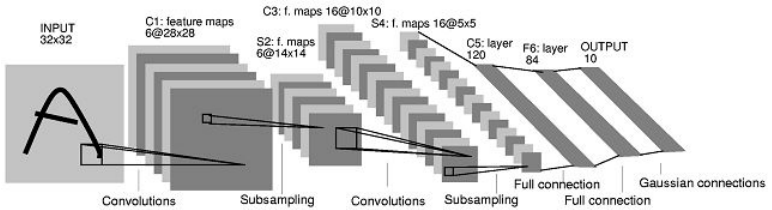
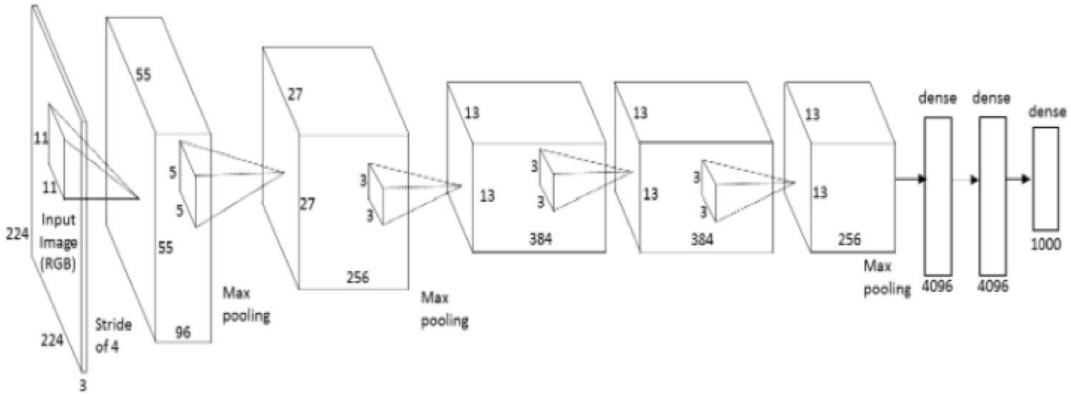
Applied Deep Learning

Day 2: Convolutional Neural Networks

Convolutional Neural Networks

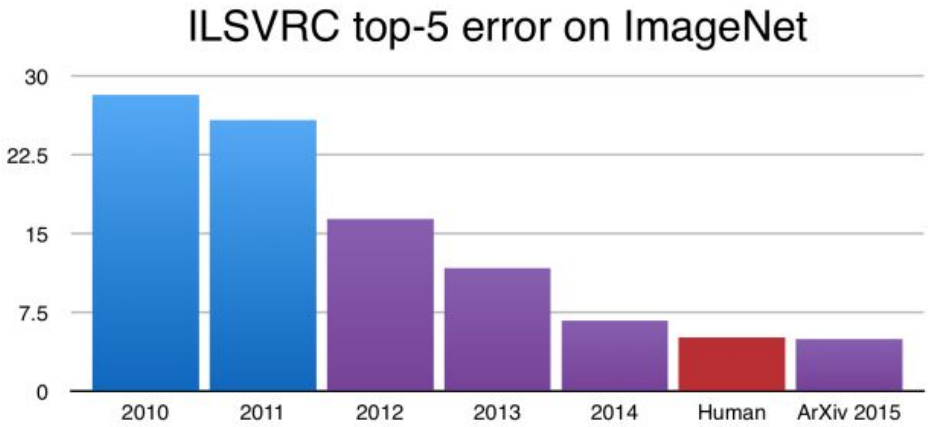
Convolutional neural networks are a special type of feedforward neural network that are particularly well suited for data which is spatially structured.





Motivation and context

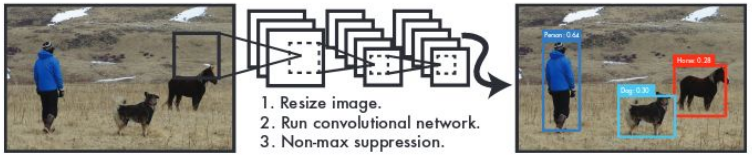
- Convolutional networks became popular mainly due to their success on images
  - ILSVRC is a 1000-way classification challenge on 1.2 million images
  - In 2012 deep learning was first applied to this dataset
  - In 2015 a convolutional network surpassed human level performance on this task



https://srconstantin.wordpress.com/2017/01/28/performance-trends-in-ai/

Motivation and context

Convolutional neural networks can also localize objects in image



Redmon et al., You Only Look Once: Unified, Real-Time Object Detection, https://pjreddie.com/media/files/papers/yolo.pdf

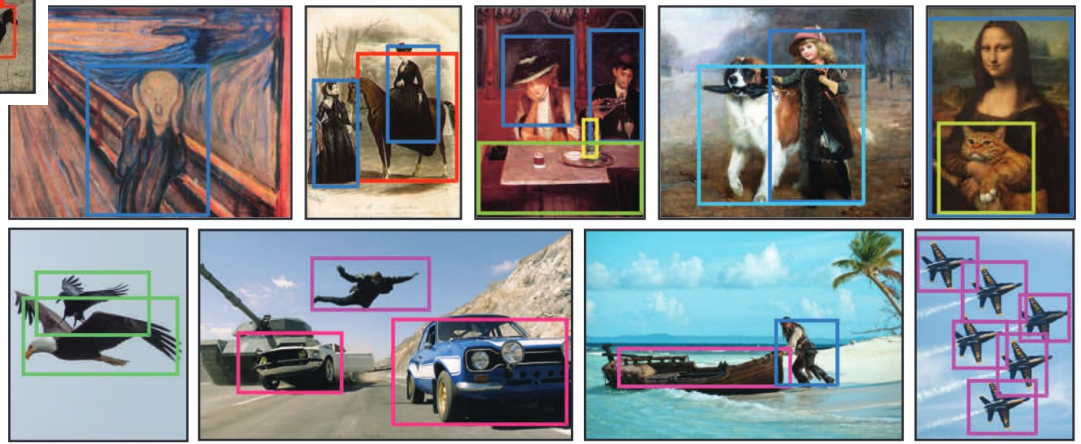


Figure 6: Qualitative Results. YOLO running on artwork and natural images. It is mostly accurate although it does think one person in an image is an airplane.

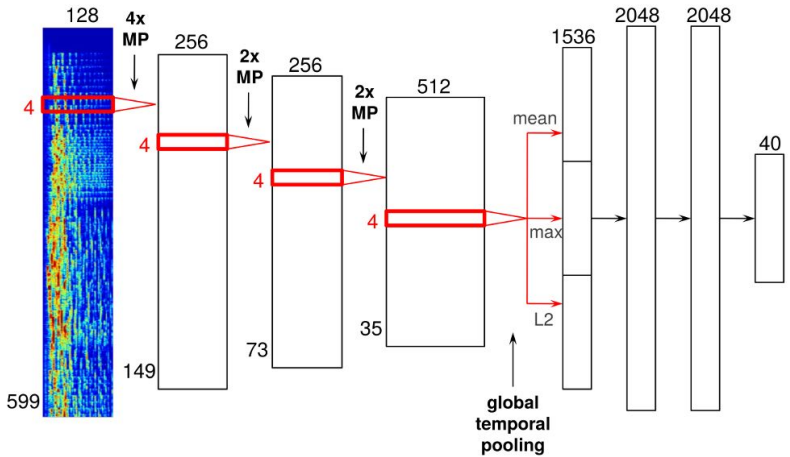
Motivation and context

We can create object masks and recognize that there can be different objects of the same type within an image

He et al., Mask R-CNN,
https://arxiv.org/pdf/1703.06870.pdf

Motivation and context

Convolutional neural networks are used in audio classification and transcription



http://benanne.github.io/2014/08/05/spotify-cnns.html

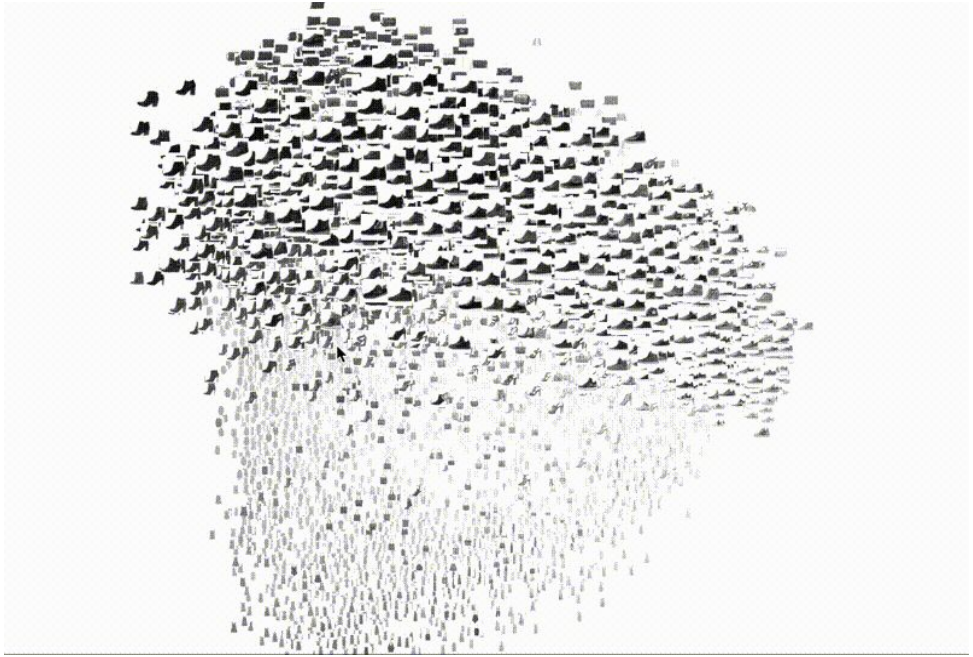
Motivation and context

Self driving cars

http://deepdrive.io/

Motivation and context

Image clustering and reverse image search

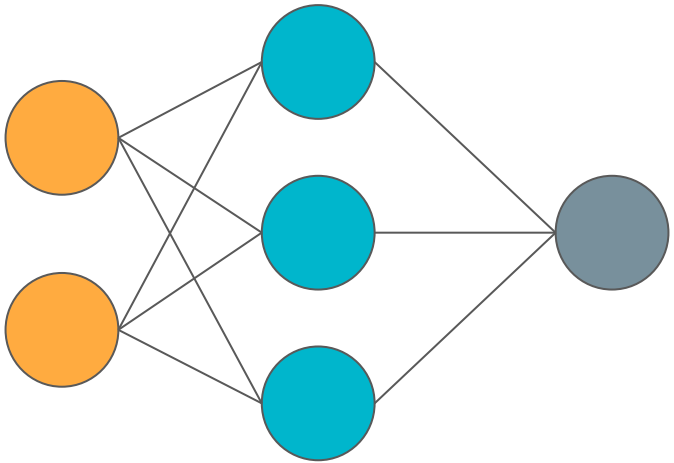


https://github.com/zalandoresearch/fashion-mnist

Convolutional Neural Network Operations

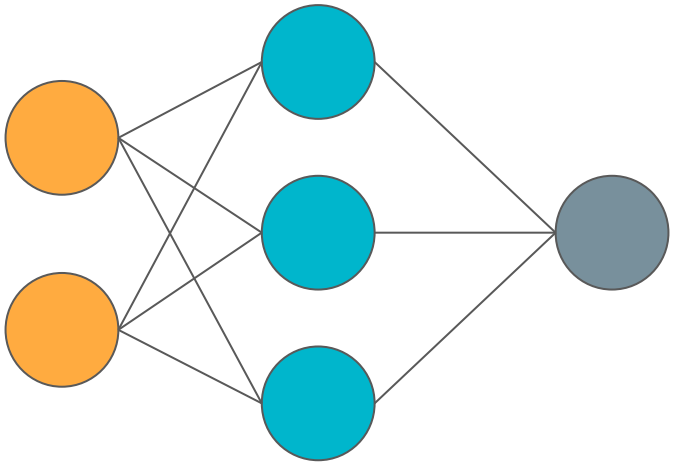
Convolutional Neural Networks

- Previously we used a linear combination of a weight matrix and input vector to perform our computations for each layer.
  - This has drawbacks when our data has some meaningful spatial structure
  - The network can not exploit the information that exists in the layout of the data



Convolutional Neural Networks

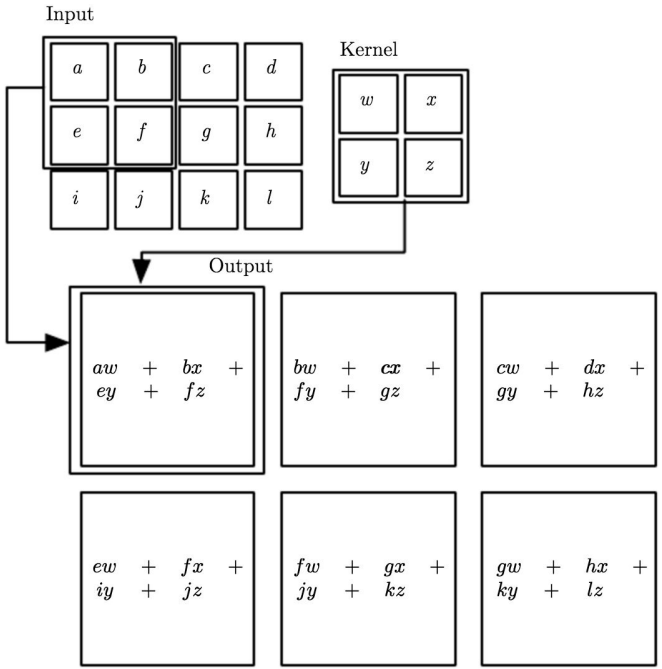
- Using this type of dense or fully connected network will not give us the maximum performance for spatially structured data
  - We need some way to exploit the way that local regions of data are related



