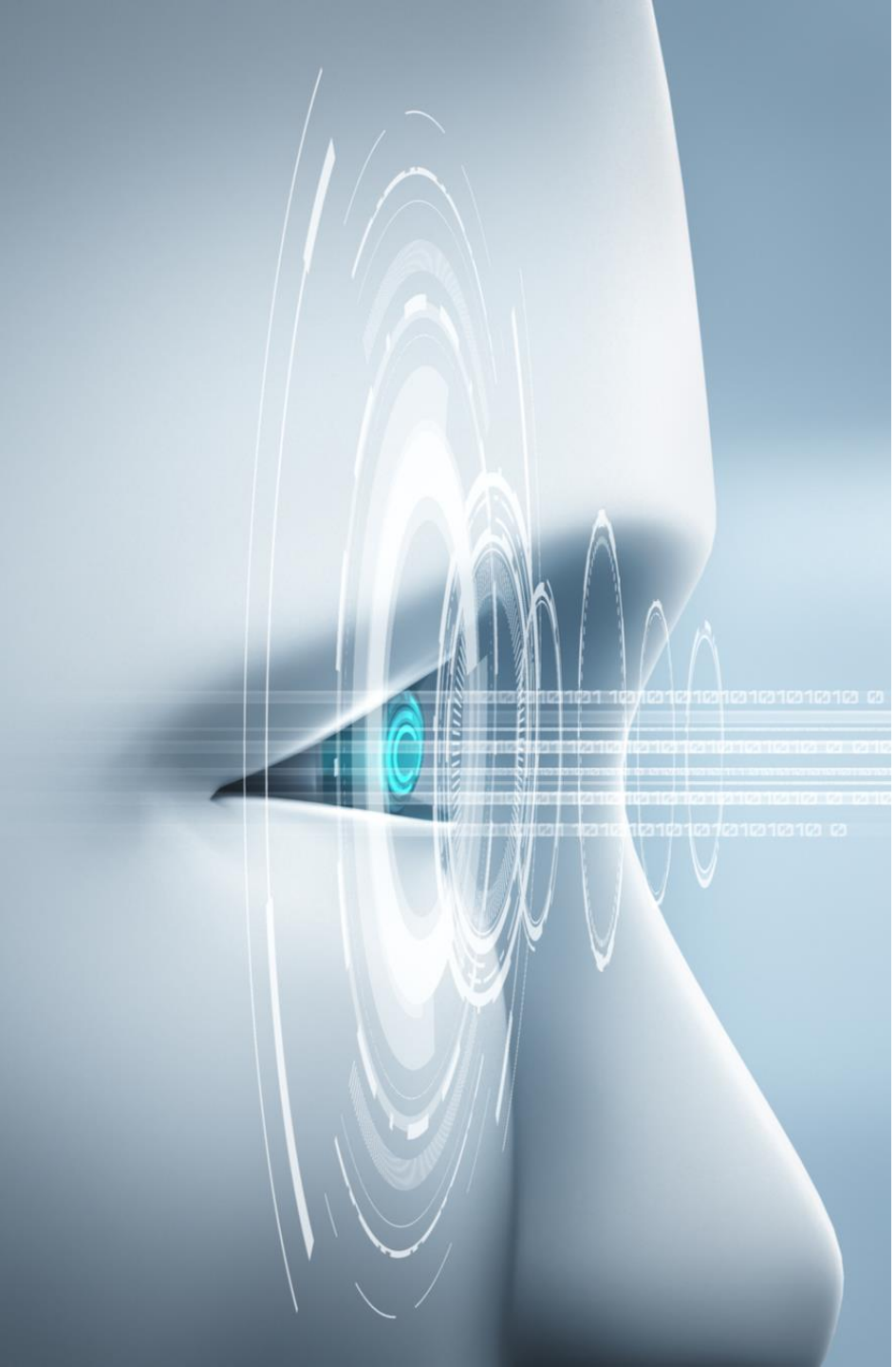


Computer Vision Applications

BY QUADEER SHAIKH

About me

Work Experience

- Risk Analyst
  - Morgan Stanley (Jan 2023 – Present)
- Data Science Intern
  - AkzoNobel Coatings International B.V. Netherlands (Feb 2022 – Dec 2022)
- Data Science Intern
  - EzeRx Health Tech Pvt. Ltd. (Jan 2022 – July 2022)
- Associate Engineer
  - Tata Communications Ltd. (July 2019 – Aug 2020)
- Network Automation and Analysis Engineer Intern
  - Cisco (June 2018 – July 2018)

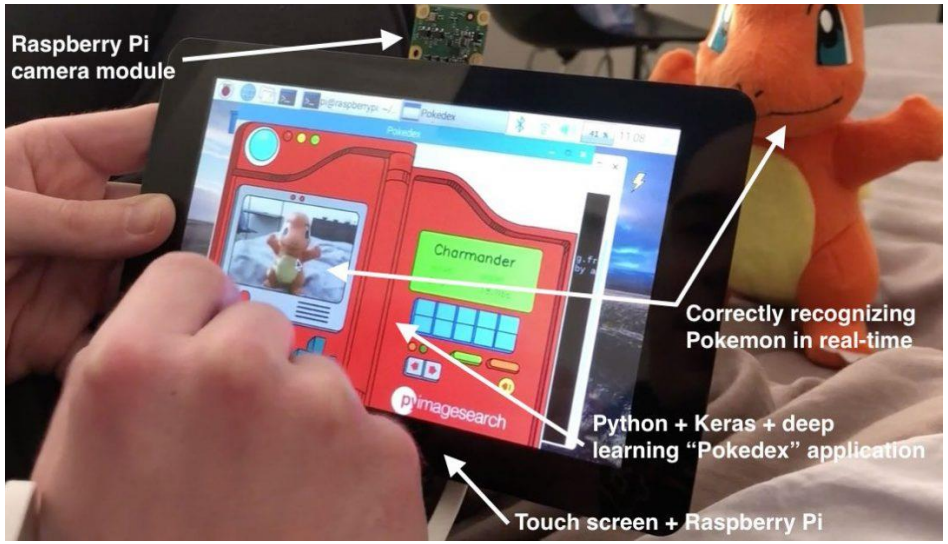
Education

- M.Tech – Artificial Intelligence
  - NMIMS (2021 - 2023, currently pursuing)
- B.E. – Computer Engineering
  - Mumbai University (2015 - 2019)

The Deep Learning Era

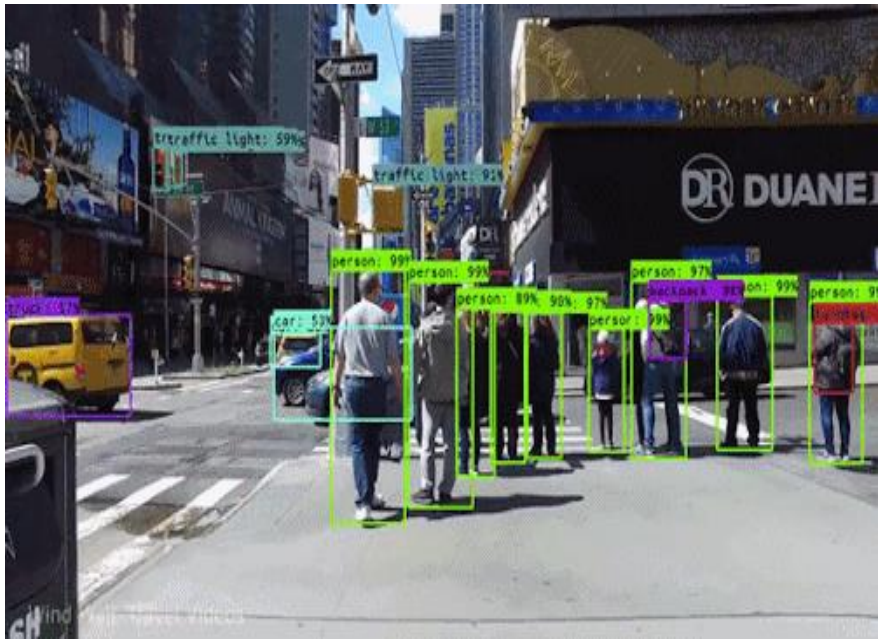
Deep Learning Phase Outline

1. A revisit to an old friend: Neural Networks
2. A visit to a new friend: Convolutional Neural Networks (CNNs)
3. Image Augmentation
4. Optimization and Improvement of CNNs
5. Diagnosis of CNNs
6. CNNs Architectures and Transfer Learning
    1. AlexNet
    2. VGG16/19
    3. InceptionNet/GoogleNet
    4. ResNet
    5. MobileNet
    6. EfficientNet



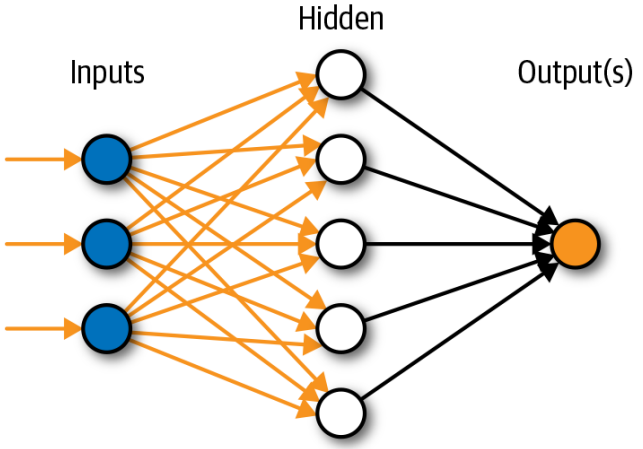
Deep Learning Phase Outline

1. What do CNNs Learn ?
2. One Shot Learning (Face Recognition)
    1. Siamese Network
3. Image Segmentation
    1. FCN
    2. U-Net (Optional)
4. Object Detection (Architecture details and training framework)
    1. RCNN
    2. Fast RCNN
    3. Faster RCNN
    4. YOLO



A revisit to an old friend: NNs

1. An ANN is based on a collection of connected units or nodes called artificial neurons, which loosely model the neurons in a biological brain
2. Used for non linear pattern recognition
3. Always count the hidden layers + output layers
   - This is a 2 layers neural network
4. Hyperparameters need to be configured:-
   1. No. of layers
   2. No. of neurons in each layer
   3. No. of epochs
   4. Learning rate
   5. Loss function, Optimizer → Notes!
   6. Batch size
5. How do you count the total parameters in your neural network ?



