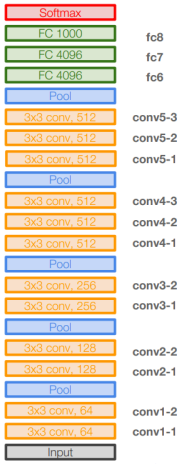
FULLY CONNECTED (DENSE) LAYER

- It is 'fully connected' since each input is present (due to matrix multiplication) in each 'number' (neuron) of the output.
  - You can add more FC layers to enhance the ability to combine complex patterns from your earlier feature maps.
- Careful though, FC layers are immensely computationally expensive!
  - (Source)

(not counting biases)

INPUT: [224x224x3] memory: 224*224*3=150K params: 0
CONV3-64: [224x224x64] memory: 224*224*64=3.2M params: (3*3*3)*64 = 1,728
CONV3-64: [224x224x64] memory: 224*224*64=3.2M params: (3*3*64)*64 = 36,864
POOL2: [112x112x64] memory: 112*112*64=800K params: 0
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*64)*128 = 73,728
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*128)*128 = 147,456
POOL2: [56x56x128] memory: 56*56*128=400K params: 0
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*128)*256 = 294,912
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
POOL2: [28x28x256] memory: 28*28*256=200K params: 0
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*256)*512 = 1,179,648
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
POOL2: [14x14x512] memory: 14*14*512=100K params: 0
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
POOL2: [7x7x512] memory: 7*7*512=25K params: 0
FC: [1x1x4096] memory: 4096 params: 7*7*512*4096 = 102,760,448
FC: [1x1x4096] memory: 4096 params: 4096*4096 = 16,777,216
FC: [1x1x1000] memory: 1000 params: 4096*1000 = 4,096,000

TOTAL memory: 24M * 4 bytes ~= 96MB / image (only forward! ~*2 for bwd)
TOTAL params: 138M parameters



VGG16

Common names

138M Parameters

122M are in FC layers

EXTRA READINGS & SOURCES

- https://towardsdatascience.com/https-medium-com-rishabh-grg-the-ultimate-nanobook-to-understand-deep-learning-based-image-classifier-33f43fea8327
- https://towardsdatascience.com/gentle-dive-into-math-behind-convolutional-neural-networks-79a07dd44cf9
- https://pythonmachinelearning.pro/introduction-to-convolutional-neural-networks-for-vision-tasks/

REGULARISATION FOR CNN

- What we already know:
  - L2 still used
  - Dropout more common in fully connected
    - Some people don't like it at all (here)

- Keras ref (here)

- A few new ones:
  - Maxnorm
  - Batch Normalisation
  - Data Augmentation

MAX NORM

- From Keras Documentation (link):
  - *Constrains the weights incident to each hidden unit to have a norm less than or equal to a desired value.*
- Essentially creating an upper bound for the magnitude of the weight vector for every neuron.
- A nice tutorial in Keras adding this and other constraints (link)

MAX NORM

- From the dropout paper (link):

One particular form of regularization was found to be especially useful for dropout—constraining the norm of the incoming weight vector at each hidden unit to be upper bounded by a fixed constant c. In other words, if $\mathbf{w}$ represents the vector of weights incident on any hidden unit, the neural network was optimized under the constraint $||\mathbf{w}||_2 \leq c$. This constraint was imposed during optimization by projecting $\mathbf{w}$ onto the surface of a ball of radius c, whenever $\mathbf{w}$ went out of it. This is also called max-norm regularization since it implies that the maximum value that the norm of any weight can take is c. The constant c is a tunable hyperparameter, which is determined using a validation set. Max-norm regularization has been previously used in the context of collaborative filtering (Srebro and Shraibman, 2005). It typically improves the performance of stochastic gradient descent training of deep neural nets, even when no dropout is used.

Although dropout alone gives significant improvements, using dropout along with max-norm regularization, large decaying learning rates and high momentum provides a significant boost over just using dropout.

BATCH NORMALISATION

- (Source) Loffe & Szegedy, 2015
- Essentially force a unit-gaussian distribution on the inputs at each layer (for each neuron).
  - Usually inserted after FC/Conv layers (before activation)
- Nice Article (here)
- Our process:
  - Calculate minibatch mean (subtract from output)
  - Divide by SD of the minibatch

$$\hat{\boldsymbol{x}}^{(\boldsymbol{k})} = \frac{\boldsymbol{x^k} - \boldsymbol{\mu}[\boldsymbol{x^k}]}{\sqrt{\boldsymbol{Var}[\boldsymbol{x^k}]}}$$