Convolutional Neural Networks

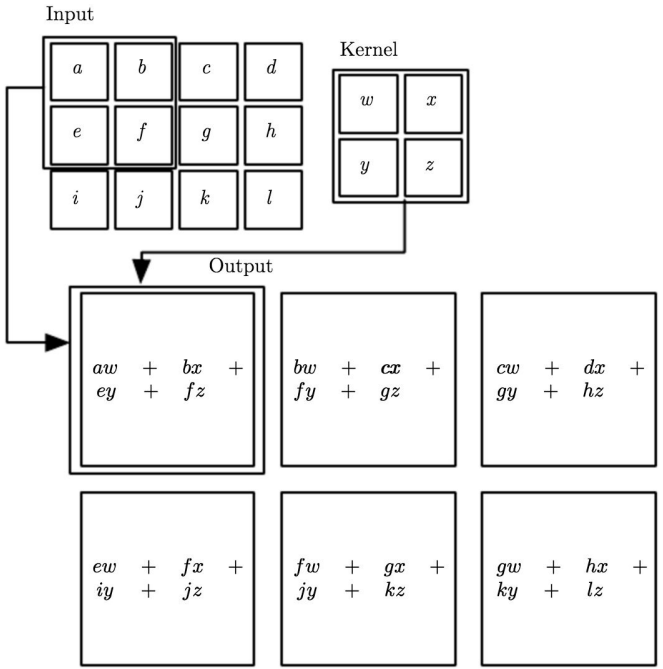
- We'll swap this dense operation for a type of sliding window operation that we will call a convolution
  - Set up a *knk* kernel to pass over the data
  - The kernel will perform calculations on small locally connected regions in our data
    - Making use of the relationship between data in local regions

Convolutional Neural Networks



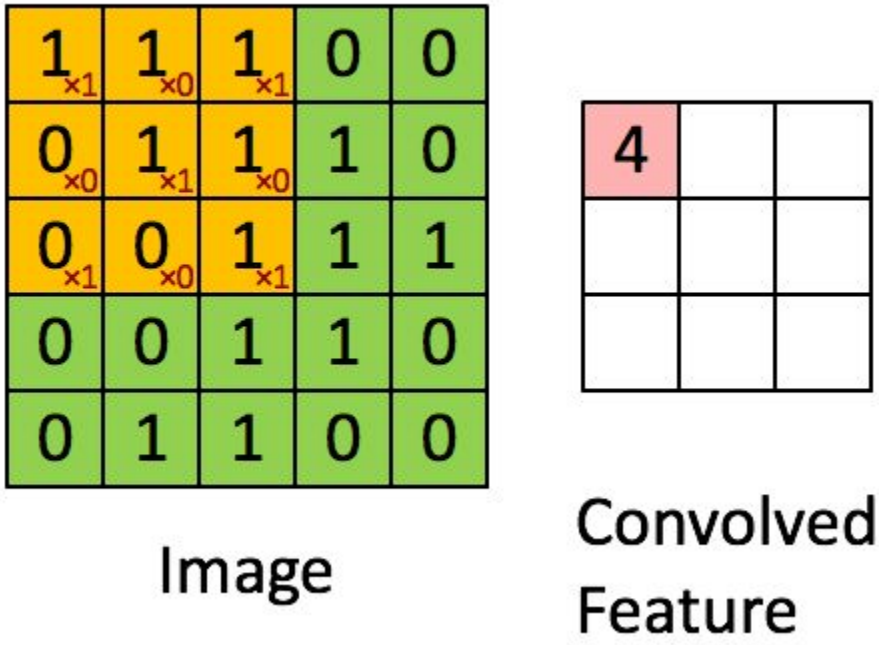
- The kernels are applied like a sliding window along the input data
- The output of the operation will be a single number
  - The sum of, each kernel weight multiplied by each corresponding data point
  - Multiplication is done element-wise

Convolutional Neural Networks



- With a kernel size of 2x2 and a data matrix of 3x4 we end up with a new 2x3 matrix
- We call this resulting matrix the ‘feature map’
    - These feature maps become the input to the following layer in our neural network

Convolutional Neural Networks

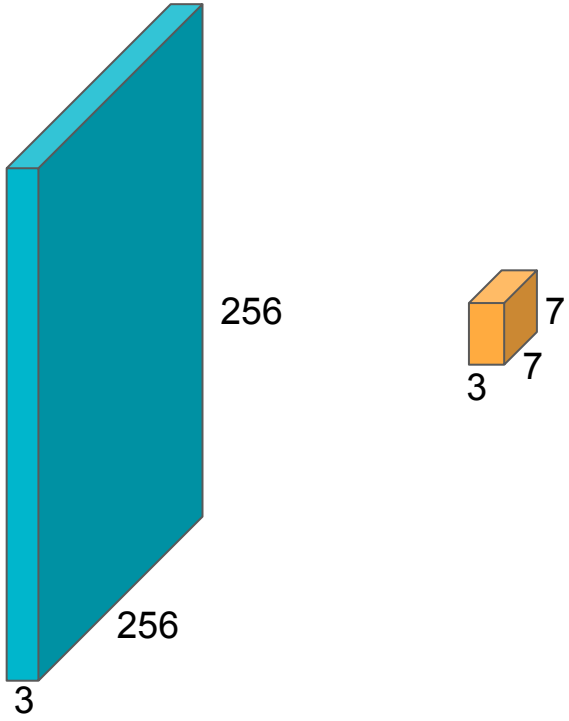


http://deeplearning.stanford.edu/tutorial/

Working with spatial data

- In dense or fully connected neural network we pass vectors around as the input data and representations used between the layers
- Convolutional neural networks work on tensors
  - An image is typically represented as a [*height* x *width* x *channels*] sized tensor
    - Channels are most often the representation of either red, green, or blue (RGB)
  - Each value in that tensor represents the intensity of a single pixel in some location and channel

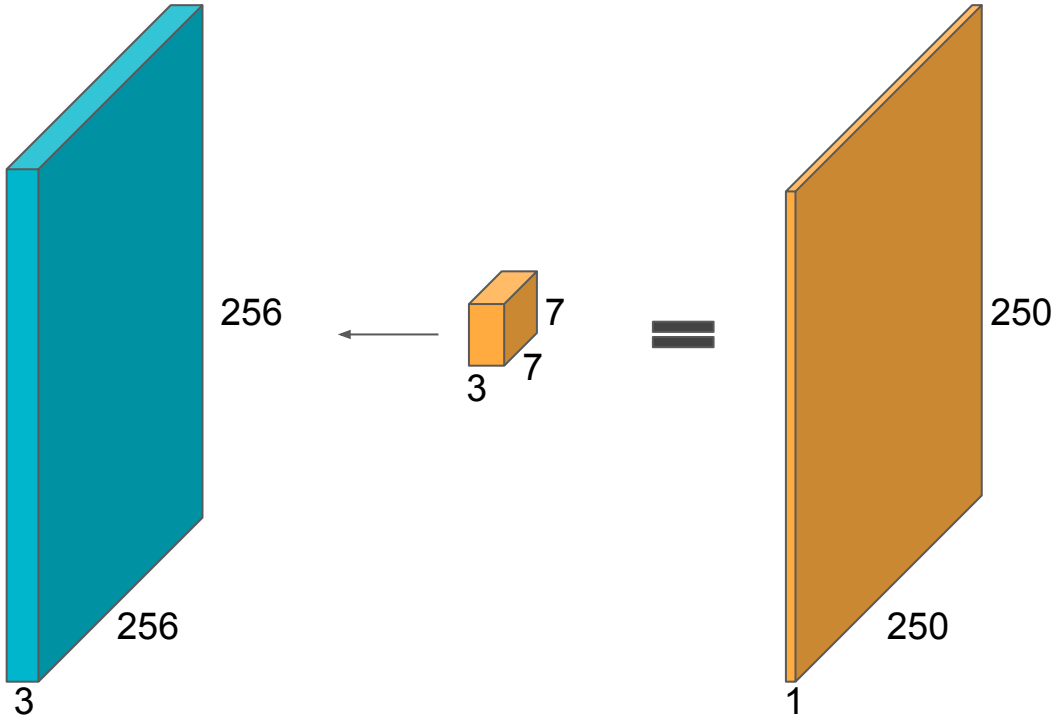
Working with spatial data



Kernels will still be defined height and width.

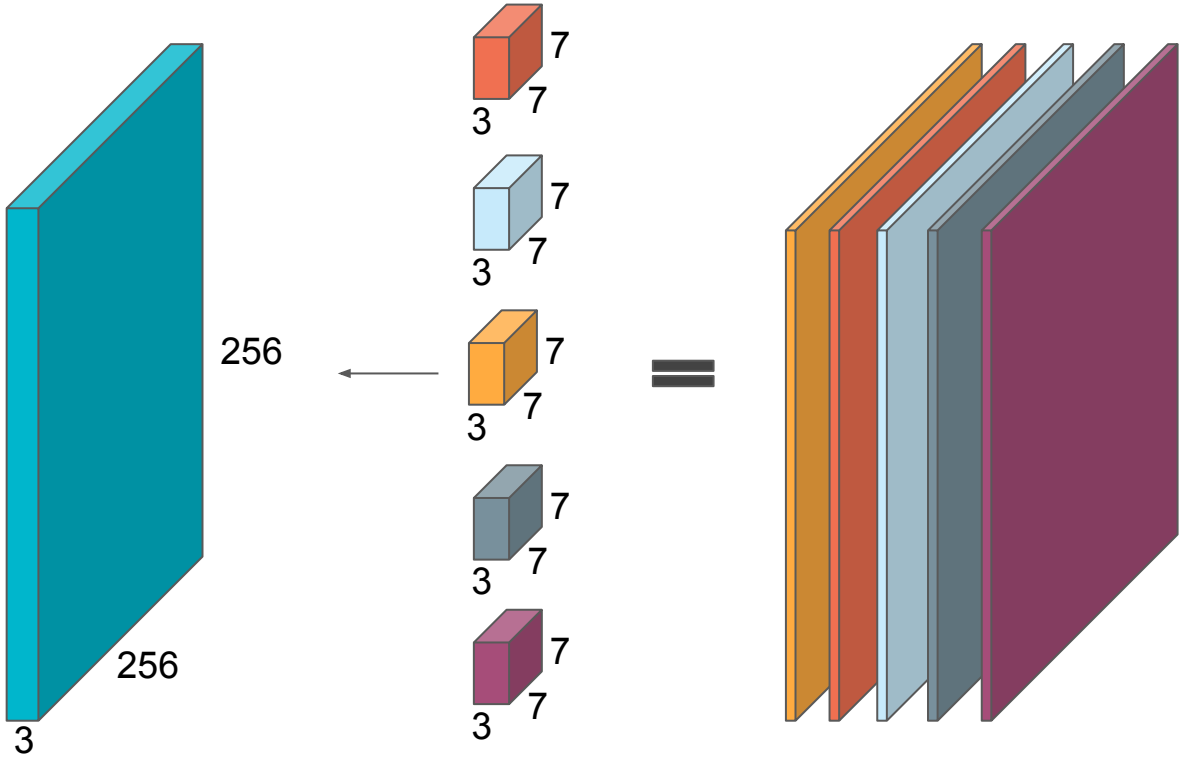
The third dimension of the kernel will be set to equal the depth dimension of whatever tensor it is applied to.

Working with spatial data



The output of this operation will be a feature map with a depth of 1.