A revisit to an old friend: NNs

1. How do you calculate the total parameters in the neural network ?

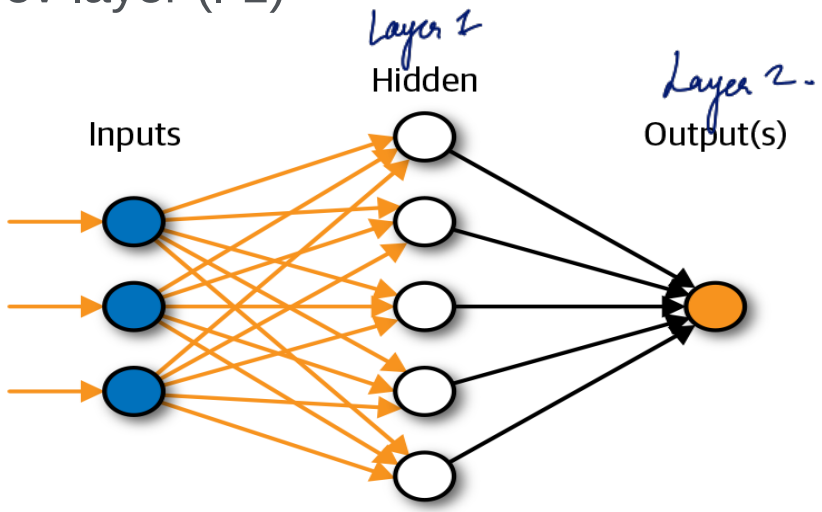
Ans. A neural network has 2 kind of params: weights and biases

Params for a given layer l

Weights = No. of units in current layer (l) x No. of units in prev layer (l-1) 5 x 3 = 15

Biases = No. of units in current layer (l) x 1 5 x 1 = 5

Params = no. of weights + no. of biases = 20



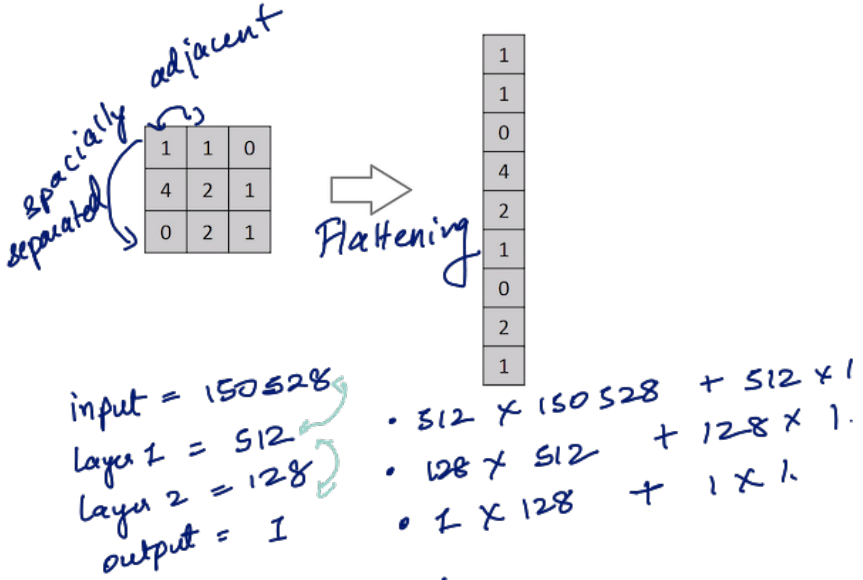
A revisit to an old friend: NNs

How to perform image classification using ANNs ?

1. Flatten the image
2. Pass the flattened image to the neural network

Problems with this for image related tasks:-

1. Images have a 2D spatial structure, therefore considering only the flattened pixel values are not very useful features for tasks like classification
   - Pixels that spatially separated are treated the same way as pixels that are adjacent
2. No way for network to learn features at different places in an input image
3. Imagine flattening an image of resolution 224x224x3, every image will be represented using a vector of size 150,528.
4. Now imagine using a neural network which has 2 hidden layers of 512 and 128 neurons each and one output layer of 1 neuron. Can you calculate the number of parameters required ?



Params weights biases.

1. 512 x 150,528 + 512 x 1 = 77,070,848
2. 128 x 512 + 128 x 1 = 65,664
3. 1 x 128 + 1 x 1 = 129

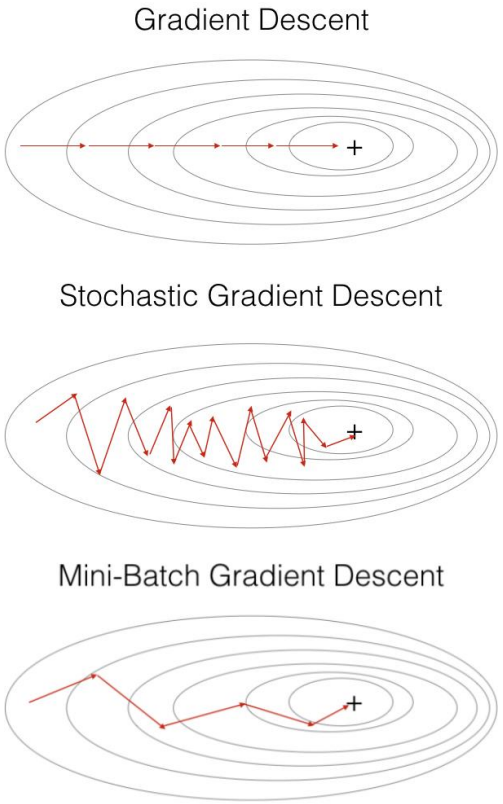
Total = 77,136,641

A revisit to an old friend: NNs

Fantastic optimization algorithms and how to use them

1. Batch Gradient Descent → on the entire dataset at once
   - A misleading name given by the ML community, the batch gradient descent updates the weight of the neural network by performing forward & backward propagation on the entire dataset i.e. all the training samples/records in a single epoch/iteration
2. Stochastic Gradient Descent → for every single datapoint separately
   - In batch gd we consider all the training records, but what if the training dataset is huge. You cannot fit huge amount of data into your memory (RAM) for the training process. We take one example each time from the records to update the weights using forward-backward prop (done for all records one by one in an entire epoch). Cost fluctuates a lot during the minimization and starts decreasing in the long run. Has a problem with convergence to minima.
3. Mini Batch Gradient Descent → for batches of data
   - SGD has a minima convergence problem and also cannot be implemented with vectorized computation (due to one sample taken each time for a forward-backward prop. Therefore we take mini batches from the dataset and update the weights repeatedly batch wise for an entire epoch. Convergence to minima is smoother than SGD.

Other optimizers like RMSProp, AdaGrad, Adam, etc also use mini batch implementations



A visit to a new friend: CNNs

Remember Convolution operation ?

Kernels/Filters were used to detect features in an image by convolving them on the image

How to decide weights of kernels ?

non-linearity : change in one variable results in a disproportional change in another variable

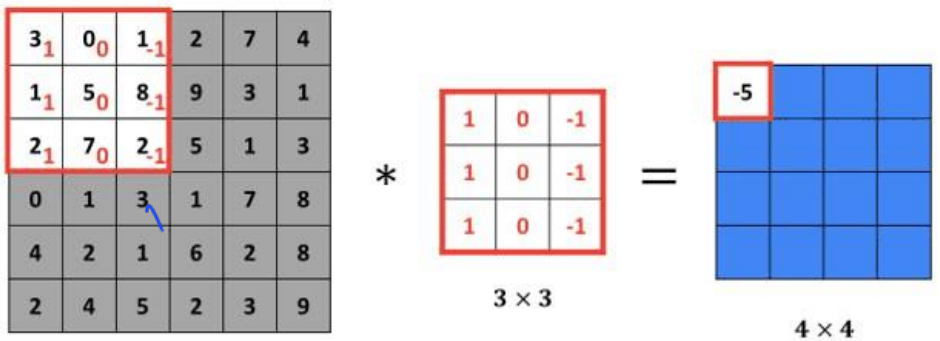
Make the kernels learnable

Treat the kernels as trainable weights

What about nonlinear patterns in images ? → curves, exponential, logarithmic, sinusoidal etc.

Add activation functions to the kernels

↳ used to add non linearity to the model.

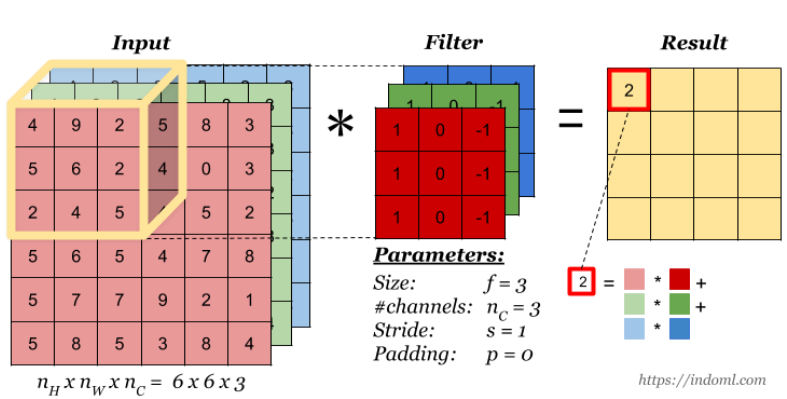


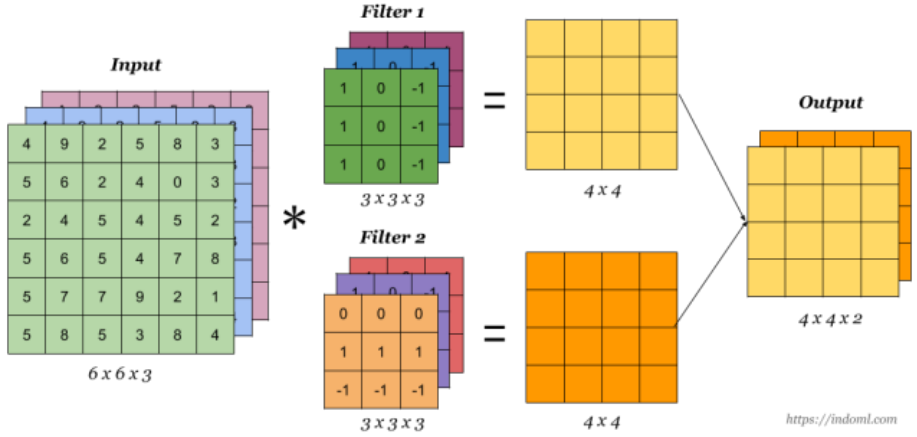
$3 \times 1 + 1 \times 1 + 2 \times 1 + 0 \times 0 + 5 \times 0 + 7 \times 0 + 1 \times -1 + 8 \times -1 + 2 \times -1 = -5$

Convolutions over Volumes

Convolutions on RGB image/over volumes

Note: Filter/Kernel depth should be the same as input image depth





CNNs: Understanding Dimensions

Input Dimensions: Width (W1) x Height (H1) x Depth (D1) [e.g. 224x224x3]

Spatial Extent of Filter/Kernel: F (depth of the filter is same as of input)

Output Dimensions: W2 x H2 x D2

Strides: S → How many spaces will you cover with every movement of your kernel across the input pixels.