- We actually have some hyperparameters Scaling (**Gamma**) and Shift (**beta**) to allow layers to determine *how much unit-gaussian-ness* is necessary

$$y^{(k)} = \boldsymbol{\gamma}^{(\boldsymbol{k})}\hat{x}^{(k)} + \boldsymbol{\beta}^{(\boldsymbol{k})}$$

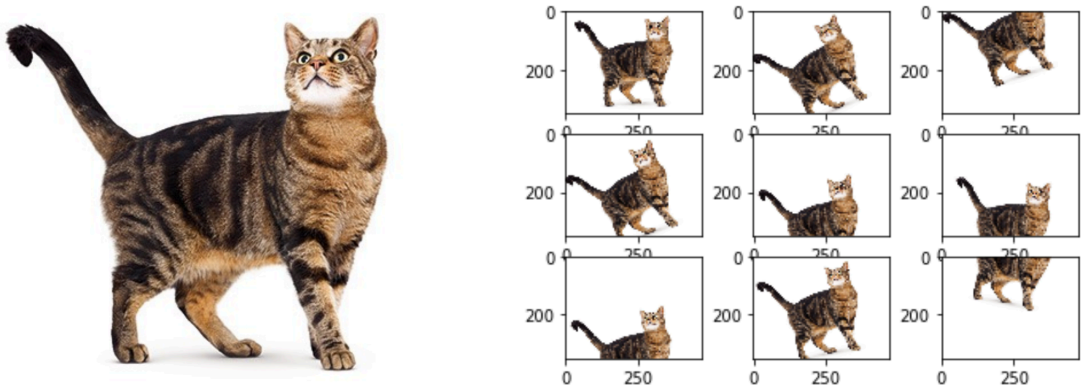
BATCH NORMALISATION

- Why? Some advantages:
  - Improves gradient flow
  - Alleviates issues with bad initialisation
  - Allows larger learning rates to be used
  - Effectively regularises
    - Why? See answer by Ian Goodfellow here

- In Keras?
  - Simply add as another layer. A nice stackoverflow answer here
  - The Keras documentation page (here)
  - A nice walkthrough example (here)

DATA AUGMENTATION

- Chop, change, flip, crop your images
  - Produces more data (Deep learning loves more data)
  - Helps with image difficulties (occlusion, variation etc)
- Keras provides a nice ImageDataGenerator class to assist



Original Image and Augmented Images

Source

BACK TO ALEXNET

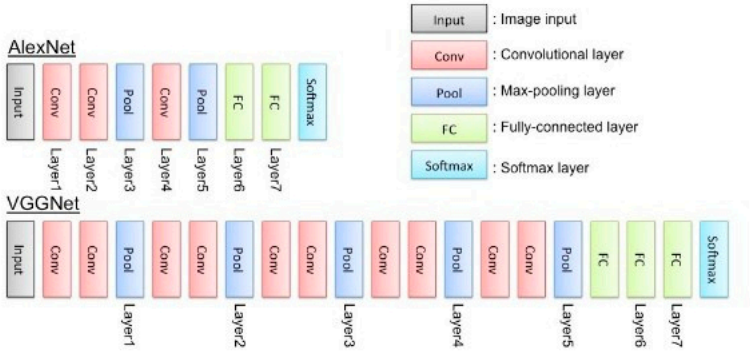
- Other Architectural choices you will be familiar with:
  - ReLu activations
  - Dropout (0.5)
  - Data Augmentation
  - Minibatch size 128
  - Learning rate $1e^{-1}$
  - L2 Weight Decay (Regularisation) $5e^{-4}$
  - Ensembled 7 CNNs and averaged
- Diagram appears ‘split’ as training had to occur on 2GPUs
- Some good Slides

VGGNET (16 AND 19)

- VGG:
  - (Source) From the Visual Geometry Group at Oxford.

- Some notable features:
  - Had more layers (19) but smaller filters (3*3 rather than 7*7 or larger) than AlexNet
  - Before batch normalisation so difficult to train network this size (trained for weeks)

AlexNet v. VGG



(Source)

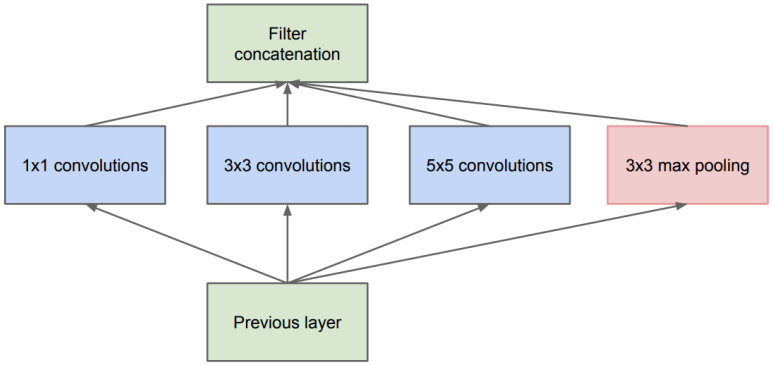
GOOGLENET ('INCEPTION')