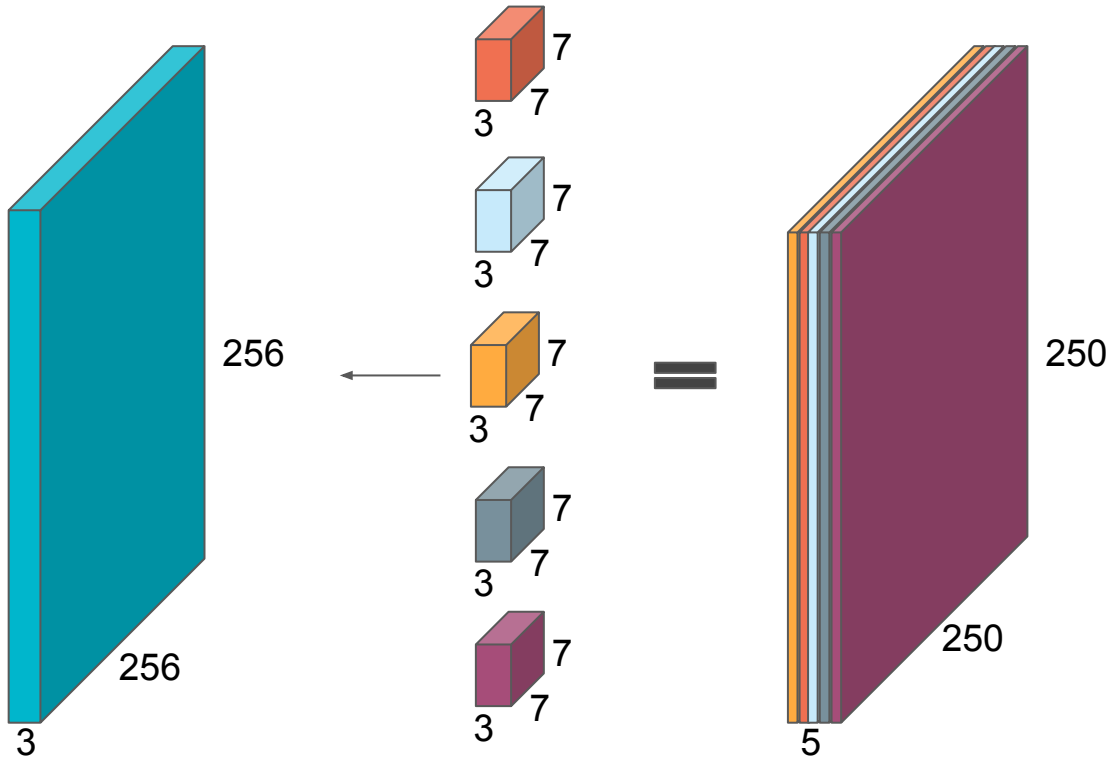
Working with spatial data



Typically there will be many kernels used in parallel to comprise a convolution layer

Each kernel will produce its own feature map

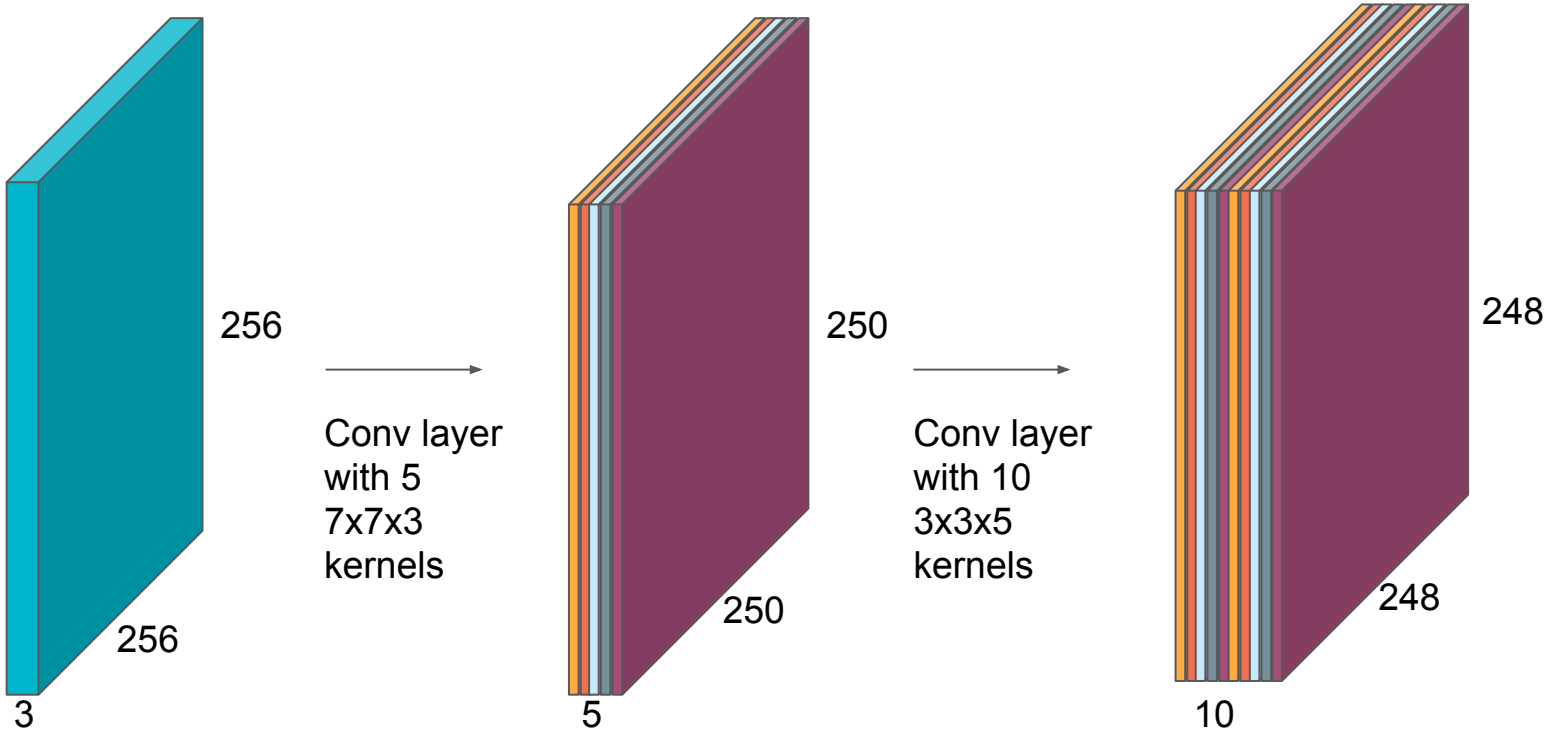
Working with spatial data



The resulting feature maps from each of the convolution operations are combined into a new tensor of depth 5

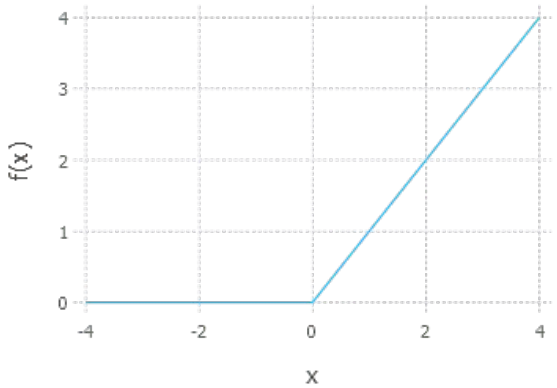
Which will serve as the input tensor to the following convolutional layer

Working with spatial data



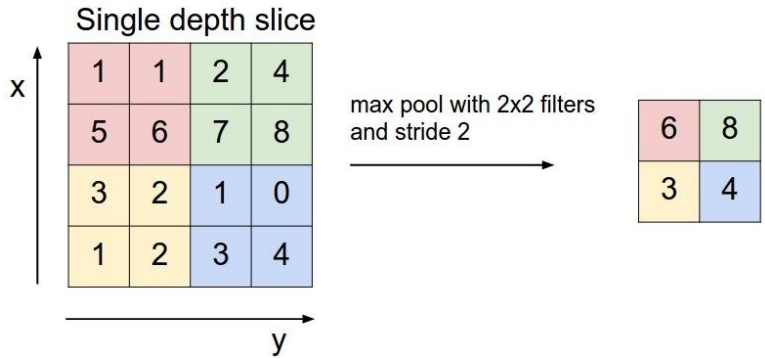
Working with spatial data

- After applying the kernel to some local region of the input data it is still necessary to use an activation function
  - Typically ReLU activations are used in convolution layers, max(x, 0)



Pooling Layers

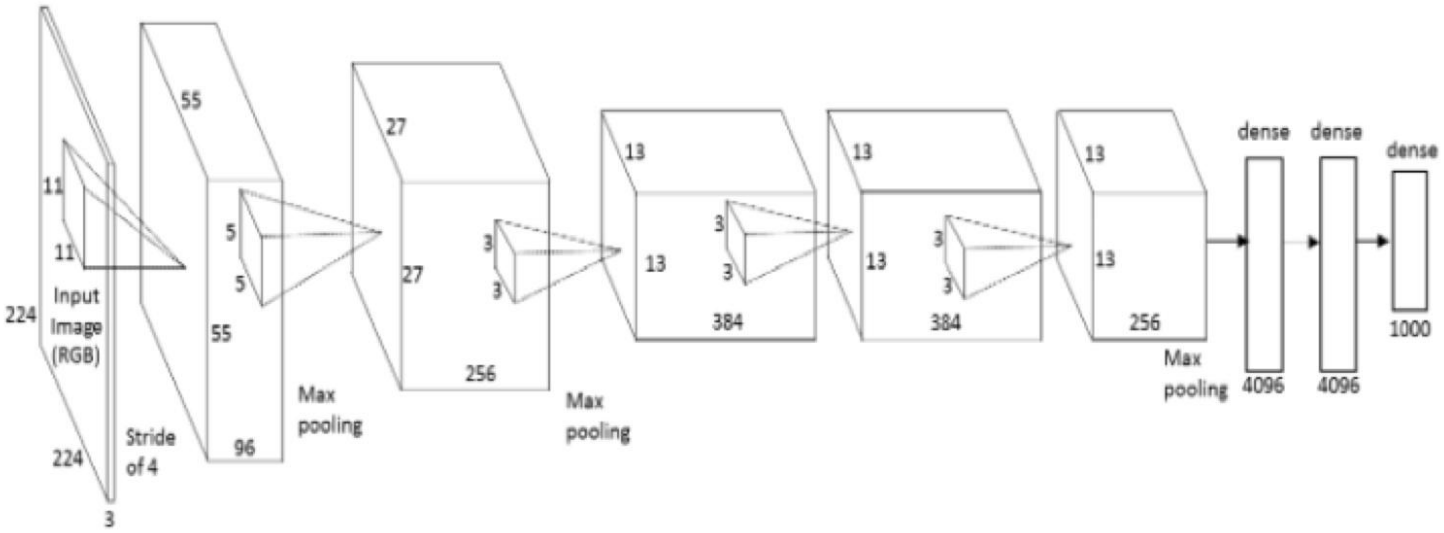
- Most commonly used is max pooling
  - occasionally we will see average pooling, but usually for very specific reasons towards the end of the network



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

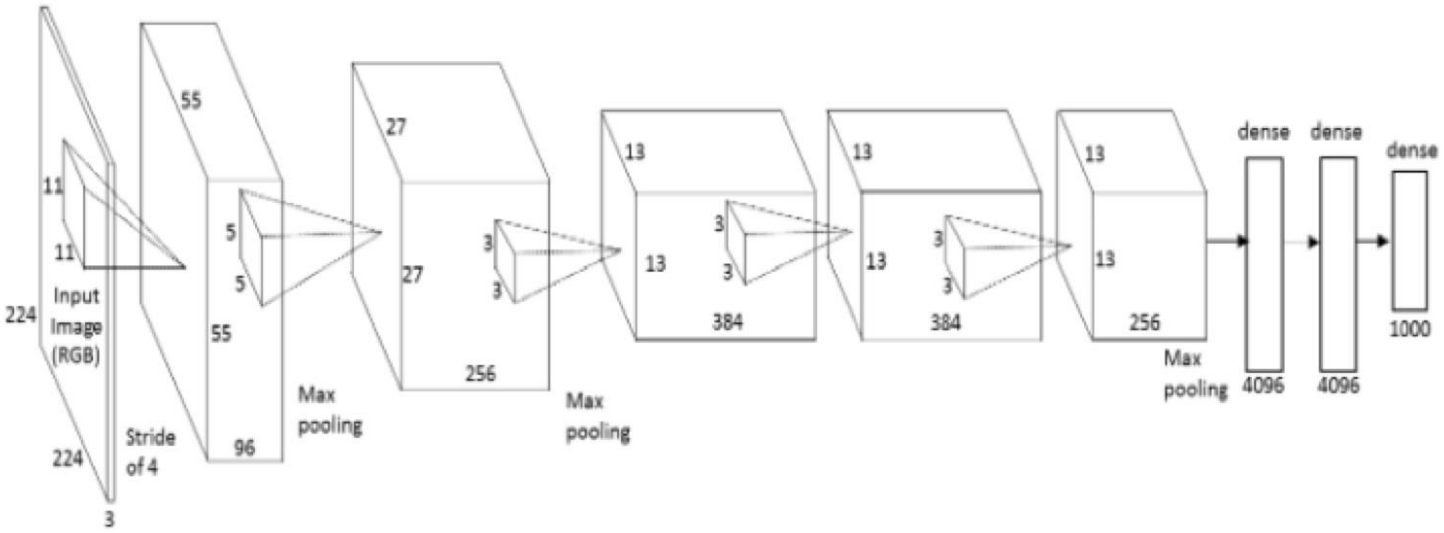
Putting a convnet together

- Typically we will see convolutional neural networks following a similar convention to the one below



Dealing with tensor sizes

- Tensor are typically downsampled in their height and width dimensions and expanded in their depth or channels dimension
    - Pooling and convolution operations both downsample the image



Dealing with tensor sizes

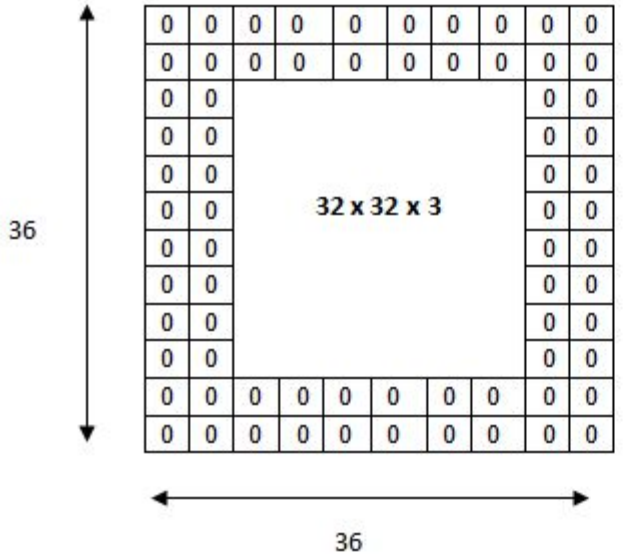
- Modern convolutional networks can have upwards of 100 layers
  - Using the operations as shown will quickly downsample the feature maps to nothing
  - We want to be able to control when we downsample the image
- Calculate amount downsampling for a convolutional layer opteration
  - W - width of featuremap/image
  - F - kernel size
  - P - padding
  - S - kernel stride

$$\frac{W - F + 2P}{S} + 1$$

Dealing with tensor sizes

- Padding involves adding rows and columns of all zeros around our image or feature map