Kernels: K

Padding: P

CNNs: Understanding Dimensions

Lets just consider the W1 and H1 for now. Suppose we have an image of size 5x5 (W1xH1) and kernel of size 3x3.

The result of convolution is 3x3

W2: W1 – F + 1 -> 5 – 3 + 1 = 3

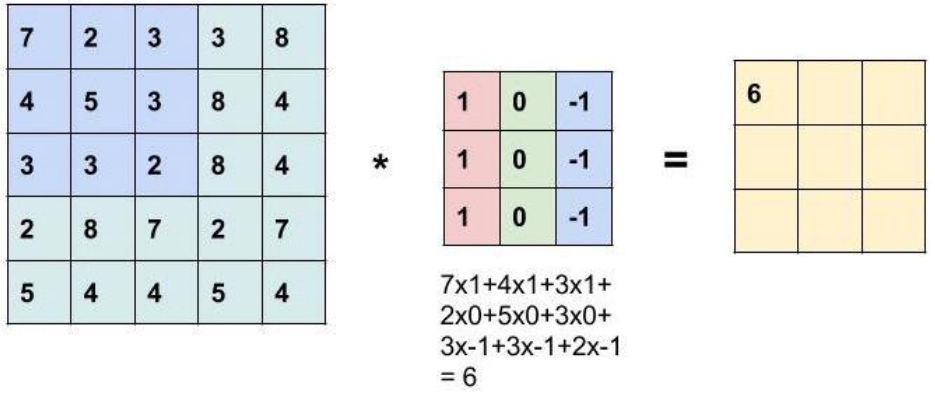
H2: H1 – F + 1 -> 5 – 3 + 1 = 3

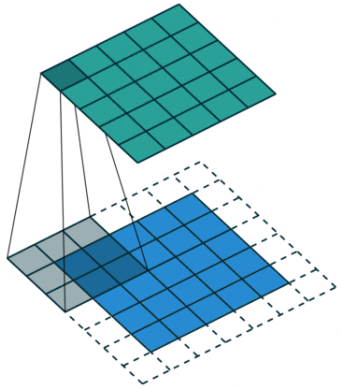
7x7
(W1 x H1)
W2 = 7-3+1 = 5
H2 = 7-3+1 = 5

Lets say we have a 7x7 image and kernel of size 3x3

7 – 3 + 1 = 5

Note: The output result dimensions are smaller than input dimensions





CNNs: Understanding Dimensions

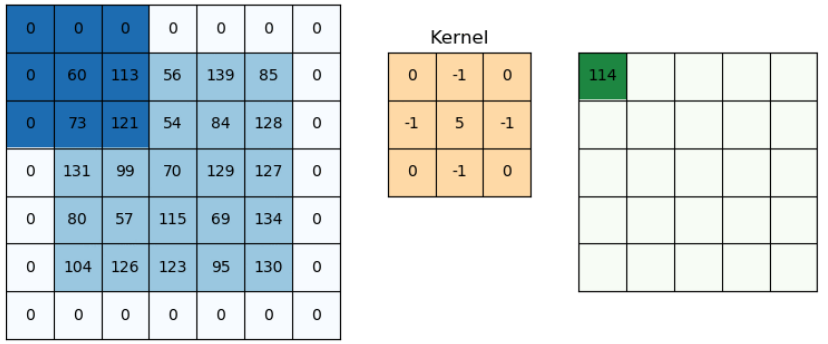
What if we want the output of convolution to be of same size as input ? padding

Pad input image with appropriate number of inputs so that you can now apply the kernel on input image corners as well. There are different methods for padding but the most commonly used one is zero padding. No. of Padding units (P)

If P is set to 1 then it will pad rows and columns of 0's to the top, bottom, left and right of the image

W2: W1 – F + 2P +1, **H2:** H1 – F + 2P +1

5 – 3 + 2x1 + 1 = 5



if p=2 : 2 layers of padding across bottom, top, left, right

CNNs: Understanding Dimensions

What does stride do ?

Defines the intervals at which the kernel is applied to the image

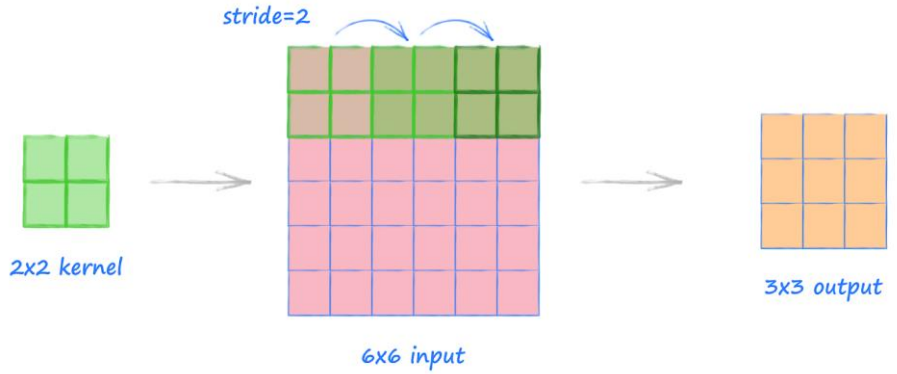
Results in an output of smaller dimensions

W2: [(W1 – F + 2P)/S] + 1

H2: [(H1 – F + 2P)/S] + 1

6x6 image, 2x2 kernel and stride 2

[(6 – 2 + 2x0)/2] + 1 = 3



CNNs: Understanding Dimensions

Lets talk about the depth of the output now.

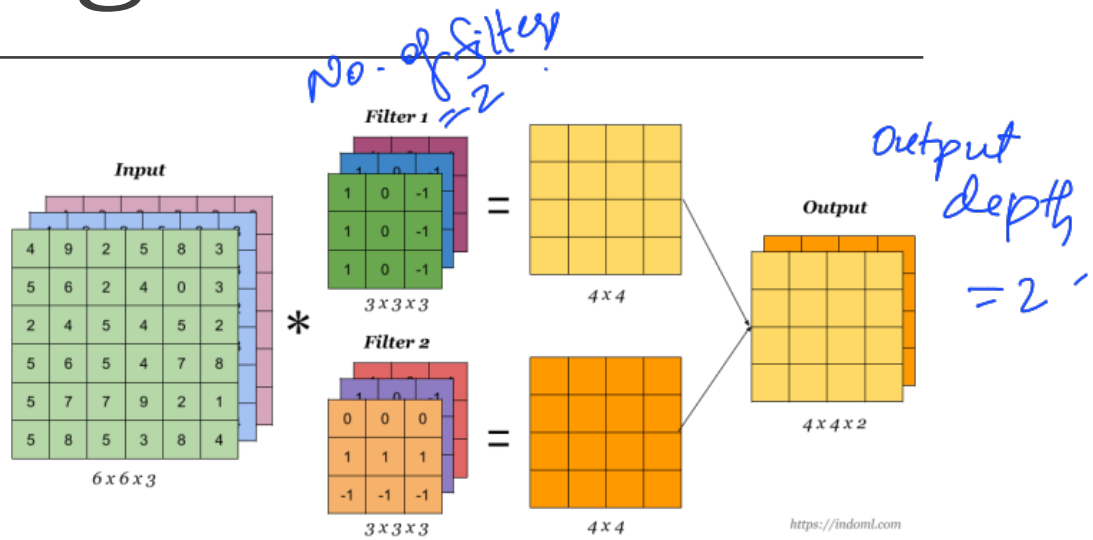
Each kernel/filter gives us a 2D output

If we use K filters then we will get K such 2D outputs

The resulting dimensions would then be W2xH2xK

Thus, D2 = K

output Depth = no. of filters applied



What were the challenges faced by ANNs aka FNNs in image tasks ?

Problems with this for image related tasks:-

1. Images have a 2D spatial structure, therefore considering only the flattened pixel values are not very useful features for tasks like classification
   - Pixels that spatially separated are treated the same way as pixels that are adjacent
2. No obvious way for network to learn features at different places in an input image (similar edges, corners or blobs)
3. Can get computationally expensive for large images

→ again cuz of the lack of 2D spatial structure

How do CNNs solve these challenges ?

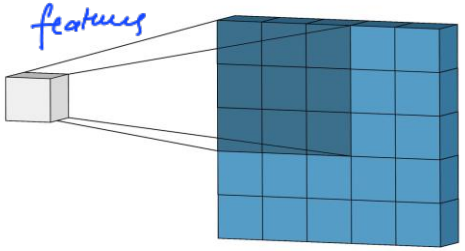
→ Local receptive fields: Each filter scans a small section of the input image, which is called local receptive field.

→ CNN uses local receptive fields to capture local patterns in the input image, such as edges, corners, textures, shapes.

→ When you stack multiple convolutional layers with different receptive fields & pooling layers, a CNN can learn hierarchical representation of input image, where higher level features are built upon lower level features.

using some sets of weights / filters / kernels to access different parts of an input image.