- (Source) From….Google
- Also 19 layers but some more computational efficiency. (More in later versions)

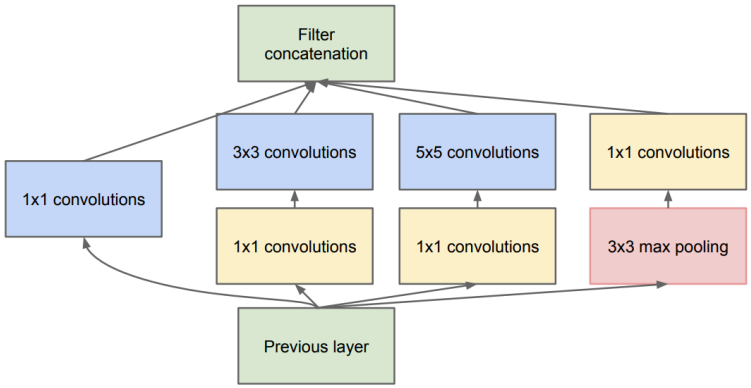
- Introduced the '**Inception Module**'

- Work on getting good **local structure** using 'a network within a network'

- In practice:
  - Apply several convolutions with different receptive field sizes **in parallel** and **concatenate depth-wise** (using some padding to help)



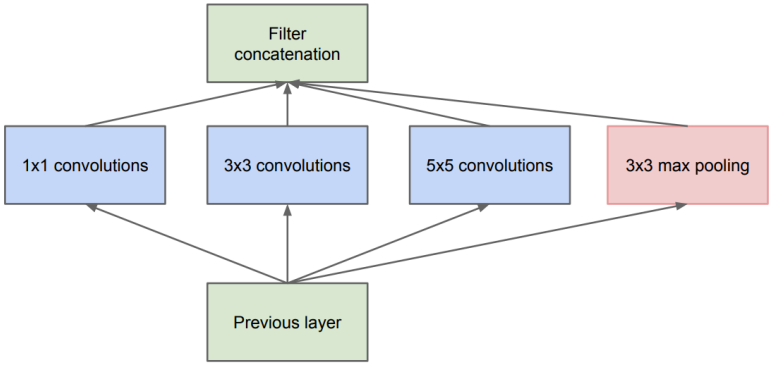
(a) Inception module, naïve version



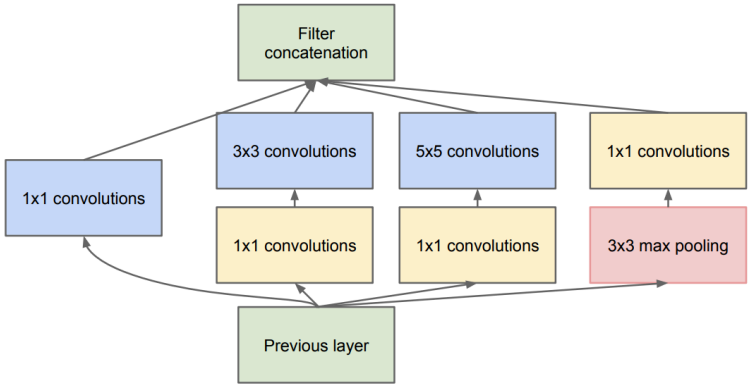
(b) Inception module with dimensionality reduction

GOOGLENET ('INCEPTION')

- **Problem**:
  - Huge computational complexity
- **Solution**:
  - Use 1*1 convolutions to bottleneck the layers.
  - (Layer-to-1*1 plus 1*1 to 5*5) is **cheaper** than direct layer-to-5*5