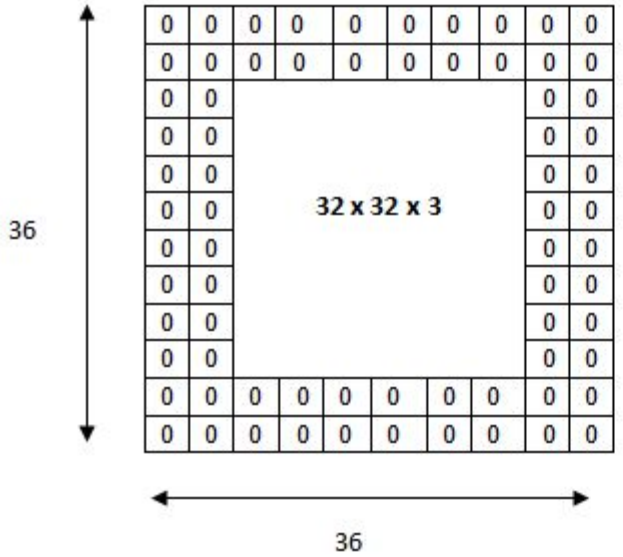
- If we have a kernel of size 5x5 and we pad our image by 2 we will get a feature map of the same size as the image



Dealing with tensor sizes

- Gives us more control over the spatial dimensions
  - Control downsampling with stride and pooling

$$\frac{W - F + 2P}{S} + 1$$



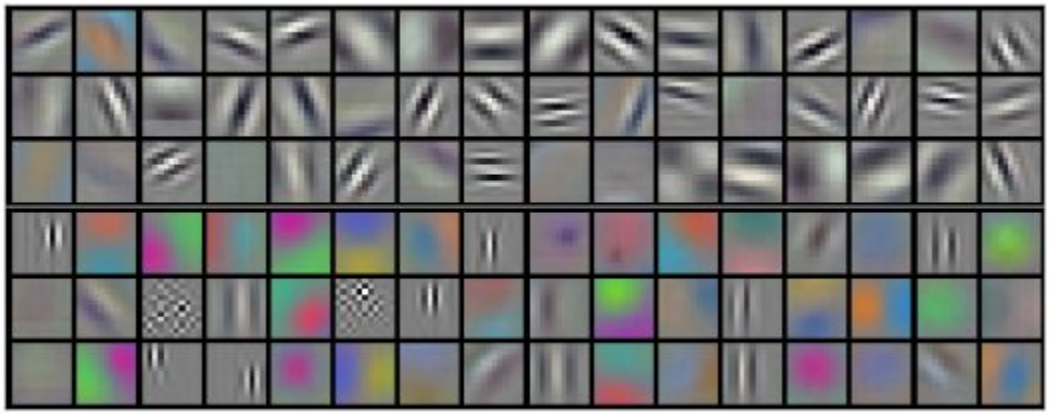
Representation Learning with Convolutional Neural Networks

Feature Maps

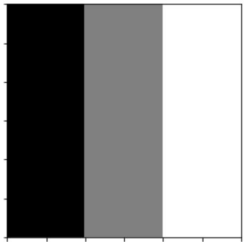
- Using layers of kernels allows us to make use of the spatial layout of our data
- Each kernel creates its own feature map
  - The feature maps should therefore be some representation of the data that maintains the spatial relationships
  - Each kernel will produce a feature map that is influenced by the weight values within that kernel
    - Through the process of training the weights of a kernel will develop into meaningful feature detectors for the given problem

Feature Maps

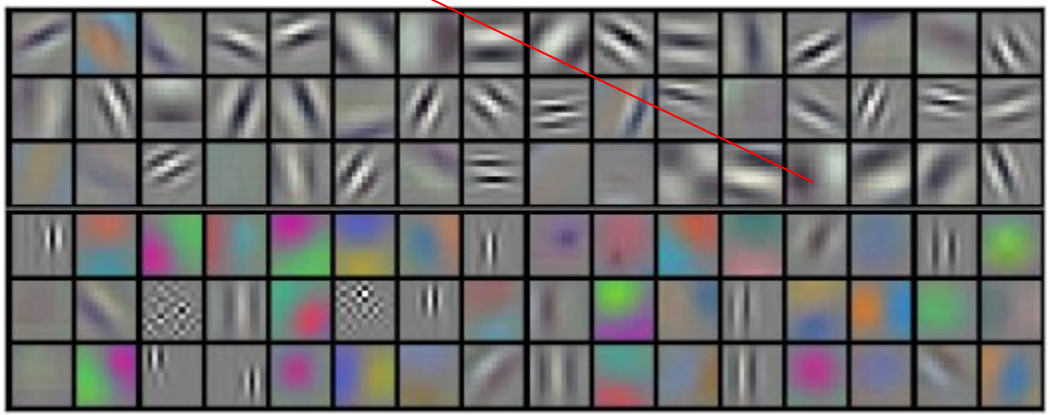
- Visualizing the kernels as images from the first layer of a trained neural network
  - Filters became edge detectors, color detectors, and color contrasts



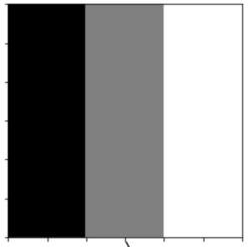
Feature Maps



Applying this kernel to an image will give us a mapping of all of the vertical edges in an image.



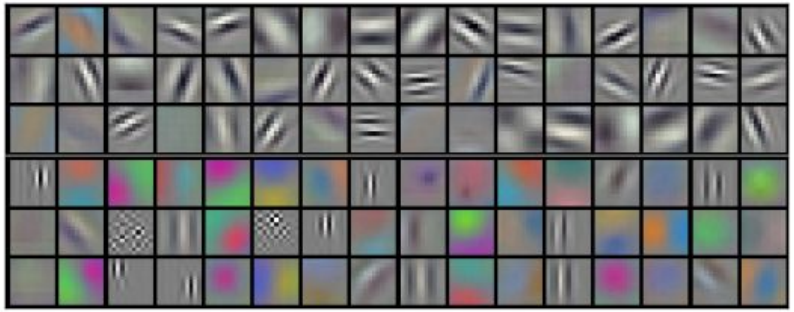
Feature Maps



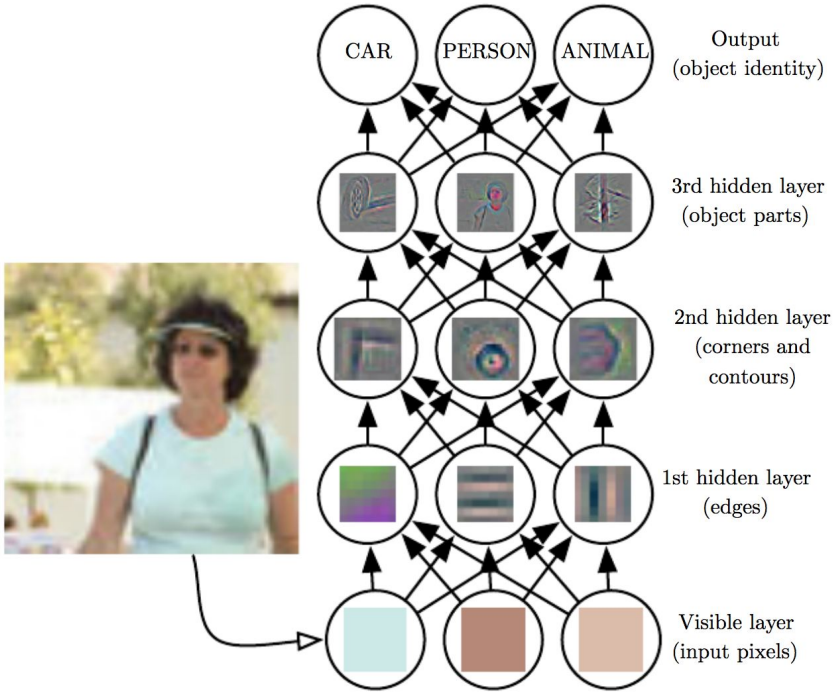
Feature Maps

Feature Maps

- Each one of these 96 kernels will create its own mapping of the type of feature that it is sensitive to
  - The kernel re-uses its weight values at every spatial location in the input (detects vertical edges everywhere)
  - The kernels in the next layer will have a depth of 96 and will evaluate how the features represented in the 96 maps come together to form higher level details



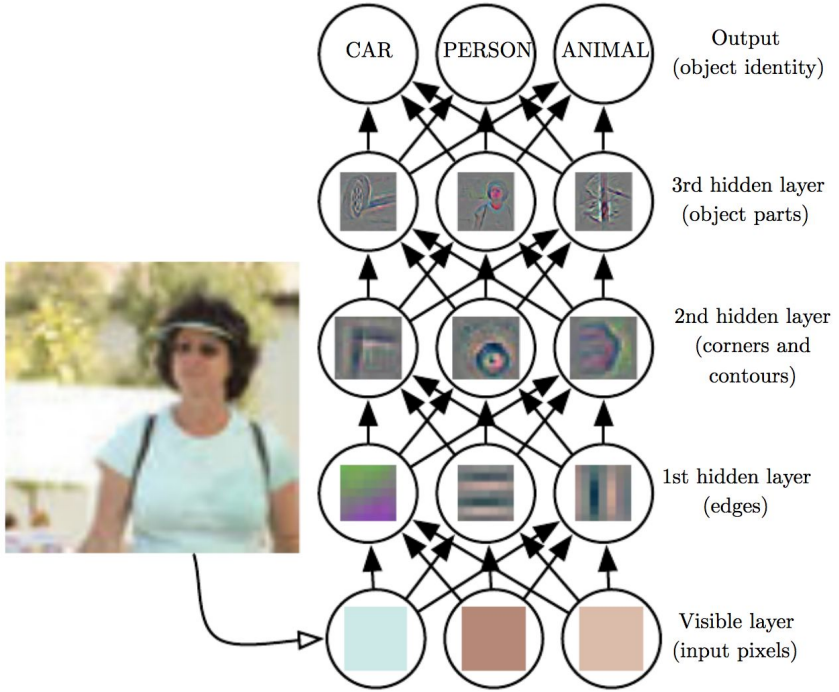
Hierarchical Learning



Deep Learning Book (2016)

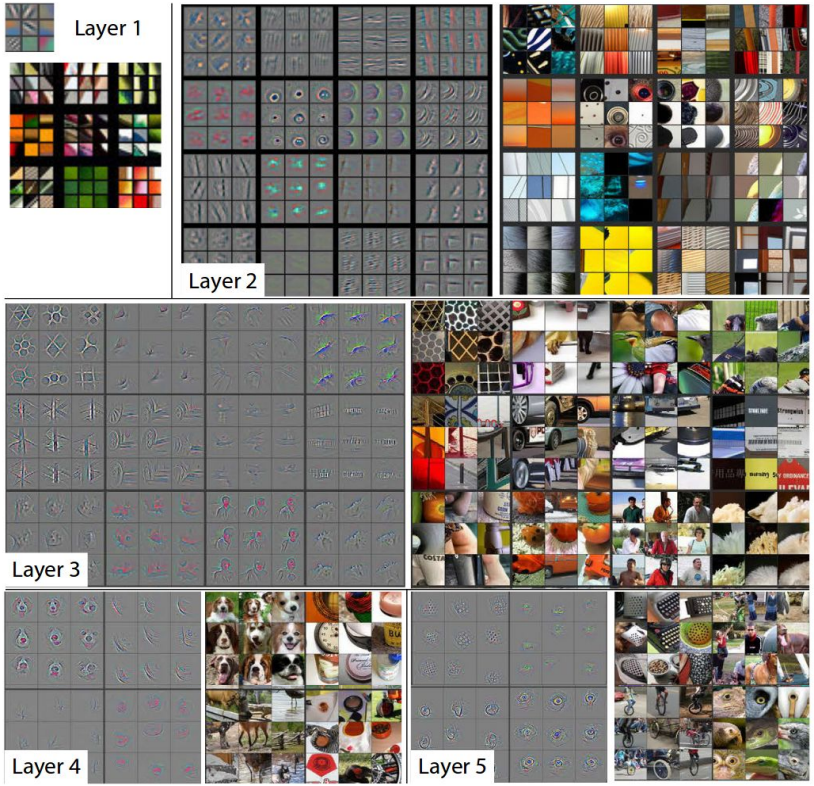
Hierarchical Learning

- Kernels are as deep as the input to that layer
- They calculate metrics on how features maps combine into higher level features
- Edges combine into corners and contours
- Corners and contours combine into object parts



Deep Learning Book (2016)

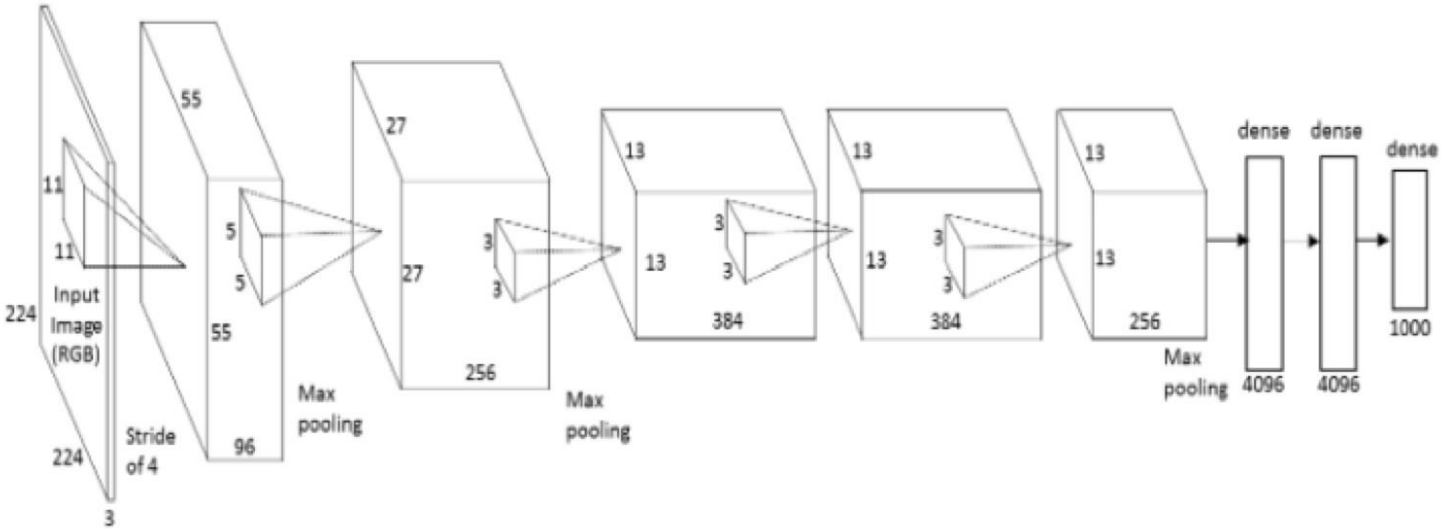
Hierarchical Learning



Zeiler and Fergus. Visualizing and Understanding Convolutional Networks,
https://cs.nyu.edu/~fergus/papers/zeilerECCV2014.pdf

- Shows the maximally activated features maps and corresponding image segments at various layers in a network
- Network focuses on larger image regions in the deeper layers

Hierarchical Learning



- The deep part of the network is still responsible for learning useful representations of the input data and converting into a form that makes the problem easy for the final, task specific, layers.