1. Local Receptive Fields: Hidden units are connected to the local patches from the previous layer. This serves 2 main purposes:-
   - Captures local spatial relationships in pixels
   - Reduces the number of parameters significantly



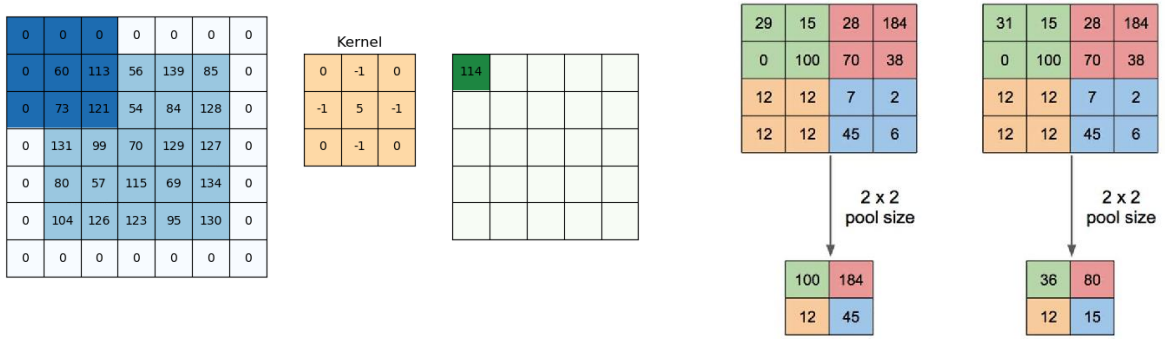
2. Weight Sharing: Also serves 2 purposes
   - Enables rotation, scale and translational invariance to objects in images (no more usage of the painful SIFT algo)
   - Reduces number of parameters in the model

used to downsample the feature maps produced by CNN, reducing its spatial dimension while retaining its important features.

3. Pooling: condenses info from previous layer
   - Aggregates info, especially the minor variations
   - Reduces size of output from previous layer, which reduces the number of computations in the previous layer

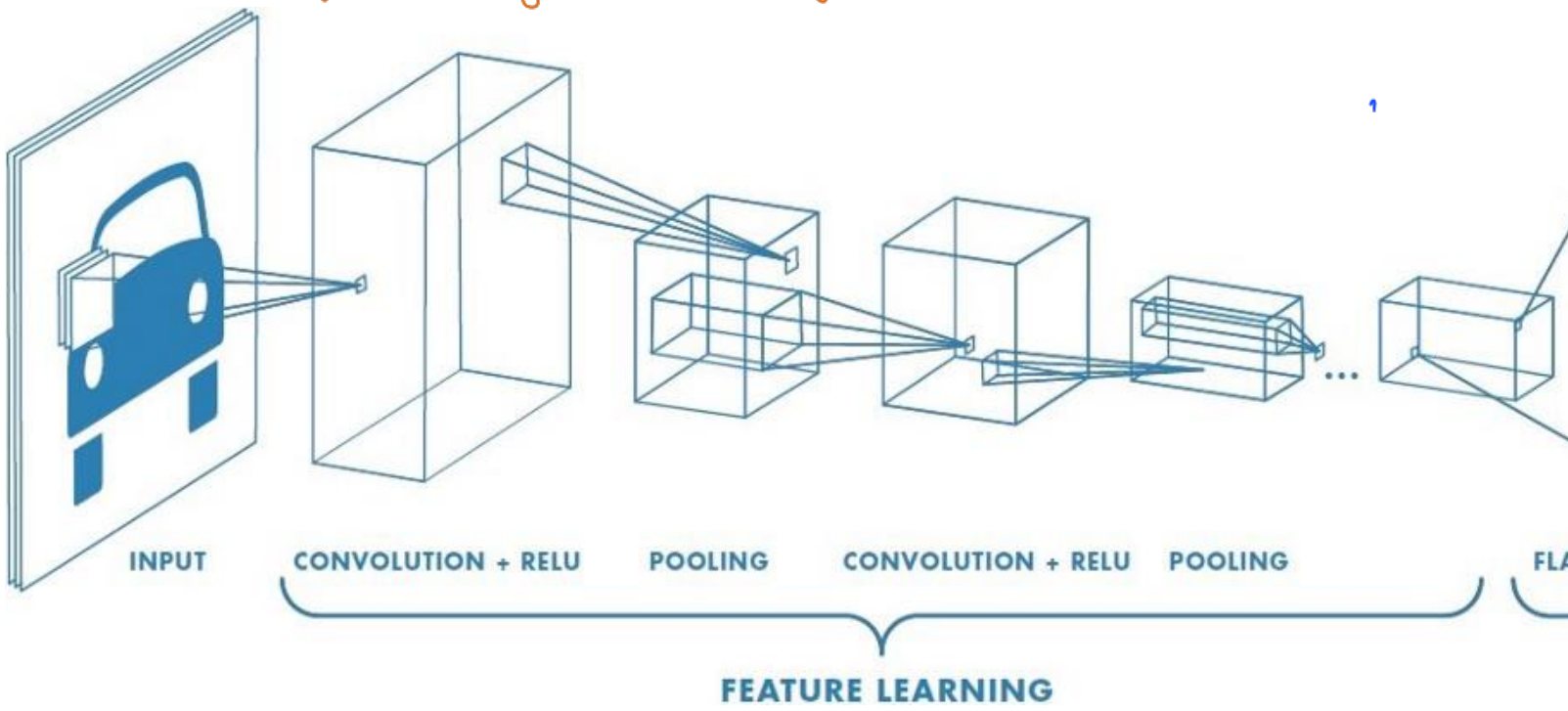
i) Max Pooling : Feature map is divided into non-overlapping regions & max value from each region is selected as output. Space decreases Impt. features retained.

ii) Avg. Pooling: Same as max pooling, except average value of each region is chosen as the output



What a typical CNN architecture looks like ?

Convolution layers are designed to extract features from the input image. They have to learn local features whereas Dense layers learn global patterns from features extracted by earlier layers



Dense Layers: every neuron in the layer is connected to every neuron in the previous layer which gives a dense, fully connected structure. This means that every output in the previous layer is combined with every weight in dense layer, resulting in a high dimensional output.

- They allow the model to learn complex non-linear relationships between features extracted by earlier layers.
- They perform the final classification or regression task.
- They take the output of previous layers & apply a set of learnable weights to produce final output.

How do CNNs go about learning patterns ?

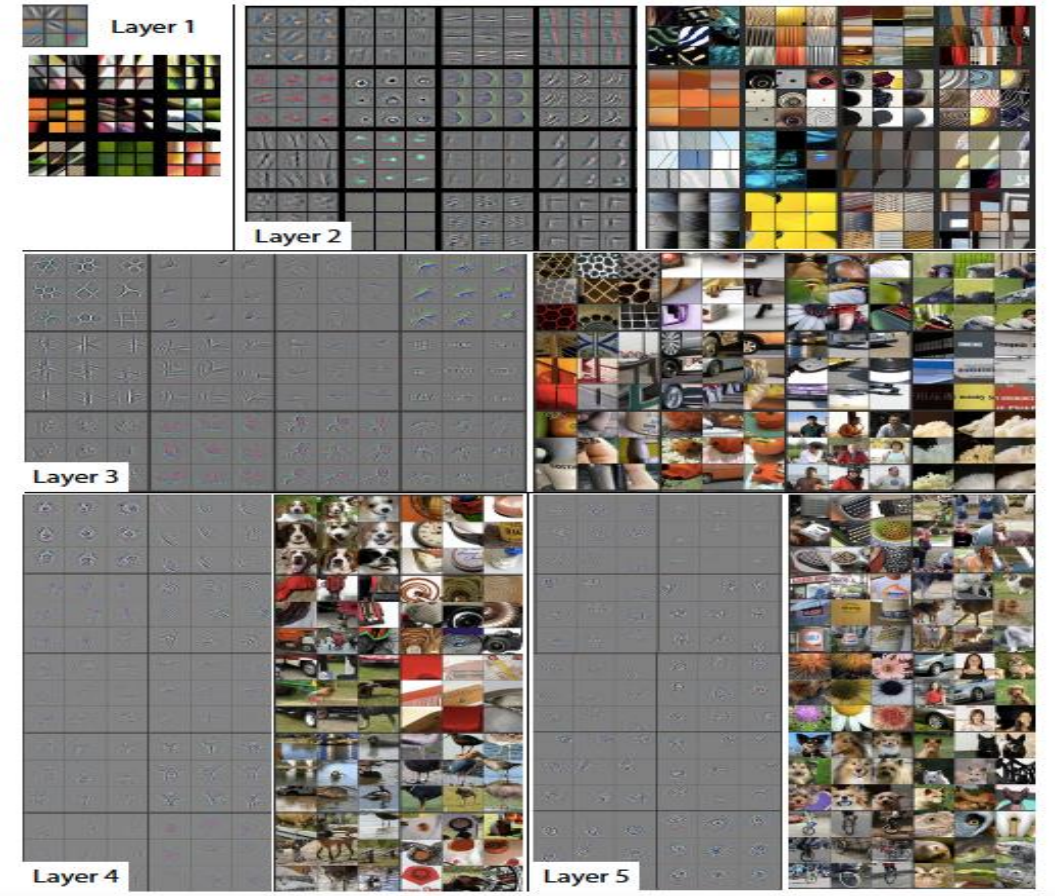
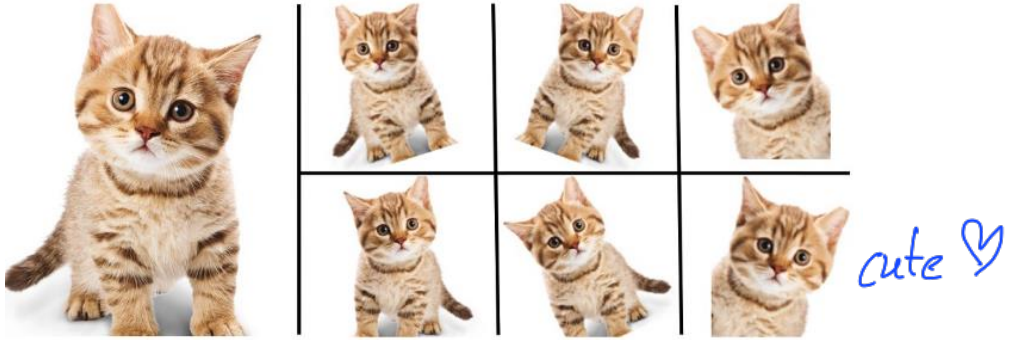


Image Augmentation

Image augmentation is **a technique that is used to artificially expand the data-set**. This is helpful when we are given a data-set with very few data samples. In case of Deep Learning, this situation is bad as the model tends to over-fit when we train it on limited number of data samples.