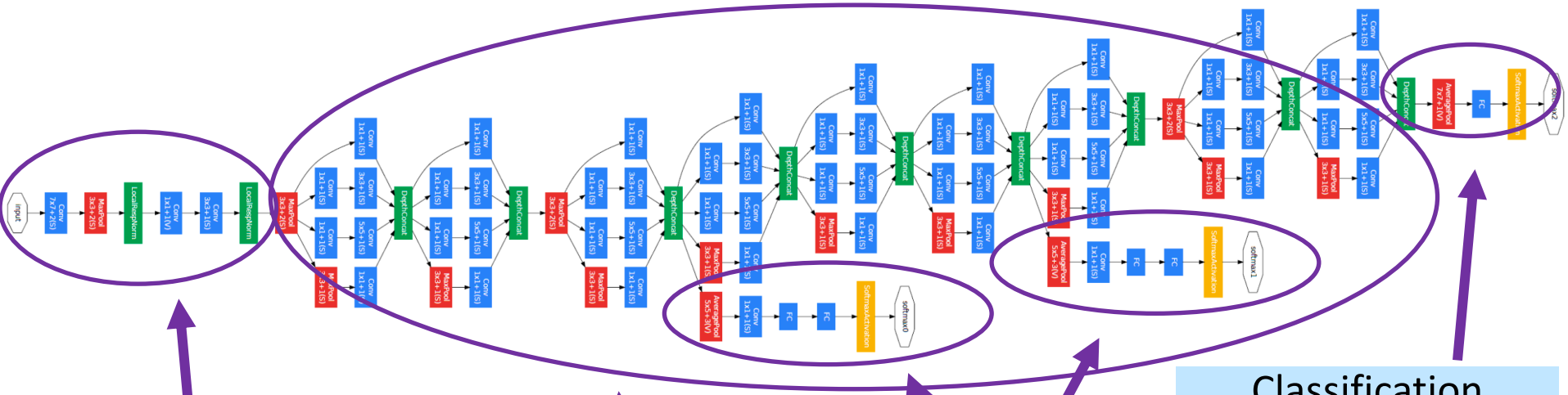


(a) Inception module, naïve version



(b) Inception module with dimensionality reduction

BACK TO THIS….



‘Vanilla’ CNN

Inception Modules

Auxilliary classification outputs

A mini-network (softmax and all) to inject gradient into initial layers of the network. Due to lack of batch-norm, this was used to ensure deeper networks would work.

Classification (Softmax) layer

RESNET & RESNEXT

- (Source) From Microsoft
- Recall: Alexnet was 7, VGG/Google was 19-22 layers.
- Vanishing gradient is stalling progress.

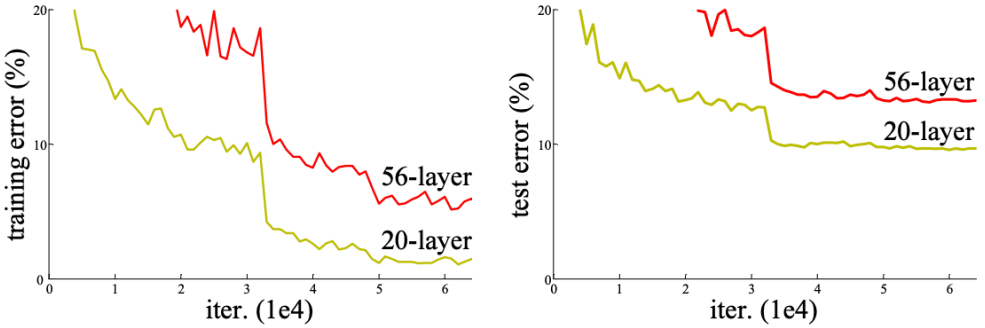


Figure 1. Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer "plain" networks. The deeper network has higher training error, and thus test error. Similar phenomena on ImageNet is presented in Fig. 4.

RESNET

- ResNet (Deep residual learning network) was **156 layers** – how??
- Introducing the **residual learning block**

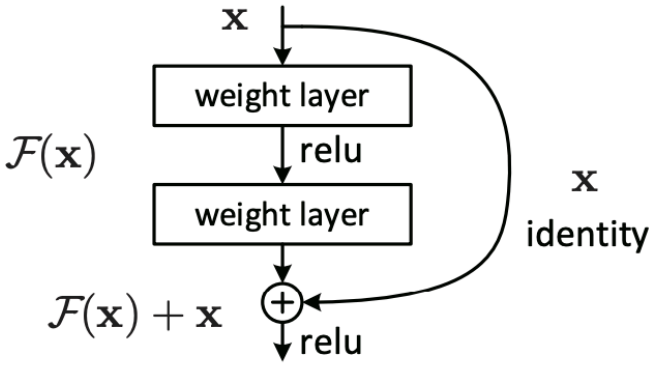


Figure 2. Residual learning: a building block.