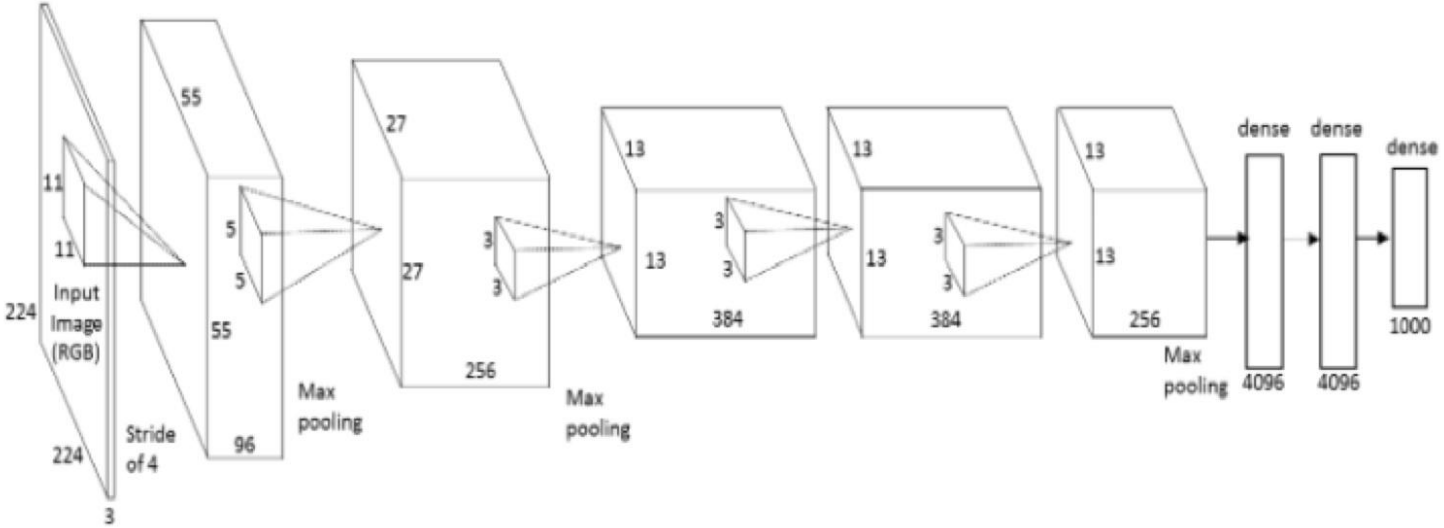
Hands on TensorFlow

TensorFlow - Convolutional Networks

- **`tensorflow_tutorials/convolutional_neural_networks.ipynb`**
- Designing convolutional neural network layers in TensorFlow
  - Emphasis on designing generic and reusable layers to aid in experimentation and prototyping
- Using TensorFlow optimizers and automatic differentiation to train a neural network
  - Mini-batch gradient descent
  - Classifying images from a clothing catalogue into various categories
- Saving TensorFlow sessions as checkpoints
  - Loading checkpoint to continue training
  - Making predictions with a trained neural network in a checkpoint

Convolutional Neural Network Architectures and Use Cases

AlexNet



- Winner of the 2012 ImageNet competition with an error rate of about 16%.
    - Fast, lightweight network. Sacrifices some performance

Krizhevsky et al. ImageNet Classification with Deep Convolutional Neural Networks,
https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

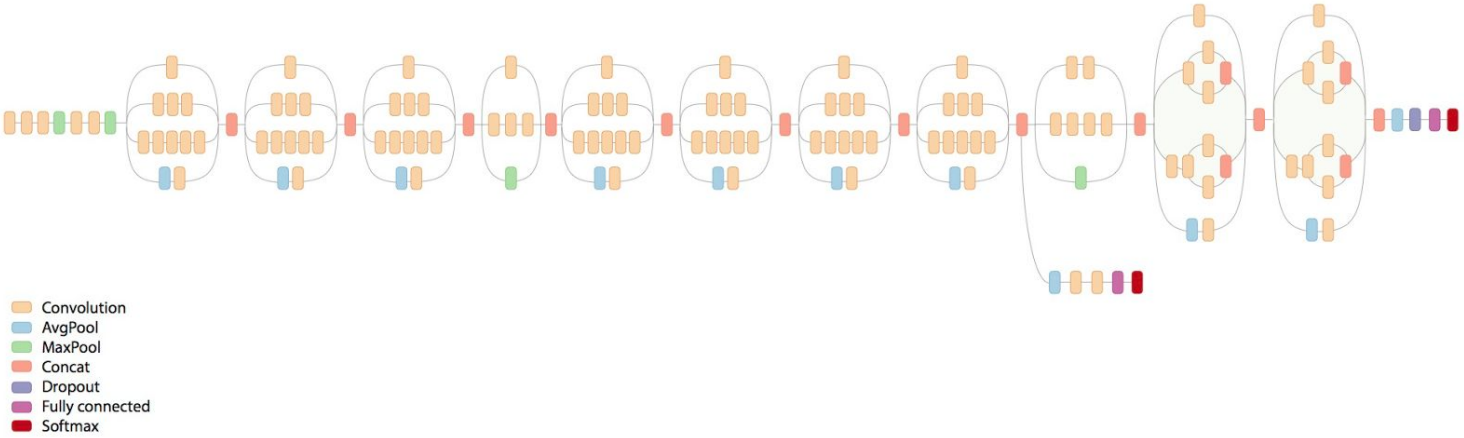
VGG

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 image)					
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					

- Runner up of the 2014 ImageNet competition with an error rate of about 9%.
  - Slower, very high performing across many tasks
  - A few architectures built on the “VGG design principles”

Simonyan et al. Very Deep Convolutional Networks for Large-Scale Image Recognition, https://arxiv.org/abs/1409.1556

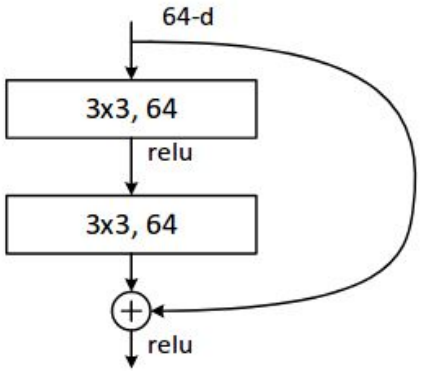
Inception V3

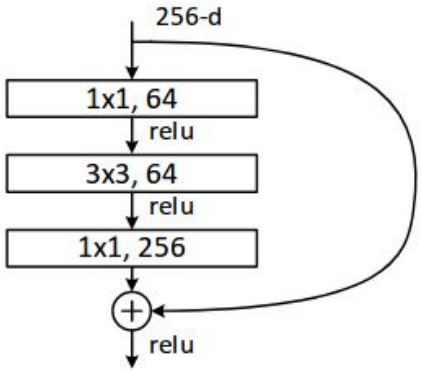


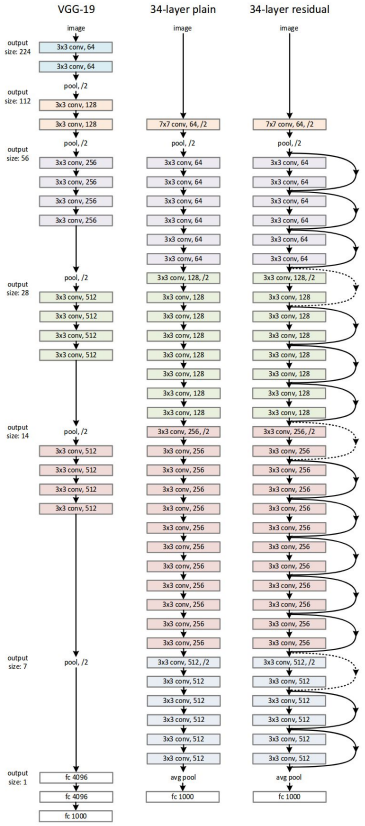
- Current iteration of the 2014 ImageNet winning architecture, error rate of 6.44%
  - Fast for how high performing it is
  - Introduces the “inception module”

Szegedy et al. Rethinking the Inception Architecture for Computer Vision, https://arxiv.org/abs/1409.1556

ResNet







- The 2015 ImageNet winning architecture, error rate of about 5.94%
  - Lower complexity than VGG with many more layers
  - Characterized by stacking two 3x3 conv layers and introducing skip connections

He et al. Deep Residual Learning for Image Recognition, https://arxiv.org/abs/1512.03385

Why care about these academic/research networks?

Transfer Learning

Transfer Learning

- Successful applications of deep learning in computer vision are often driven by this technique
- Transfer learning is the method where we take a model trained on some task and adapt it to be used on a new task
    - We are transferring the knowledge gained while learning one task to get a jump start on a new task

Transfer Learning

- Effective due to representation learning and the hierarchical nature of how a convolutional network learns
- Many of the kernels in a fully trained network tend to be generic and applicable to various types of images
    - Reusing those generic kernels decreases training time
    - Boosts generalization
    - Use much less data then if we trained from scratch

Transfer Learning

- Transfer learning (fine tuning) has been shown to outperform training from scratch on small and medium sized datasets

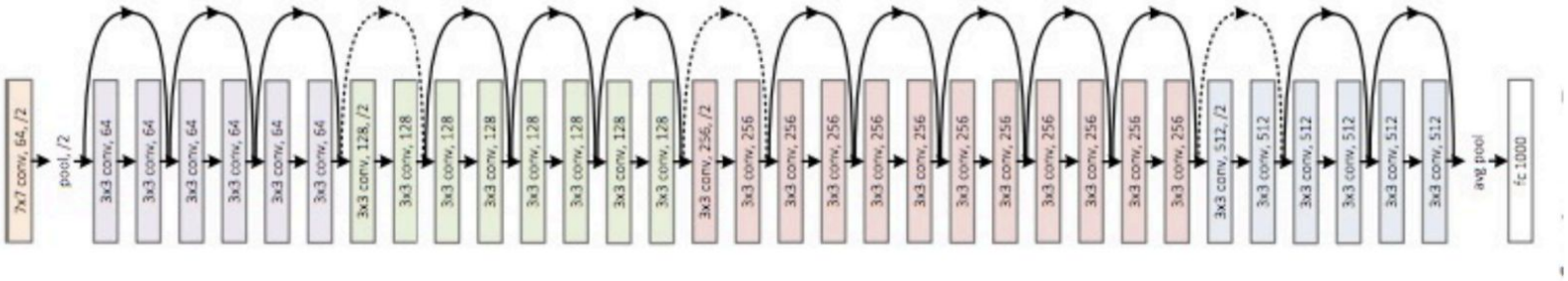
Model	Test top 1 (top 5)	Assoc. train top 1 (top 5)
(a) Bag of visual Words	23.96	–
(b) BossaNova	28.59	–
(c) Overfeat & Extraction	33.91	–
(d) Overfeat & From Scratch	47.46 (69.37)	79.14 (94.49)
(e) Overfeat & Fine Tuning	57.98 (78.86)	89.69 (97.96)
(f) Vgg16 & Extraction	40.21	–
(g) Vgg16 & From Scratch	53.62 (74.67)	88.17 (97.68)
(h) Vgg16 & Fine Tuning	65.71 (82.54)	96.18 (99.39)
(g) InceptionV3 & Fine Tuning	**66.83 (84.53)**	85.34 (95.91)

Remi Cadene's masters thesis - Deep Learning for Visual Recognition

Model type	Test mAP	Train mAP
(a) BoW	53.2	–
(b) BossaNova and FishersVector	61.6	–
(c) Vgg16 from scratch	39.79	99.73
(d) Vgg16 extraction	83.22	–
(e) Vgg16 fine tuned	**85.70**	98.81