We use both image filtering and image transformations (affine transformations) to expand the dataset.

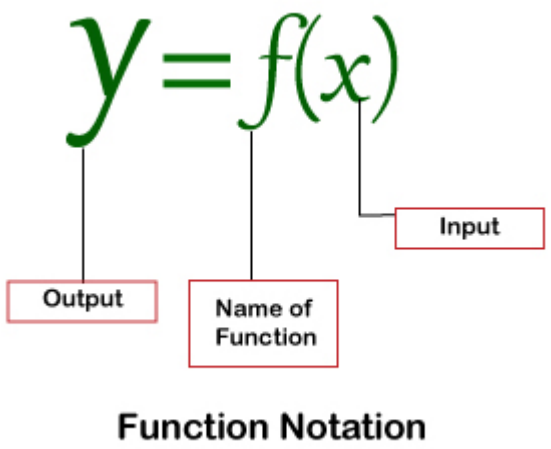
Since we know that the weight sharing property of the CNNs make the network rotation, scale and translational invariant we can easily leverage these transformation methods.

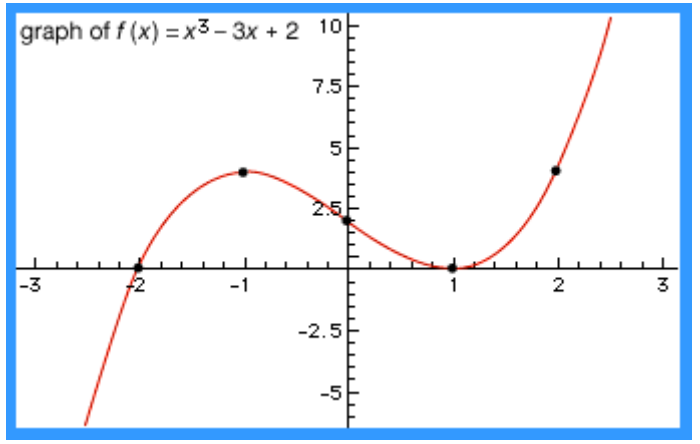


Diagnosis of CNNs

1. Use Classification Metrics
2. Observe the high and low probability scores
3. Observe the feature maps
4. Extract the features from the second last layer
5. Perform 2D or 3D plots to visualize them in the x-y(z) coordinate space
    1. Requires dimensionality reduction → Max / Avg pooling
6. Advanced visualization methods
    1. Maximal activated image patches
    2. Gradient based class activation maps

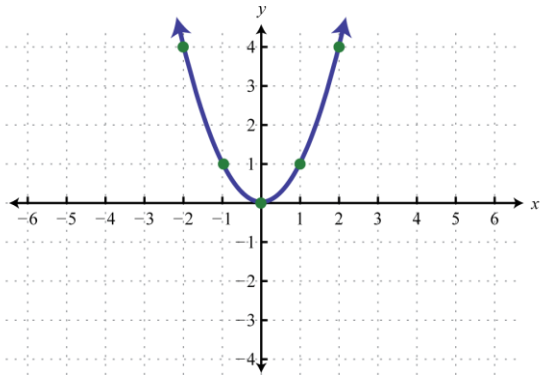
Neural Networks as Functions





$f(x) = x^2$

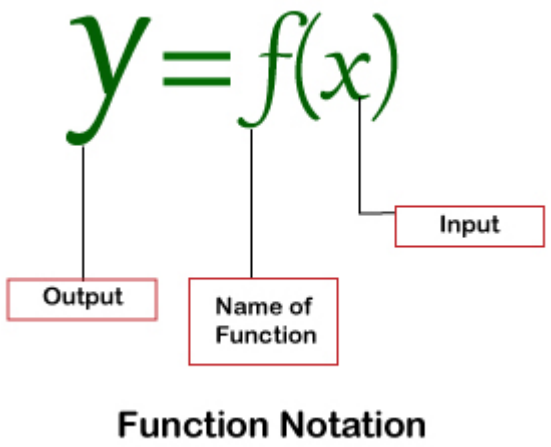
x	$f(x)$
−2	4
−1	1
0	0
1	1
2	4



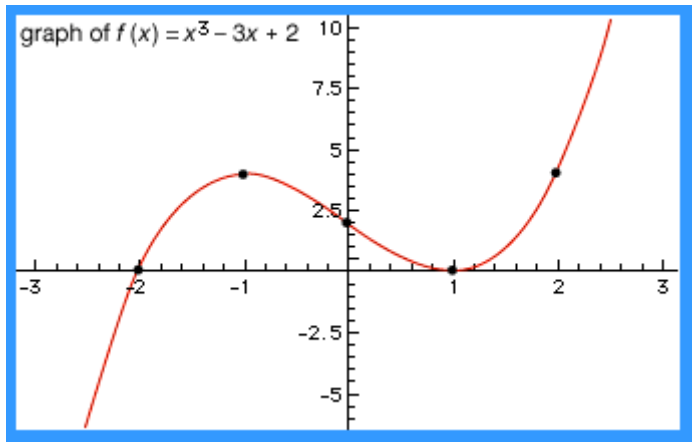
→ Trainable: parameters used by the model are learned during a training process, rather than being fixed in advance. This allows the model to adapt to data and learn to make better predictions over time.

Neural Networks as Functions

→ Parametric: model has a fixed number of parameters that are learned during training. These parameters decide the behaviour of the model & are adjusted to minimize loss function.

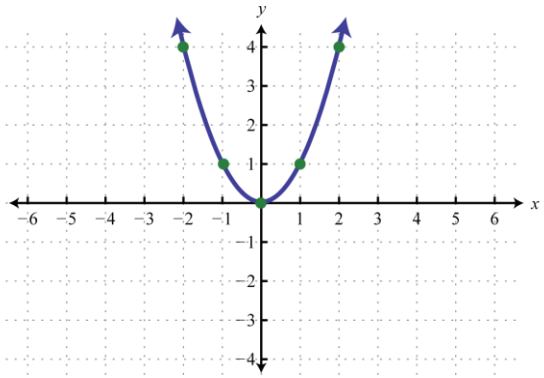


Function Notation



$f(x) = x^2$

x	$f(x)$
−2	4
−1	1
0	0
1	1
2	4



→ non-linear: The model can represent complex, non-linear relationships b/w input & output. It allows the model to learn more complex patterns than a simple linear function

Note: Neural Networks are nothing but trainable parametric non linear functions. The complexity of the function depends on the number of layers of the neural network

Overfitting in Neural Nets

What happens when a function is overtrained on same instances of data ? It overfits on those values and fails to generalize on new instances of data.

Solution: -

1. Regularization
   1. L1 and L2, ElasticNet
   2. Dropout
2. Addressing the difference in data distributions
3. Adding more data (diverse kind not the repeated instances)
4. Stop being greedy and stop the training a little earlier

→ L1 → Lasso → makes the model more simple by reducing the weights of less important features towards zero. Useful when there are many features.

→ L2 → Ridge → same as Lasso but it does not force the weight of any feature to become zero. Useful when all features are important, but some have a high Variance.

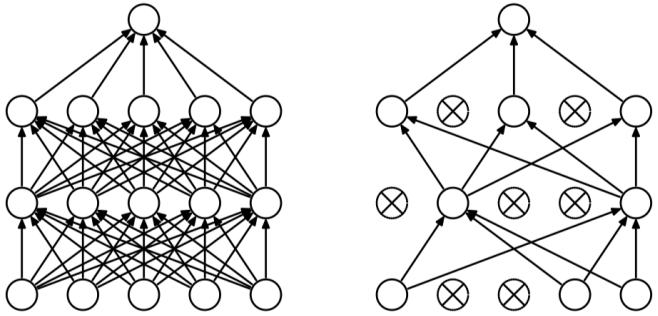
Balancing

Early Stopping.

Dropout

The term “dropout” refers to dropping out the nodes (input and hidden layer) in a neural network. All the forward and backwards connections with a dropped node are temporarily removed, thus creating a new network architecture out of the parent network. The nodes are dropped by a dropout probability of p. E.g. if a hidden layer has 8 neurons and the dropout rate p is 0.5, then 4 neurons will be randomly dropped of from that hidden layer.

Another way to think of it is to realize that a unique neural net is generated at each training step. The final neural network during the inference time will be nothing but the average ensemble of these smaller reduced neural nets. It has the same intuition as Random Forest.



(a) Standard Neural Net (b) After applying dropout.

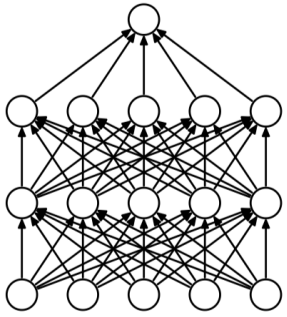
Dropout

Dropout solves the problem of overfitting which is prominently caused by the problem of co-adaptation.

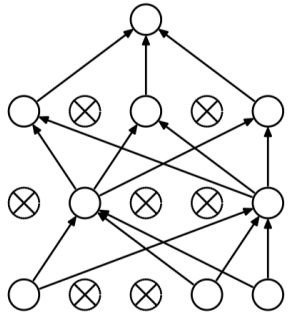
In neural network, co-adaptation means that some neurons are highly dependent on others. If those independent neurons receive “bad” inputs, then the dependent neurons can be affected as well, and ultimately it can significantly alter the model performance, which is what might happen with overfitting.

toxic co-dependency

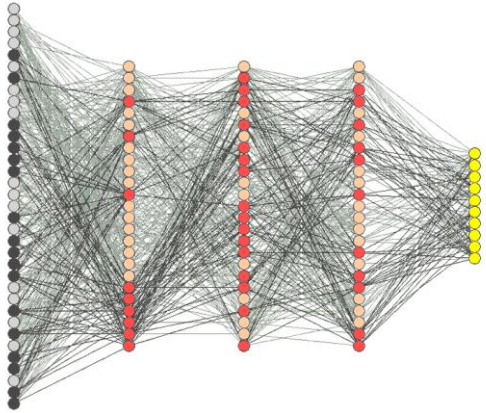
babes!