- In essence it allows for a 'skip connection' around layers of the network (that have the same size).
  - A deeper network shouldn't produce higher error than a shallower unless it is being forced to learn things it doesn't want to.
- So the network can **decide not to use layers**
  - Assists gradient flow ('Gradient Highways' - Source)

RESNEXT & DENSENET

Nice blog (here)

- ResNext (source) took inspiration from Inception (*split-transform-merge*) and ResNet together.
  - Defined 'cardinality' which is the number of different paths offered to the network.
- Densenet (Source) kind of goes all out in terms of connections between layers

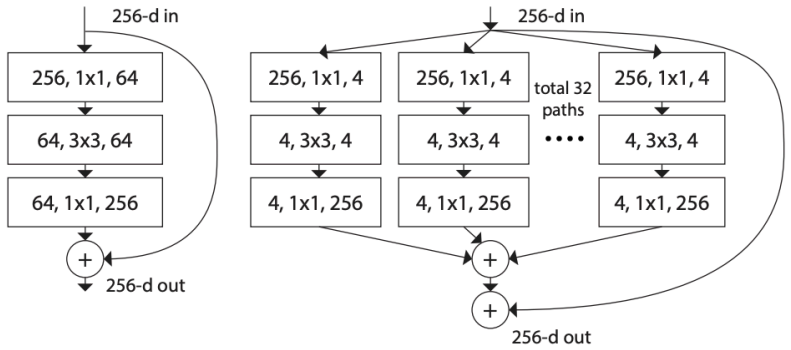
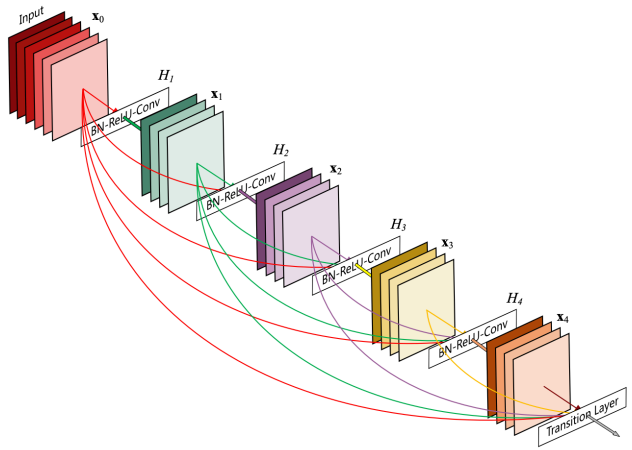
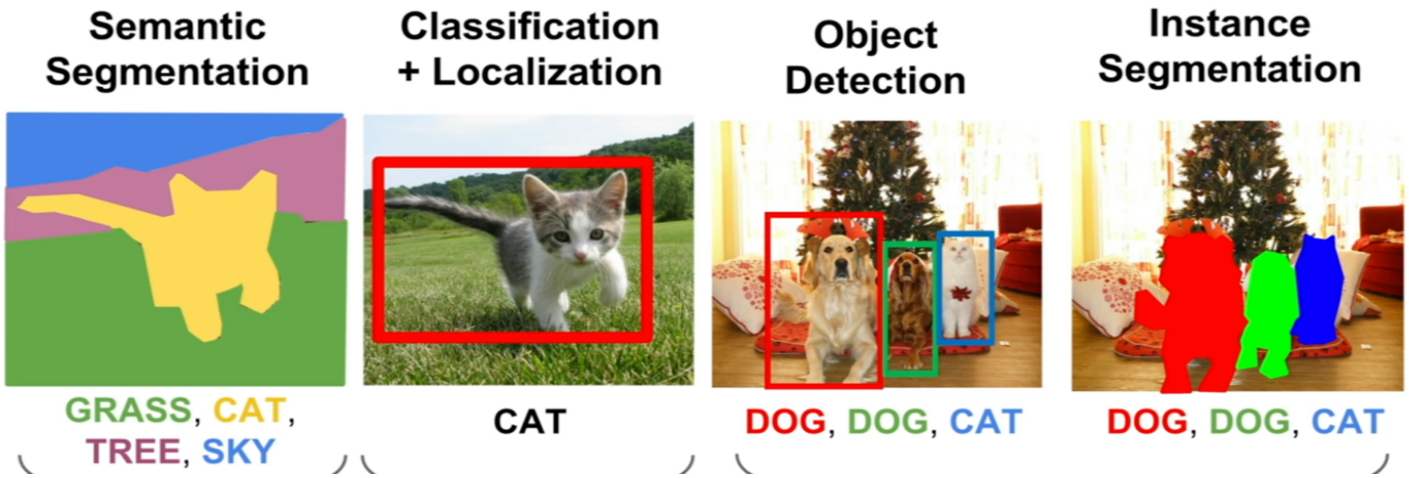


Figure 1. **Left**: A block of ResNet [14]. **Right**: A block of ResNeXt with cardinality = 32, with roughly the same complexity. A layer is shown as (# in channels, filter size, # out channels).