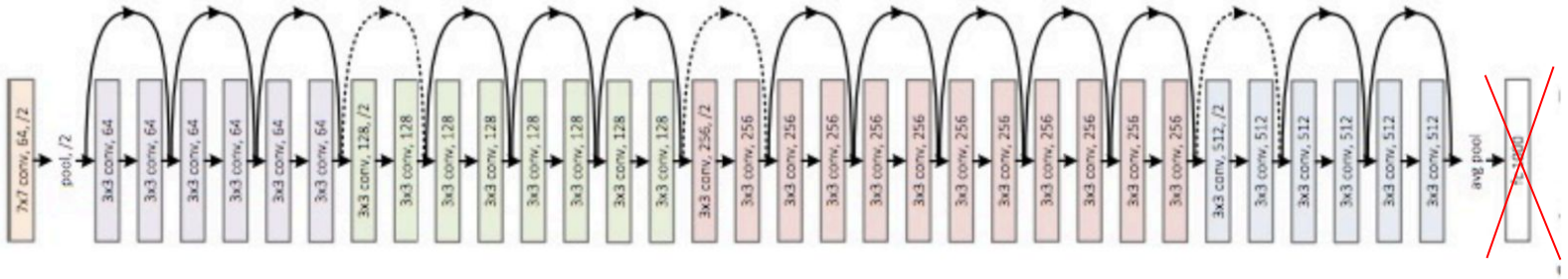
Transfer Learning

- We start with a trained neural network on some large generic dataset



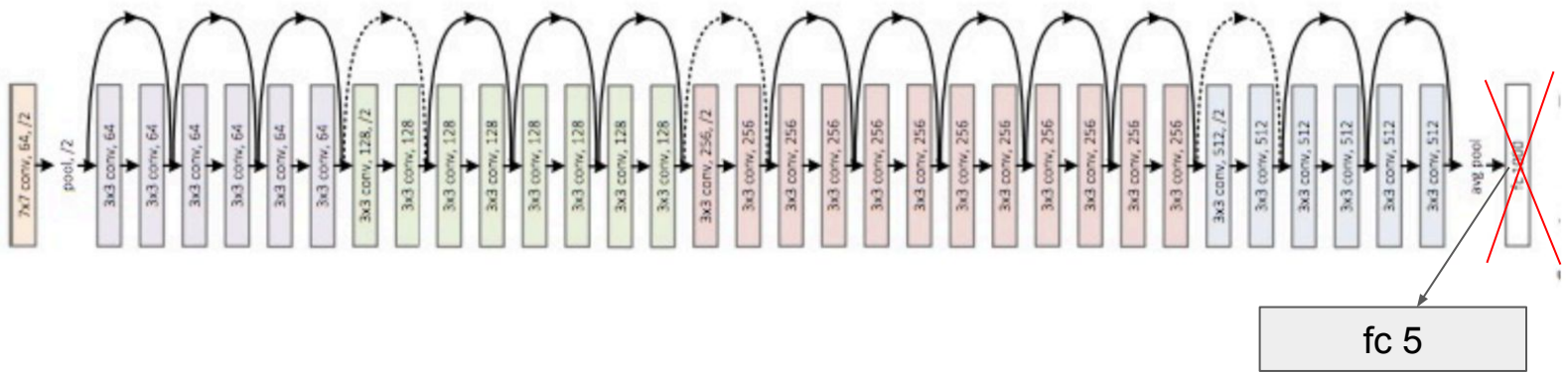
Transfer Learning

- Only the last layer in this network is specific to the task that it was originally trained on
  - In this case it's a 1000-way classification task
- Remove this layer



Transfer Learning

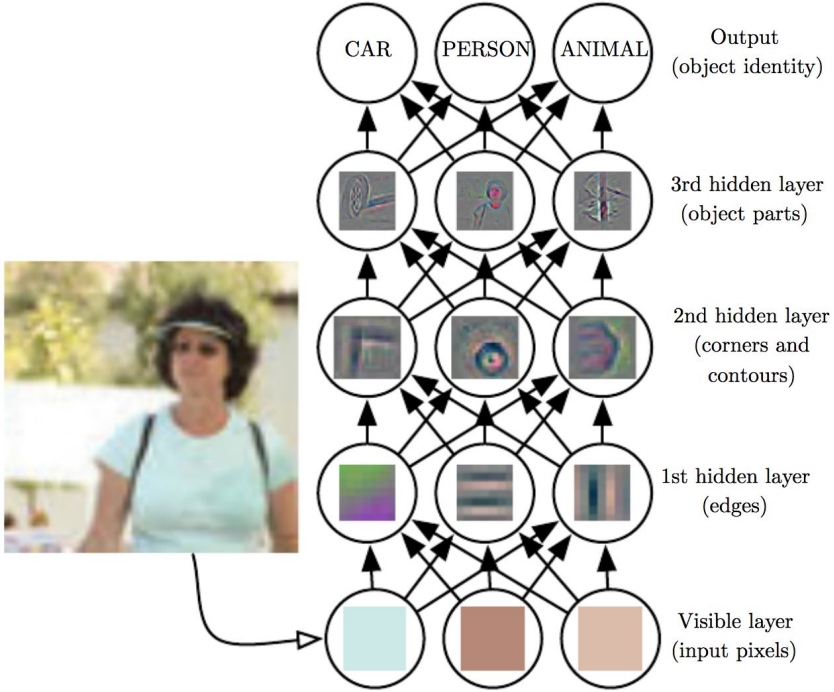
- Add our new task specific layer in place of the layer that we removed
  - Train the network as if we were starting from scratch, but
    - Use a lower learning rate
    - Requires less iterations over data to converge on a solution



Why does it work?

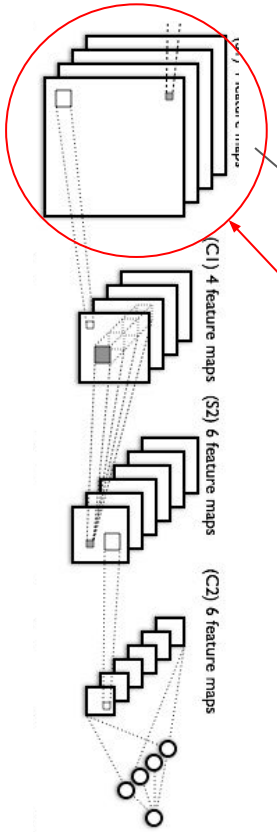
- Deep learning works by learning useful representations of data
- With images many of these representations are going to be generic and true for all images
    - Edges, contours, shapes, colors
- Network already learned how to ‘see’ by learning a dataset of 1.2 million images across 1000 categories
    - We are then fine-tuning what a network will focus on

Why does it work?

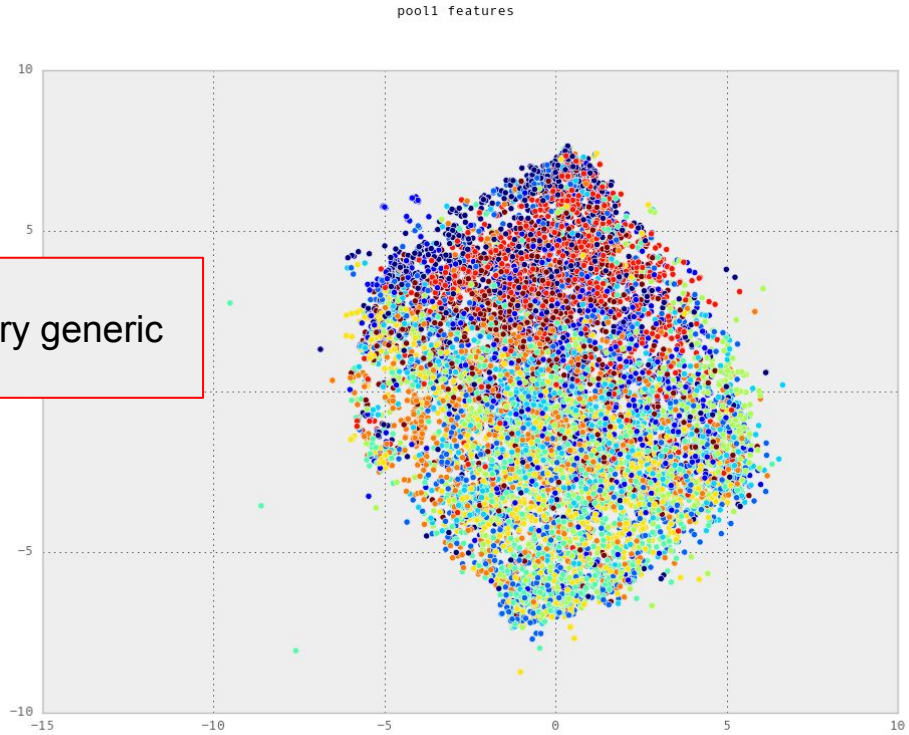


Deep Learning Book (2016)

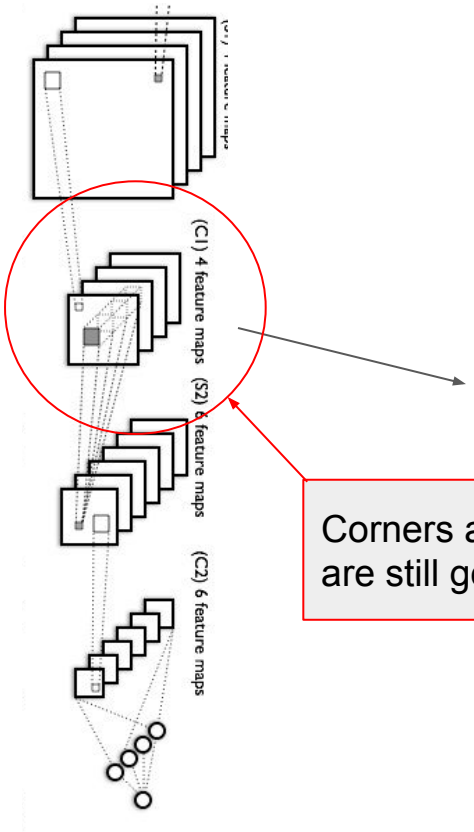
Why does it work?



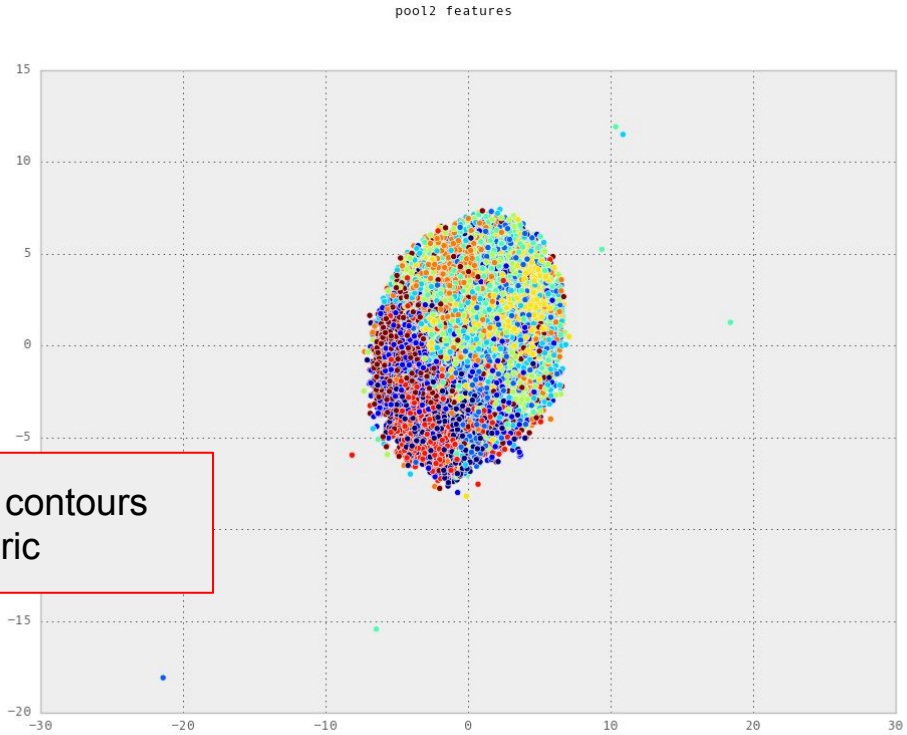
Edges are very generic



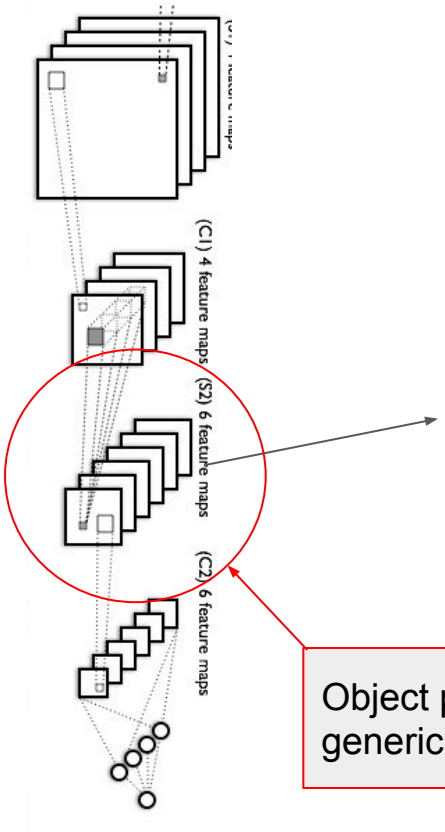
Why does it work?