(a) Standard Neural Net



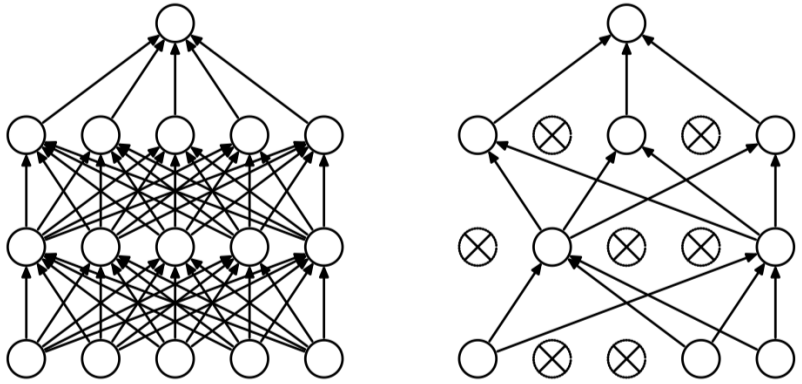
(b) After applying dropout.



used after Dense layer, before output layer.

Dropout During Inference (Test/Val)

During inference (testing/validation) dropout layer is not used. This means that we consider all the neurons of the original/parent neural network. But due to this the final weights of the neural network will be larger than expected. Therefore the weights are scaled by multiplying p the dropout rate with the weights for each unique neurons dropped out neural network and then summed together to get the final weights of the neural network at inference time.

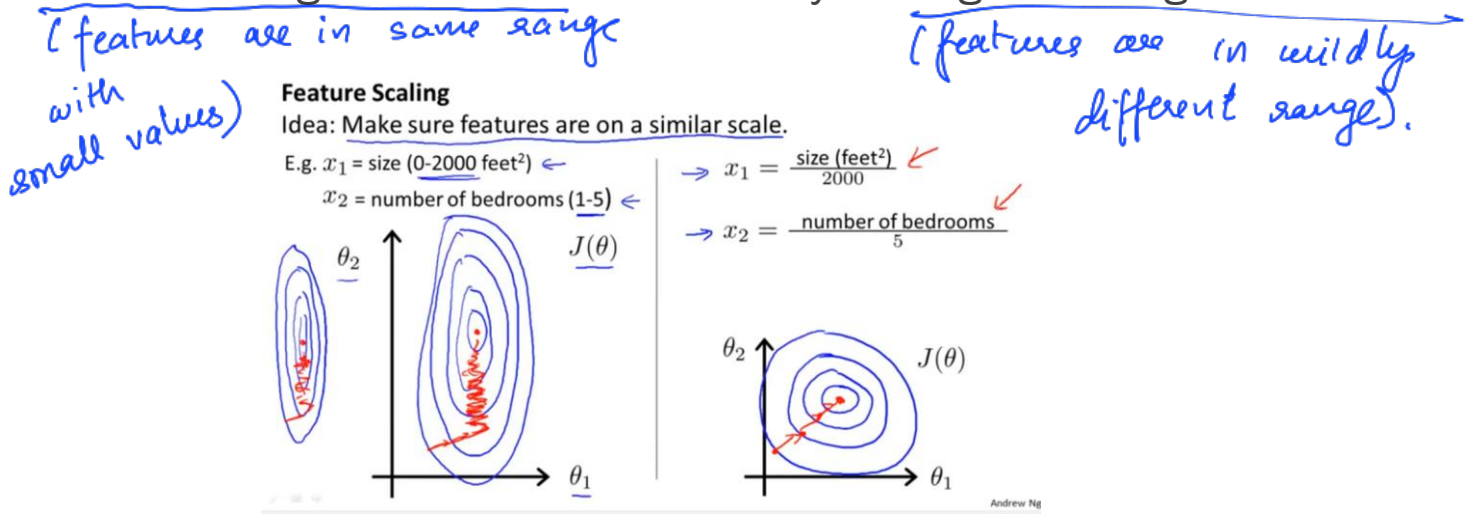


(a) Standard Neural Net

(b) After applying dropout.

Why Feature Scaling is Important ?

We need **input feature scaling** in parametric methods like neural networks because the parameters descend quickly on smaller homogeneous scales and slowly on larger heterogeneous scales.



But what about the scales of features of the hidden layers of the neural network ?

→ It can cause the network to converge more slowly or even not converge at all.

• parameters of each layer are updated based on error calculated by previous layer.
• parameters (weights/biases/act.fns) of each layer changes → distribution of inputs (normally distributed, uniformly distributed, etc.) for next layer also changes
• This makes the network more difficult to train.

Internal Covariate Shift

- "We define Internal Covariate Shift as the change in the distribution of network activations due to the change in network parameters during training." – Authors of **Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift.**
- In neural networks, the output of the first layer feeds into the second layer, the output of the second layer feeds into the third, and so on. When the parameters of a layer change, so does the distribution of inputs to subsequent layers.
- These shifts in input distributions known as **Internal Covariate Shifts** can be problematic for neural networks, especially deep neural networks that could have a large number of layers.

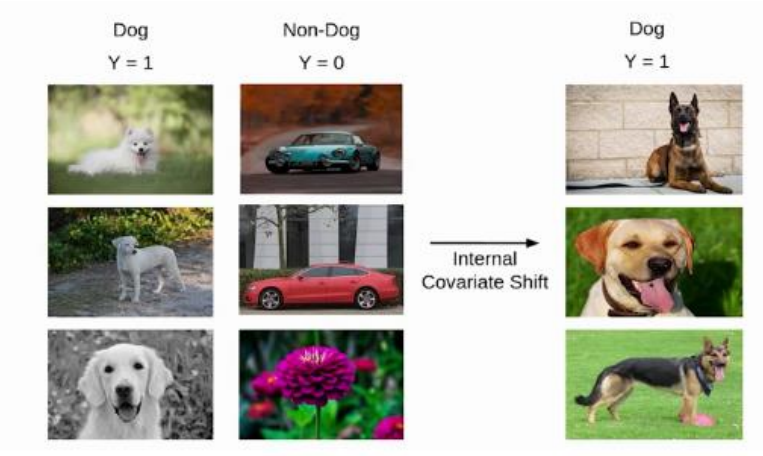
Example of Internal Covariate Shift

Lets say we are solving a binary classification problem of Dogs vs..... Not Dogs ?

Let's say we have the images of white dogs only in a certain batch, these images will have certain distribution as well. Using these images model will update its parameters.

Later when we introduce a new set of images consisting of now white dogs, they will have a different distribution from the previous images.

Hope: *hemk I is part of distribn*



Coco: *wut the floof, demn you internal covariate shift*

→ In NN, the inputs in each layer are not normalized, which makes the network difficult to train.
→ Batch Norm. fixes this problem by normalizing these inputs across a batch of training examples.

Batch Normalization

→ For each mini-batch, the mean & variance of input to each layer is calculated.
→ The inputs are normalized by subtracting the mean and dividing by standard deviation.

This technique adds an operation in the model before or after the activation function of each hidden layer.

This operation simply zero centers and normalizes each input then scales and shifts the result using two new parameter vectors per layer

- One for scaling
- The other for shifting

→ The normalized inputs are then scaled & shifted using learned parameters. This allows the network to learn the optimal shift & scale for each layer.

→ Reduces internal cov. shift, leads to faster conv-ergence, better generalization & more stable training of neural network.

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Batch Normalization during Testing/Validation

During Training Batch Normalization standardizes intermediate inputs, then rescales and offsets them.

What about Test/Validation Time ?

We may be passing just one instance of image at test time, or maybe a smaller batch of images so we may have no way to compute mean and SD or we may have to compute the mean and SD over a smaller batch/sample.

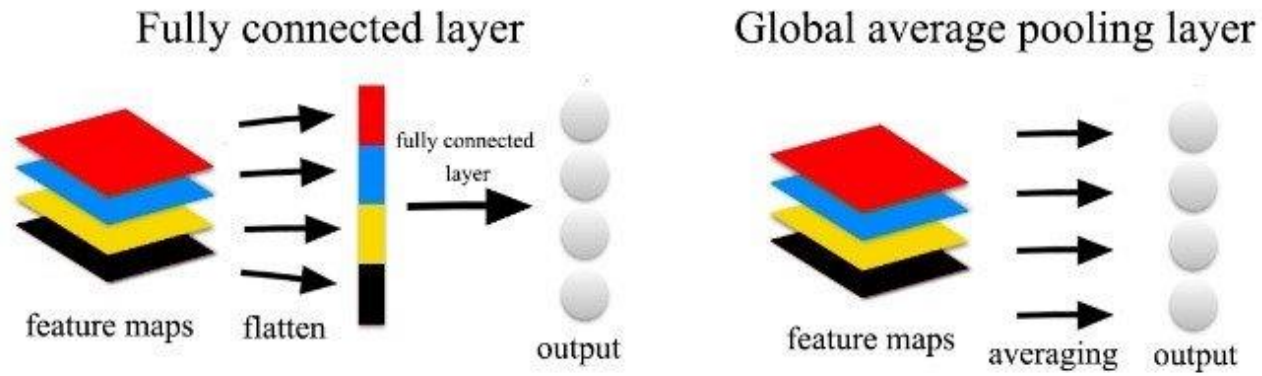
Most DL frameworks estimate the mean and SD for test and validation by performing a moving average of layer's input means and SD then uses this mean and SD for BatchNormalization during the test/validation