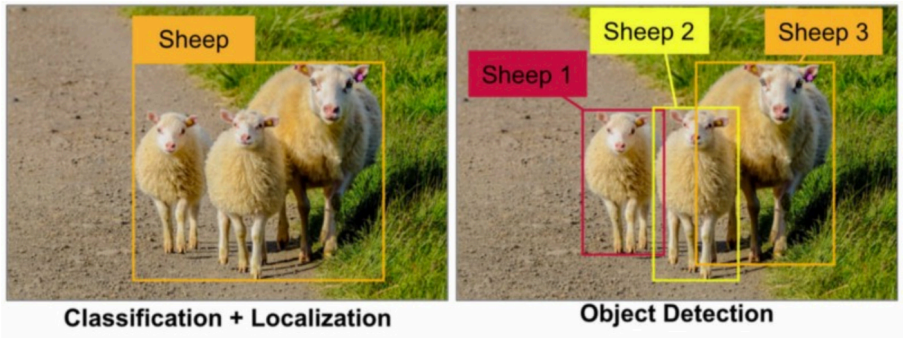
OTHER COMPUTER VISION TASKS



http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture11.pdf

BOUNDING BOXES (LOCALISATION)

- Here we not only care about *what* is in the image but we care about *where* it is in the image.
  - What would be our objective function for this?



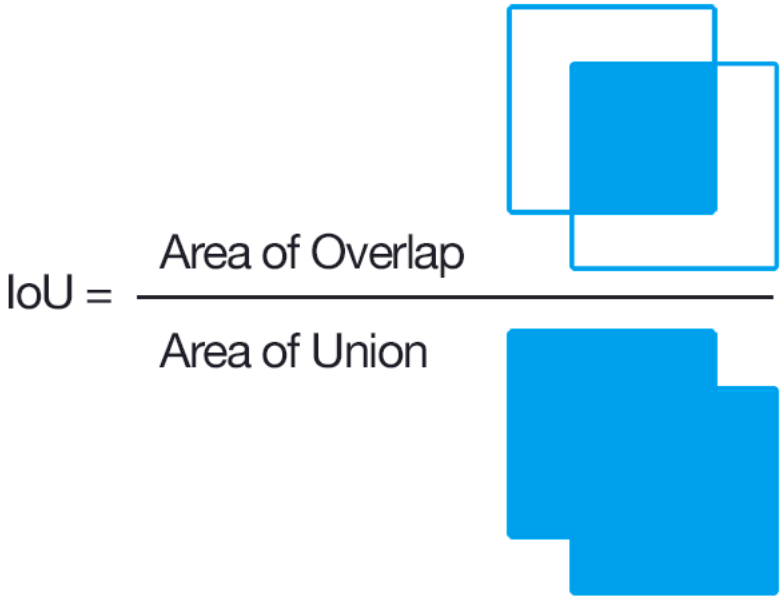
Source

We have two objective functions:

- A softmax (detection)
- Four geographical points for the box corners (Localisation)
  - (X, Y, W, H) [**Regression** Loss]

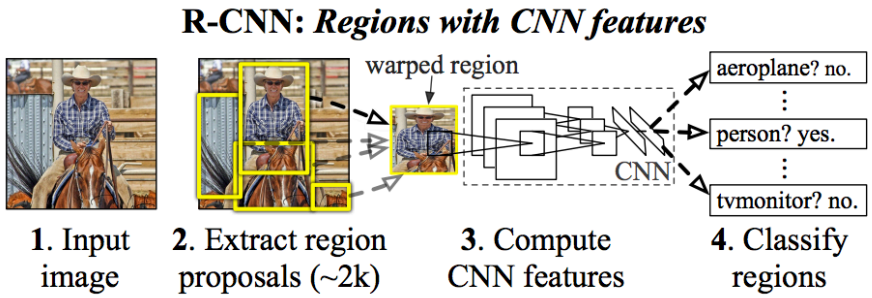
INTERSECTION OVER UNION

- A common metric to determine how close your bounding boxes are to the exact ones is **Intersection of Union (IOU)**
  - We will rarely get an exact match of bounding boxes, so this metric helps approximate our boxes.
  - Over 0.5 is pretty good.
  - A nice tutorial here



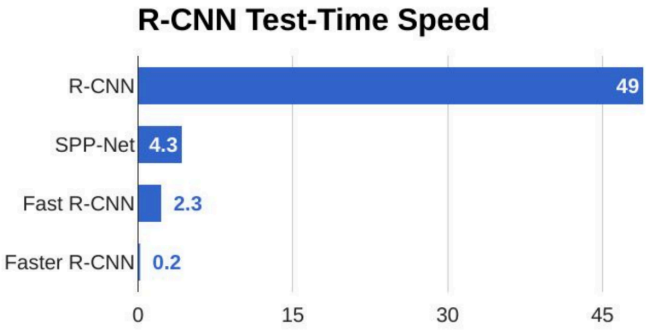
HOW DO WE DETERMINE WHERE TO PUT BOXES?