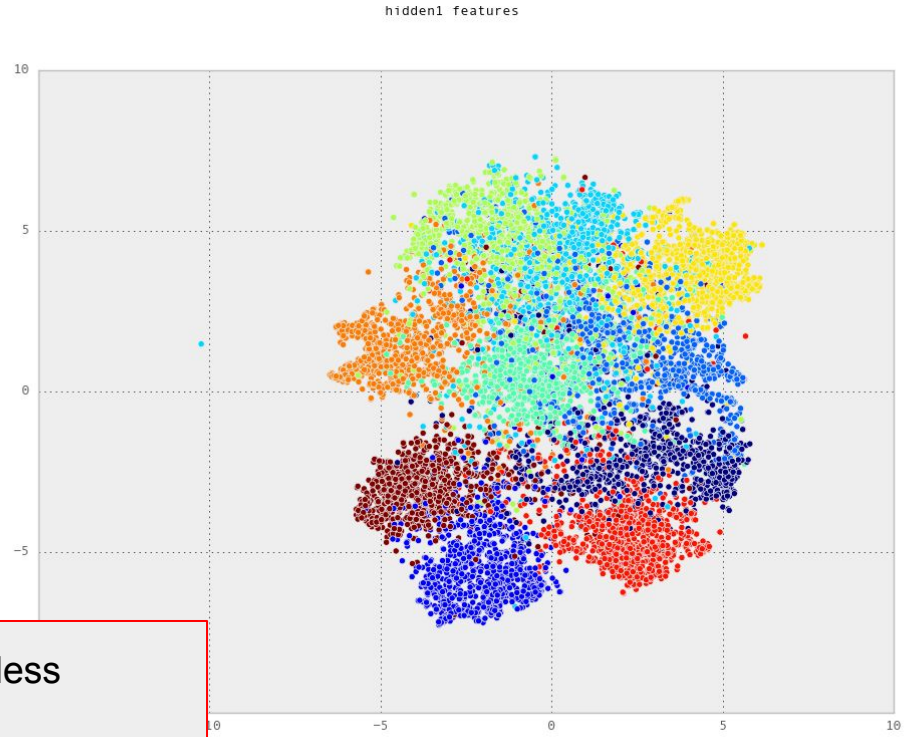


Corners and contours are still generic



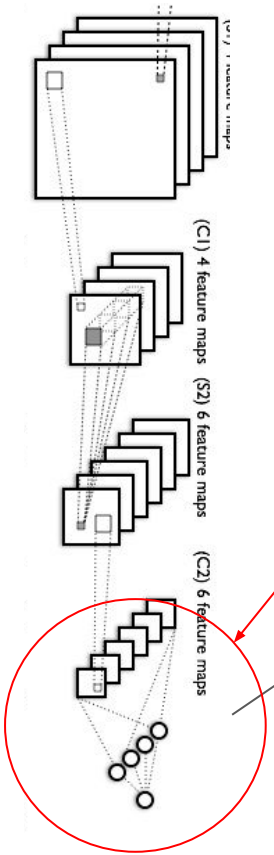
Why does it work?



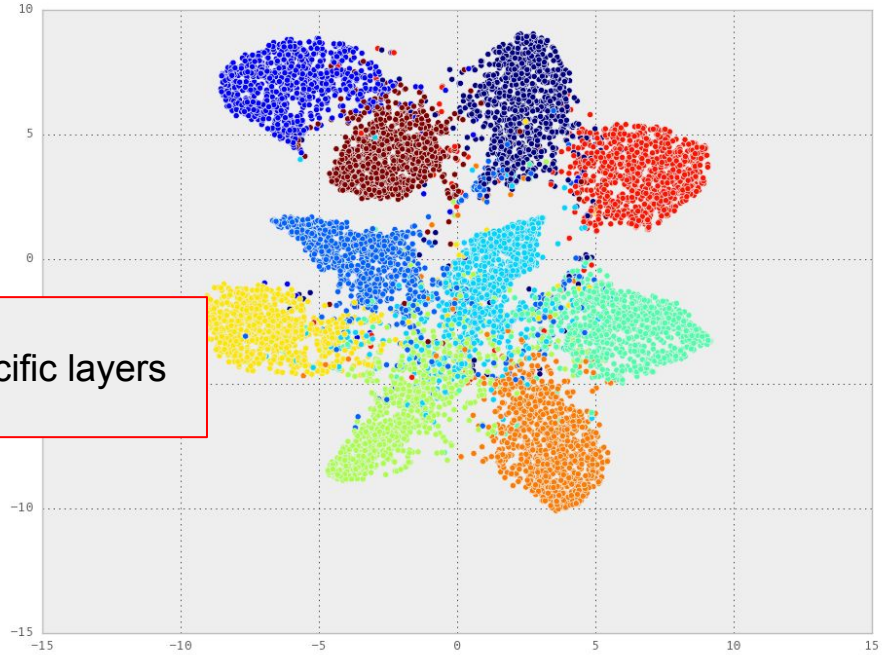


Object parts less generic

Why does it work?



Problem specific layers



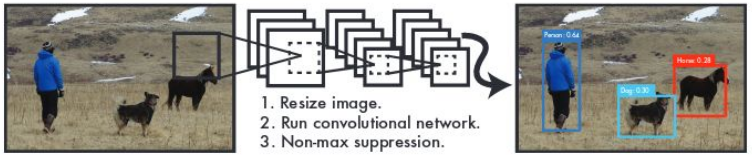
Where to get pretrained networks?

- Most deep learning frameworks contain something called a ‘Model Zoo’
  - Researchers and practitioners will post code and weights for fully trained models
    - Authors of papers or people reproducing research

Object Detection

Object Detection

Convolutional neural networks can also localize objects in image



Redmon et al., You Only Look Once: Unified, Real-Time Object Detection, https://pjreddie.com/media/files/papers/yolo.pdf

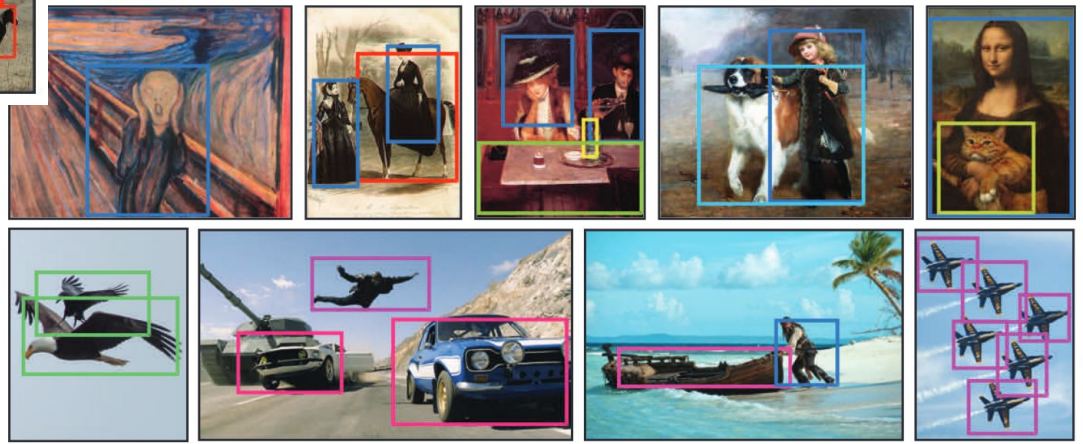
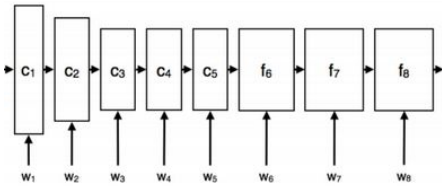


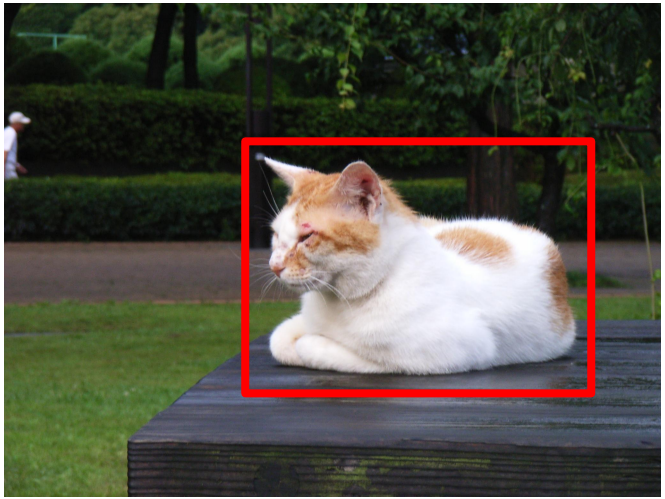
Figure 6: Qualitative Results. YOLO running on artwork and natural images. It is mostly accurate although it does think one person in an image is an airplane.

Object Detection - One object

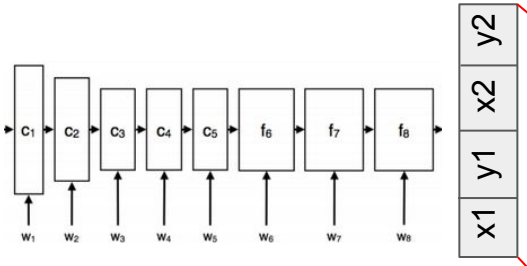
- When trying to localize a single object in an image we can set up our network to predict the coordinates of the box that would surround the object
  - Set up this problem as a regression
  - Predict:
    - x1, y1, x2, y2
    - x1, y1, width, height
  - Use least squared error as the loss function

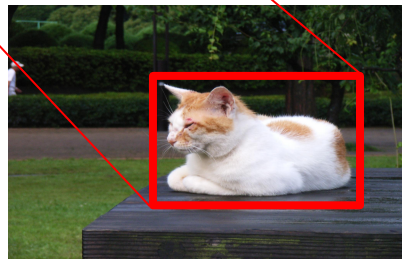
Object Detection - One object



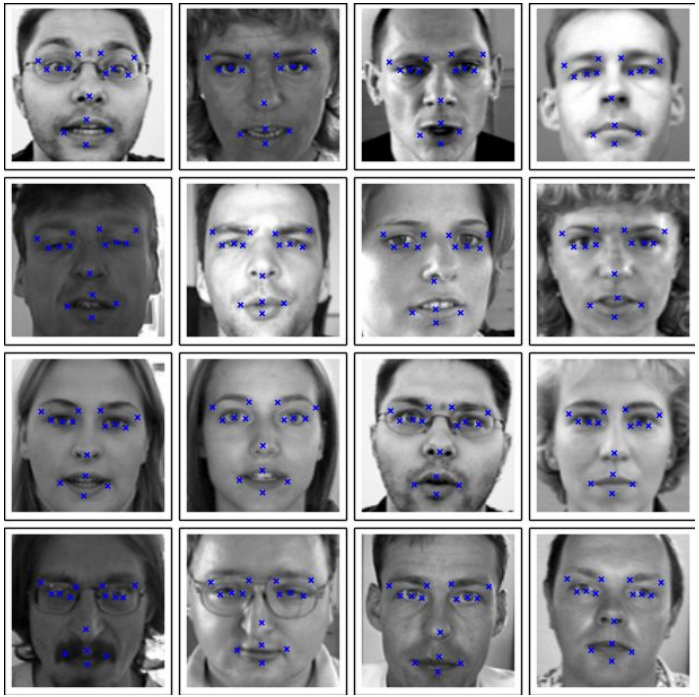


Object Detection - One object



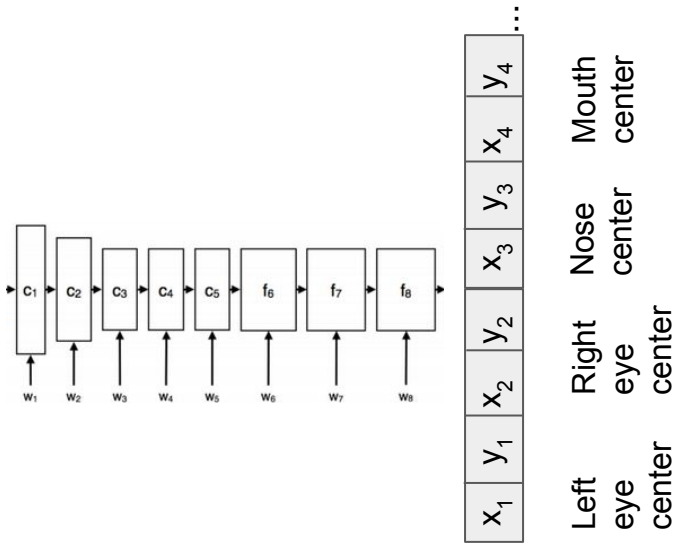


Object Detection - Fixed number of objects



http://danielnouri.org/notes/2014/12/17/using-convolutional-neural-nets-to-detect-facial-keypoints-tutorial/

Object Detection - Fixed number of objects



Object Detection - variable number of objects

- What if there can be any number of objects in an image?