Global Average Pooling

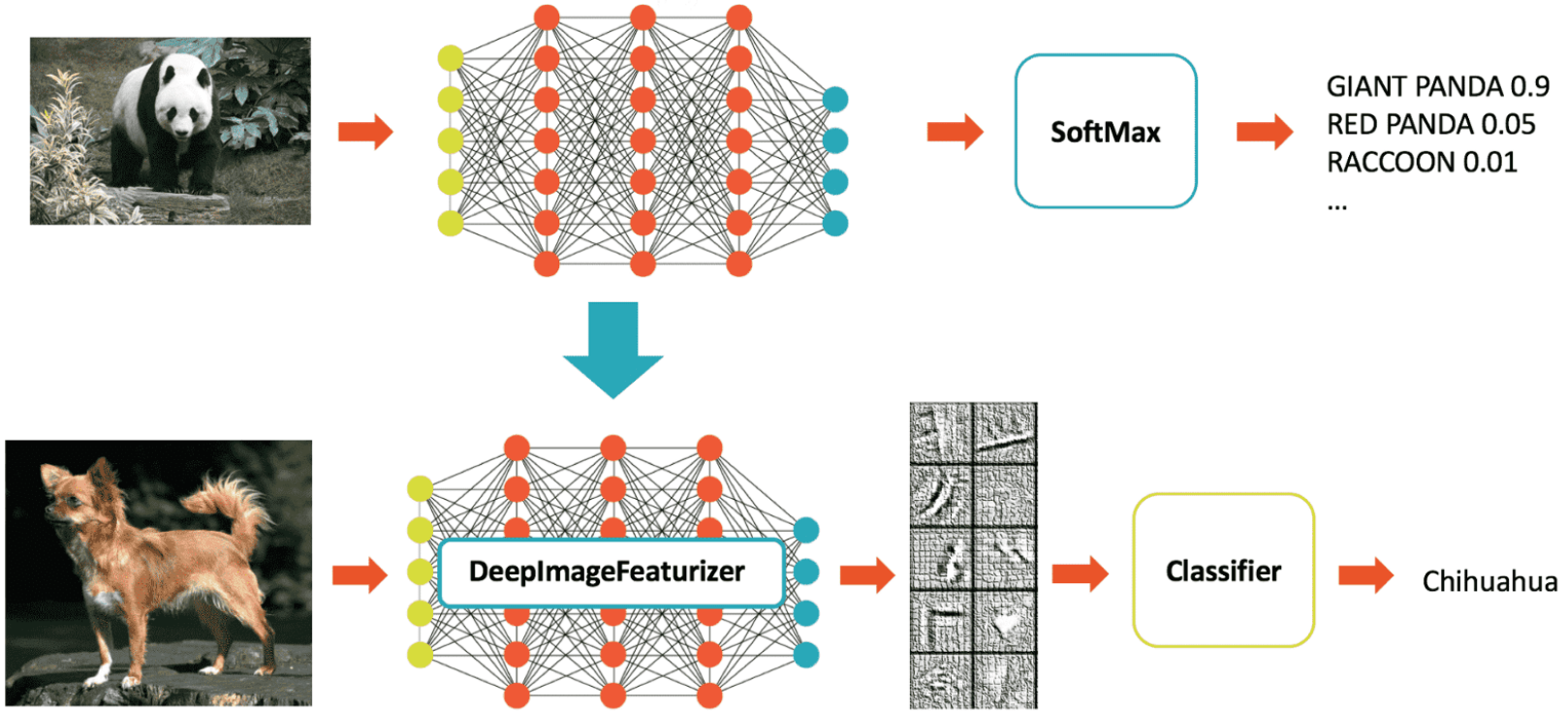
Reduces the increase in number of params caused by flattening layer

Takes an average of all the feature maps of the layer

E.g. if the last conv layer in the network has 512 filters thus producing 512 feature maps, the global average pooling will produce an output vector of 512



Transfer Learning

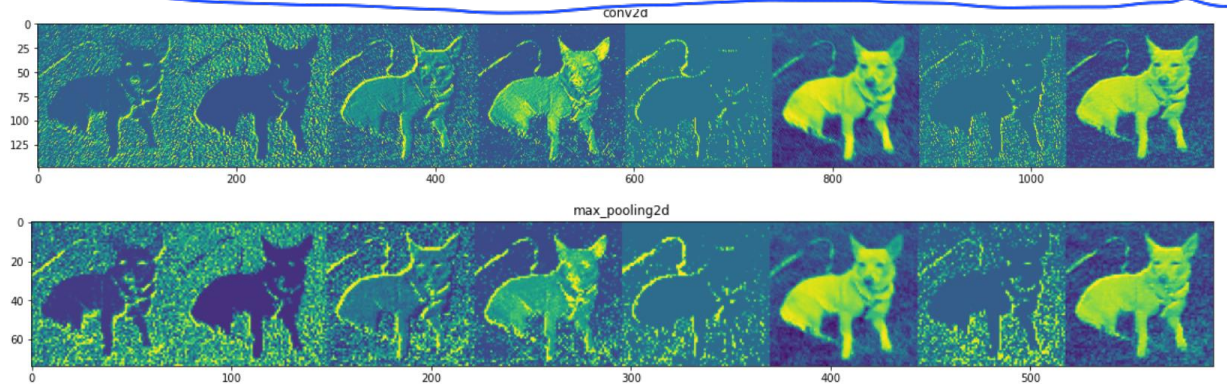


Transfer Learning

1. Transfer Learning is the transfer of the knowledge (feature maps) that the network has acquired from one task, where we have large amount of data, to a new task where the data is not abundantly available.
2. Generally used where a neural network model is trained on a problem similar to the problem that is being solved.
3. The intuition behind transfer learning is that if a model is trained on a large and general enough dataset, the model will effectively serve as a generic function for that problem.
4. We use the feature maps that this model has learnt, and without having to train a new model on a large dataset we can transfer the learning of the trained model's knowledge to our model and use it as a base starting model for our own task

How Transfer Learning works ?

1. What is really being learned by a CNN during training ? Feature Maps
2. How are these features learned ? During backpropagation process the weights are updated until we get to the optimized weights that minimize the error function
3. What is the relationship between features and weights ? Feature maps are a result of passing the weights filter/kernel on the input image during the convolution process
4. What is really being transferred from one network to another ? To transfer features we use the optimized weights of the pretrained network and reuse them as the starting point for the training process to adapt to our new problem



One Problem to Rule them all: ImageNet

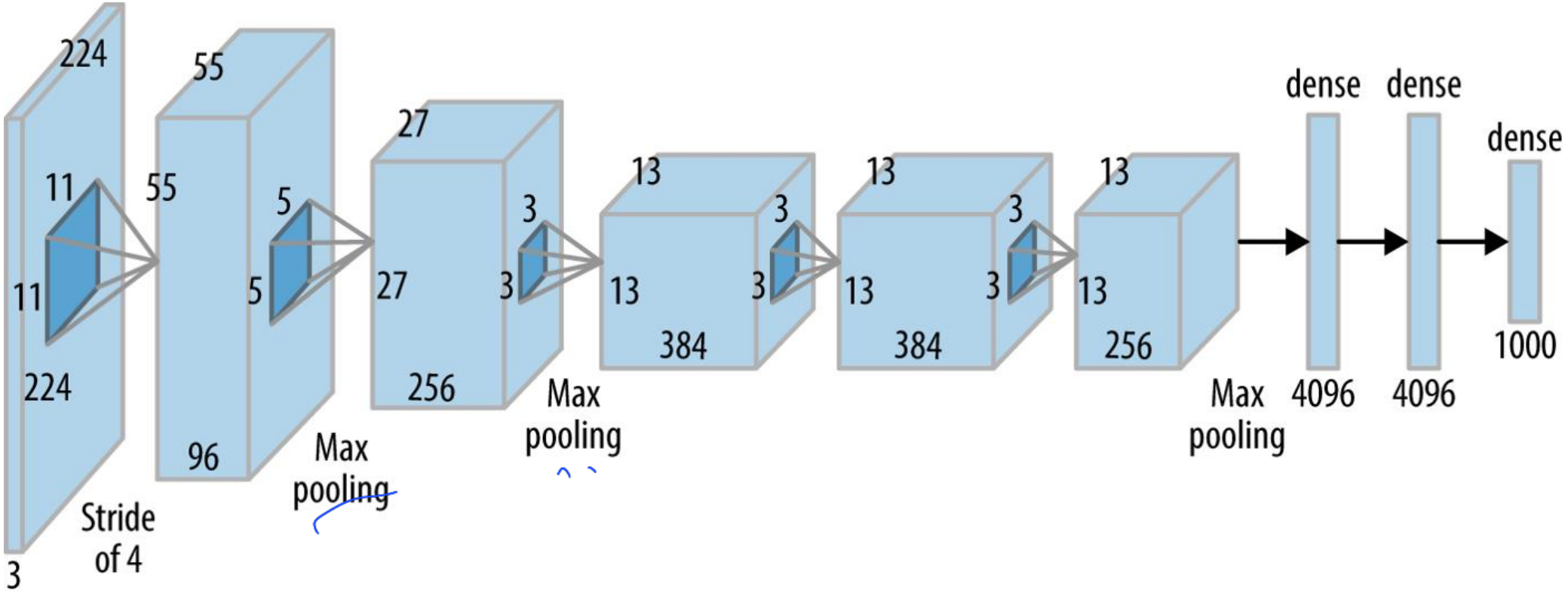
ImageNet Large Scale Visual Recognition Challenge

1. The ImageNet Large Scale Visual Recognition Challenge (ILSVRC) evaluates algorithms for object detection and image classification at large scale. One high level motivation is to allow researchers to compare progress in detection across a wider variety of objects -- taking advantage of the quite expensive labeling effort. Another motivation is to measure the progress of computer vision for large scale image indexing for retrieval and annotation.