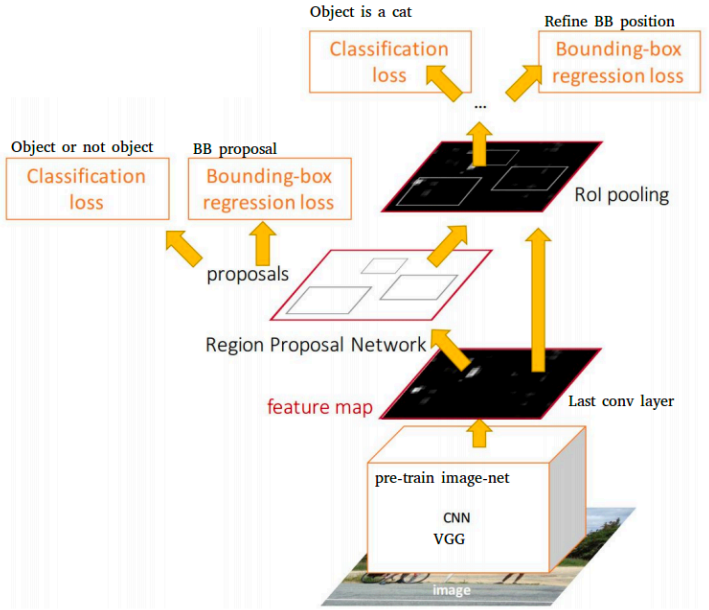
- Historically there was a proposed 'sliding window' approach where you slide many windows of different sizes over the image, each feeding through a CNN.
- There was great work done between 2013 and 2015/16 on region-based methods. (A nice youtube video here)
  - R-CNN (Source, 2014).
    - Generate 2000 'Region Proposals' (based on 'selective search'), warp and feed each into CNN to get features and classification (including 'background' as category).



HOW DO WE DETERMINE WHERE TO PUT BOXES?

- Fast R-CNN (Source, 2015)
  - Takes the region proposals from last feature map of convnet, sharing processing, 9x faster than R-CNN in training, hundreds of times faster in testing, better performance, combines losses.
- Fast-er R-CNN (Source, 2016)
  - The bottleneck with Fast R-CNN was the region proposal generation
  - Therefore has 'Region Proposal Network' that predicts proposals from features, has its own loss



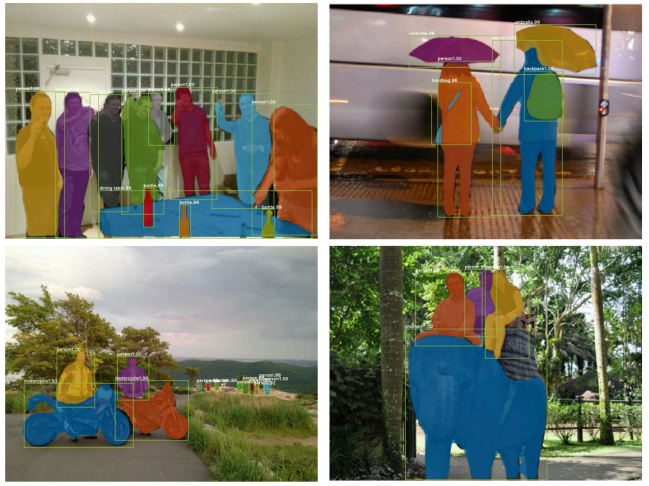


SSD & YOLO (REAL TIME BOUNDING BOXES)

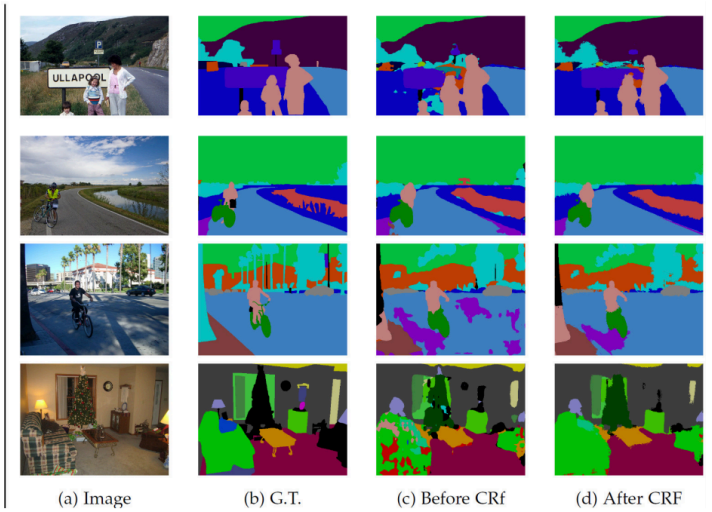
- SSD (Source, 2015-16) and Yolo (Source, 2015-16)
  - Yolo has released YoloV2, Yolo9000 (‘Better, Faster Stronger’)
    - Slides on Yolo9000 and author website papers
  - Latest version is YoloV3 (‘An incremental Improvement)
- Nice Explanation (here) and (here)
- More on v3 here
- A paper and blog on the trade off between these networks.
- Stanford slideshow on the different networks (here)