Object Detection - variable number of objects

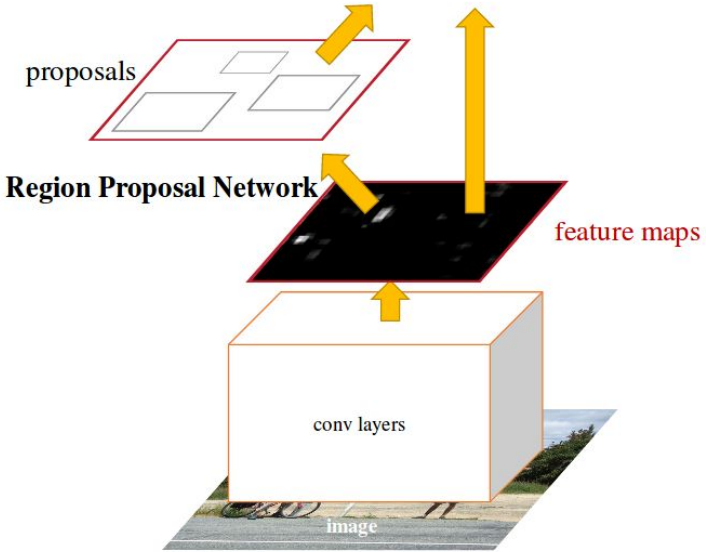
- The previous framework would not work on these images
  - We would have to choose some finite number of boxes to predict
  - Most multi object problems will have some variable number of objects
  - Need to use a more sophisticated model that is flexible with respect to the number of outputs

Object Detection - variable number of objects

- Use a region proposal method
- Use a single shot method

Object Detection - Using region proposals

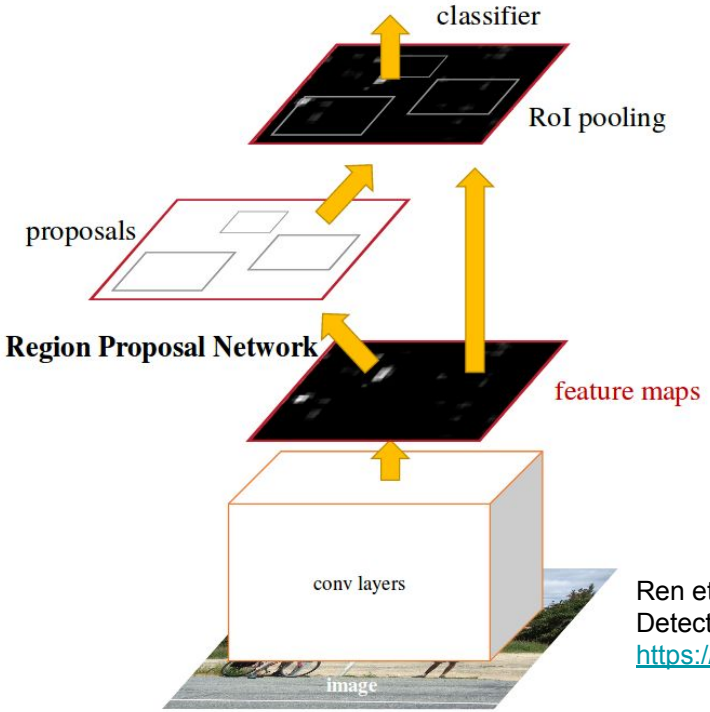
- One solution to this is that we use a region proposal technique
  - They detect objects without specifying the class
  - Simply look for regions in an image that are likely to contain an object
  - Can do this with the feature maps coming out of a ConvNet



al. Faster R-CNN: Towards Real-Time Object
on with Region Proposal Networks,
arxiv.org/pdf/1506.01497.pdf

Object Detection - Using region proposals

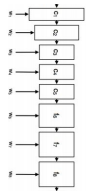
- Can then classify each region as what object is contained within



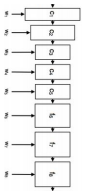
Ren et al. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks, https://arxiv.org/pdf/1506.01497.pdf

Object Detection - Getting boxes

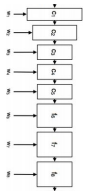
- The region proposals give us the location of the objects
  - Use those locations as-is
  - Or, add a section to the network to double check the regions



(0, 0, 0, 0)
Good box



(-0.15, 0, 0, 0)
Shift left



(0, 0, 0, 0.15)
Make taller

Object Detection - Training

- Setting up the training for this type of object detection network is not trivial
  - Training from scratch is a beyond the scope of the class
  - However, we mostly likely will never train from scratch
    - Transfer Learning!

Object Detection - Training

- TensorFlow's object detection models already have the details of training worked out
  - Need only to provide data, boxes and classes
  - https://github.com/tensorflow/models/tree/master/research/object_detection

Tensorflow Object Detection API

Creating accurate machine learning models capable of localizing and identifying multiple objects in a single image remains a core challenge in computer vision. The TensorFlow Object Detection API is an open source framework built on top of TensorFlow that makes it easy to construct, train and deploy object detection models. At Google we've certainly found this codebase to be useful for our computer vision needs, and we hope that you will as well.

Contributions to the codebase are welcome and we would love to hear back from you if you find this API useful. Finally if you use the Tensorflow Object Detection API for a research publication, please consider citing:

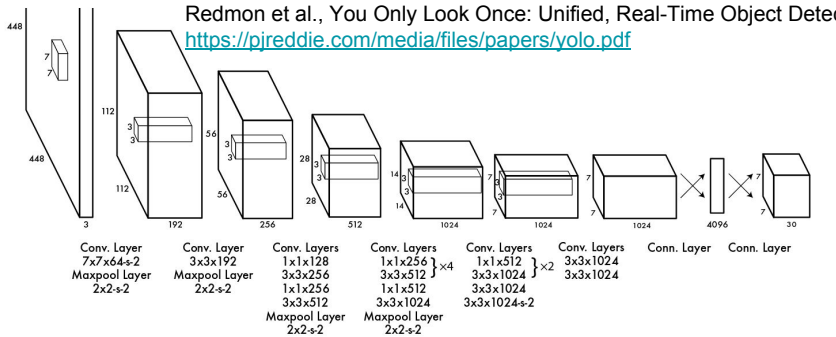
```
"Speed/accuracy trade-offs for modern convolutional object detectors."
Huang J, Rathod V, Sun C, Zhu M, Korattikara A, Fathi A, Fischer I, Wojna Z,
Song Y, Guadarrama S, Murphy K, CVPR 2017
```

Object Detection - End-to-end

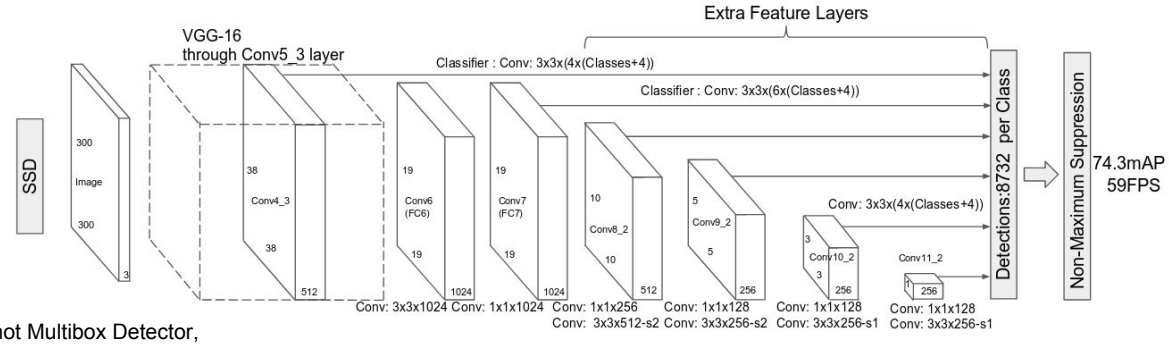
- The region proposal method (R-CNN) has the downside of being slow and expensive to run
  - “Looks” at image multiple times
    - Get region proposals
    - Check each proposal
- Would be faster to just run the image through a network once and get an output

Object Detection - End-to-end

- Two networks designed with this idea
  - Single shot multibox detector (SSD)
  - You only look once (YOLO)



Redmon et al., You Only Look Once: Unified, Real-Time Object Detection, https://pjreddie.com/media/files/papers/yolo.pdf



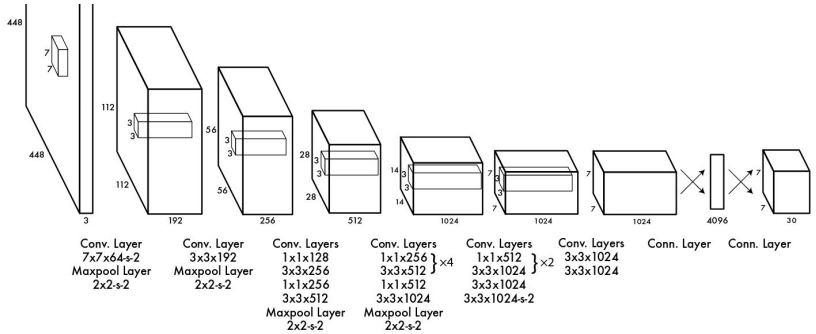
Liu et at., SSD: Single Shot Multibox Detector, https://arxiv.org/pdf/1512.02325.pdf

Object Detection - YOLO is super fast

Redmon et al., You Only Look Once: Unified, Real-Time Object Detection, https://pjreddie.com/media/files/papers/yolo.pdf

Object Detection - YOLO

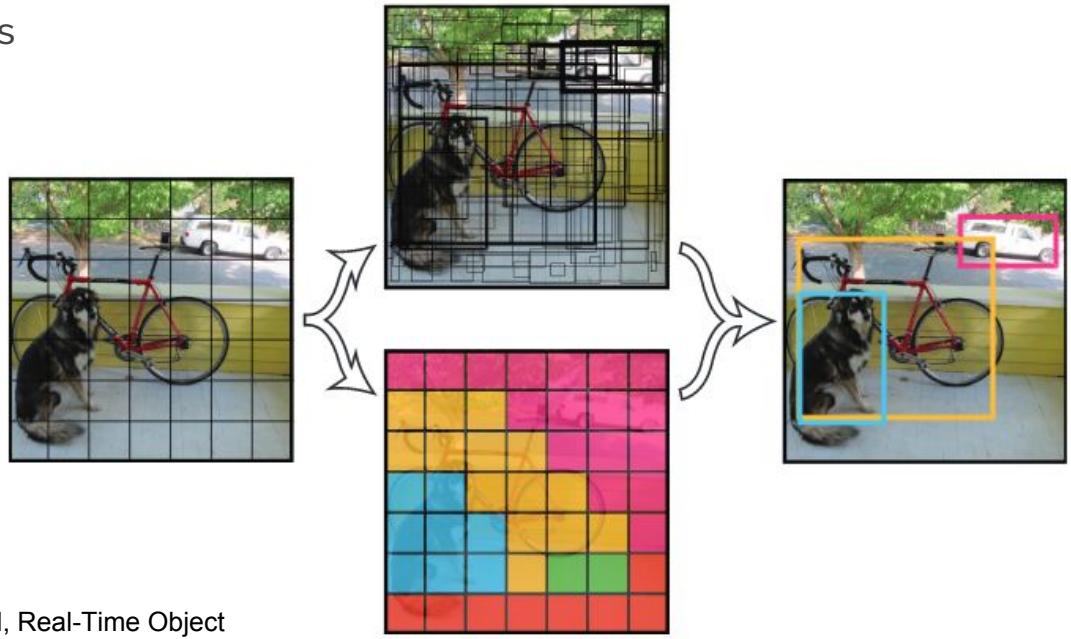
- Fairly standard looking CNN except for the output
  - Predicts a 7x7x30 tensor rather than a vector



Redmon et al., You Only Look Once: Unified, Real-Time Object Detection, https://pjreddie.com/media/files/papers/yolo.pdf

Object Detection - YOLO

- 7x7x30 tensor
  - Divides image into 7x7 grid
  - Each square outputs 30 values
    - Class of object (20)
    - Confidence of box (1)
    - Coordinates of box (4)
      - Predicted two bounding boxes in paper (5 *2)



Redmon et al., You Only Look Once: Unified, Real-Time Object Detection, https://pjreddie.com/media/files/papers/yolo.pdf

Object Detection - YOLO Training

- Training YOLO amounts to segmenting out the parts of the output tensor that correspond to their respective ground truths
  - Calculating the loss for each
  - Summing all the losses up for the total loss to be backpropagated

$$\lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left(x_i - \hat{x}_i\right)^2 + \left(y_i - \hat{y}_i\right)^2$$

$$+ \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left(\sqrt{w_i} - \sqrt{\hat{w}_i}\right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i}\right)^2$$

$$+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left(C_i - \hat{C}_i\right)^2$$

$$+ \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} \left(C_i - \hat{C}_i\right)^2$$

$$+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{\text{obj}} \sum_{c \in \text{classes}} \left(p_i(c) - \hat{p}_i(c)\right)^2$$

Redmon et al., You Only Look Once: Unified, Real-Time Object Detection, https://pjreddie.com/media/files/papers/yolo.pdf

Hands on TensorFlow

TensorFlow - Object Localization

- `tensorflow_tutorials/object_localization.pynb`
- Running bounding box regression on convolutional neural networks
- Using TensorFlow optimizers and automatic differentiation to train a neural network
    - Mini-batch gradient descent
    - Classifying images from a clothing catalogue into various categories
- Saving TensorFlow sessions as checkpoints
    - Loading checkpoint to continue training
    - Making predictions with a trained neural network in a checkpoint

TensorFlow - Object Localization

- **`tensorflow_tutorials/localization_with_classification.ipynb`**
- Running bounding box regression and class prediction jointly with convolutional neural networks
  - See how TF will allow a user to make complex loss functions and still train a network
- Using TensorFlow optimizers and automatic differentiation to train a neural network
  - Mini-batch gradient descent
  - Classifying images from a clothing catalogue into various categories
- Saving TensorFlow sessions as checkpoints
  - Loading checkpoint to continue training
  - Making predictions with a trained neural network in a checkpoint