SEMANTIC/INSTANCE SEGMENTATION

- The state of the art for a while has been Mask R-CNN (Source)

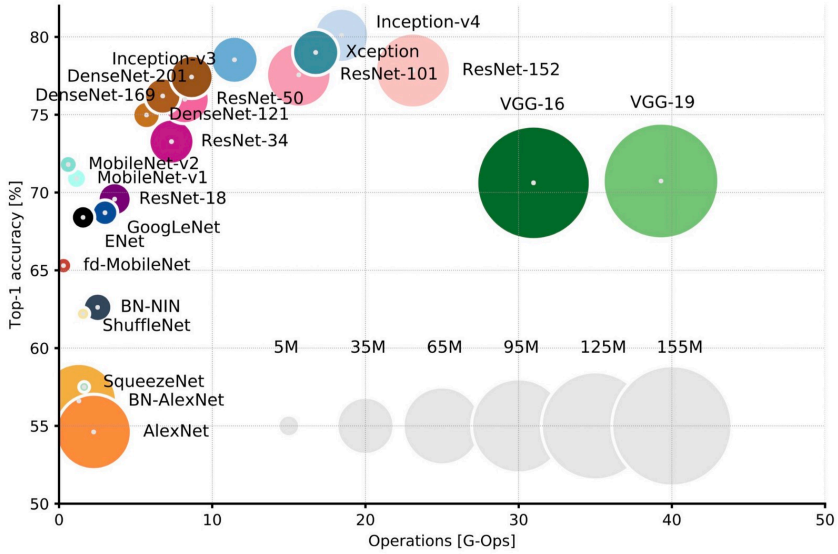


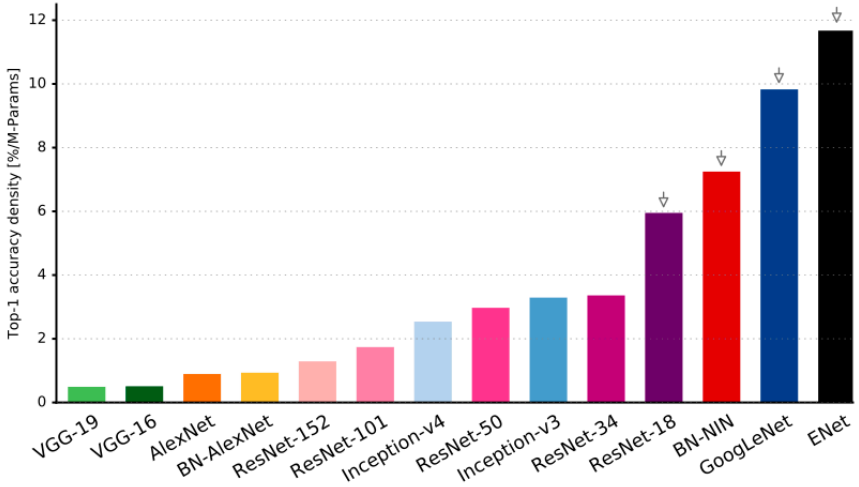
- There is also
  - U-Net (source) and in Keras (here)
  - DeepMask (Facebook) (Here)
  - Deeplab (google) (here / here)



WHICH ARCHITECTURE?

- A good paper (here)and (blog) compares computational power vs efficiency for a number of classic models.





- This is accuracy per million parameters

WHY START FROM SCRATCH?

- We know that some CNN architectures are great at extracting features (to the level of being able to classify objects) from images.
  - We can think of CNNs as feature extractors or data transformers.
- So can't we leverage some of the hard work already done with large training sets?
- This is the idea behind *transfer learning*

1. Train a neural network on a **large dataset**
2. Reinitialise and train (or add) the last FC layer
   > Perhaps multiple FC layers if you have the time/resources
3. Perhaps 'unfreeze' more layers with time/resources

LAB – COMPUTER VISION IN KERAS