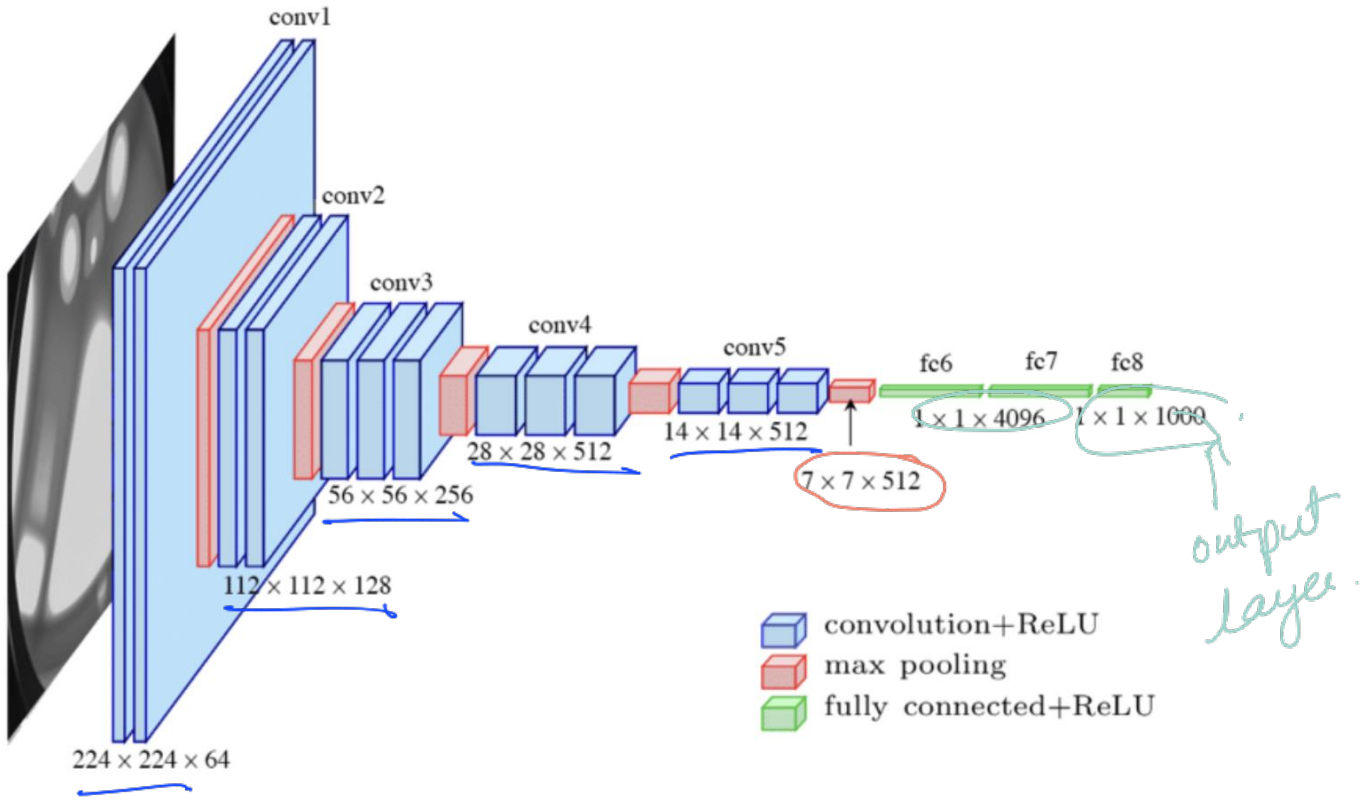
Transfer Learning Approaches

1. Using a pretrained network as a classifier
2. Using a pretrained network as a feature extractor
3. Finetuning

AlexNet

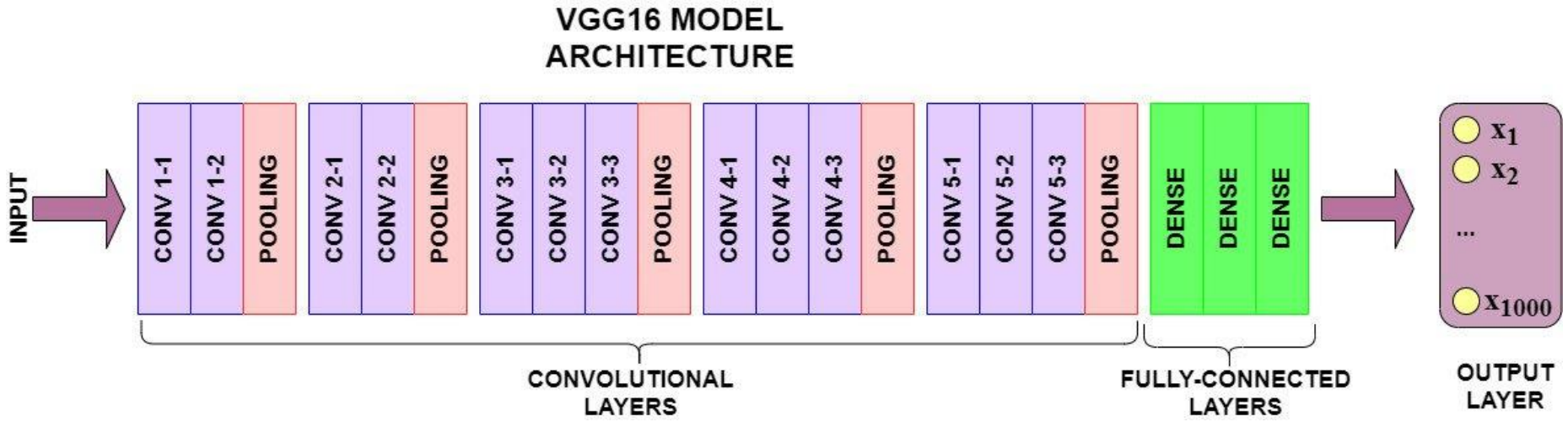

224
11
11
224
3
Stride
of 4
55
55
96
5
5
Max
pooling
27
27
256
3
3
Max
pooling
13
13
384
3
3
13
13
384
3
3
13
13
256
Max
pooling
dense
4096
dense
4096
dense
1000

VGG-16


conv1
conv2
conv3
conv4
conv5
fc6
fc7
fc8
224 × 224 × 64
112 × 112 × 128
56 × 56 × 256
28 × 28 × 512
14 × 14 × 512
7 × 7 × 512
1 × 1 × 4096
1 × 1 × 1000
output layer
convolution+ReLU
max pooling
fully connected+ReLU

Pretrained Network as a Classifier

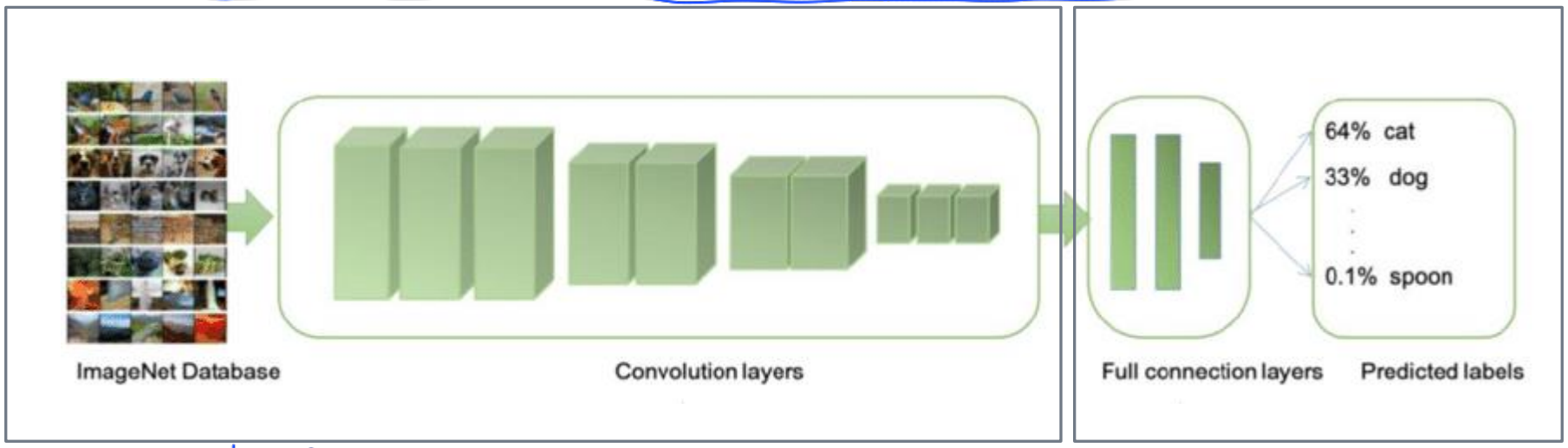
Produces the class output from the data labels it has been trained on



This is what we are doing.

Pretrained Network as a Feature Extractor

1. Use the convolutional layers of the trained model to extract features
2. Remove the classifier of the pretrained network and add your own classifier



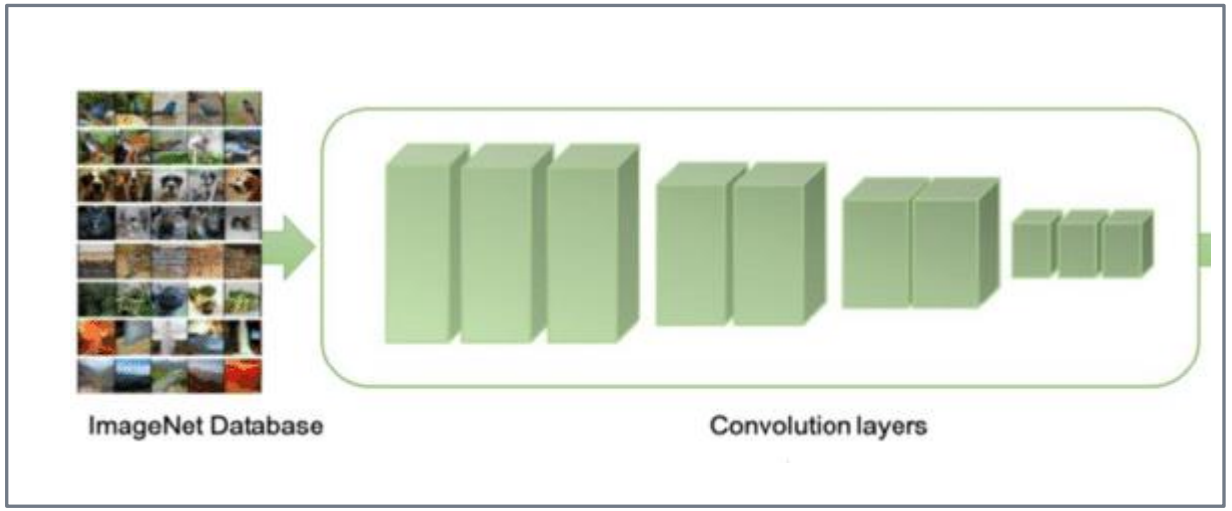
So we don't have to train our own model.

Freeze the layers and use pretrained weights

Add a classifier on top, fully connected layers + decision neuron

Pretrained Network as a Feature Extractor

Use the convolutional layers of the trained model to extract features. What if you could use some other classifier instead of a fully connected layers + decision neuron (which is basically a NN) ? What if you could use a Naïve Bayes, SVM, RandomForest or a XGBoost Classifier



SVM
Naïve Bayes
Random Forest
XGBoost
Etc.

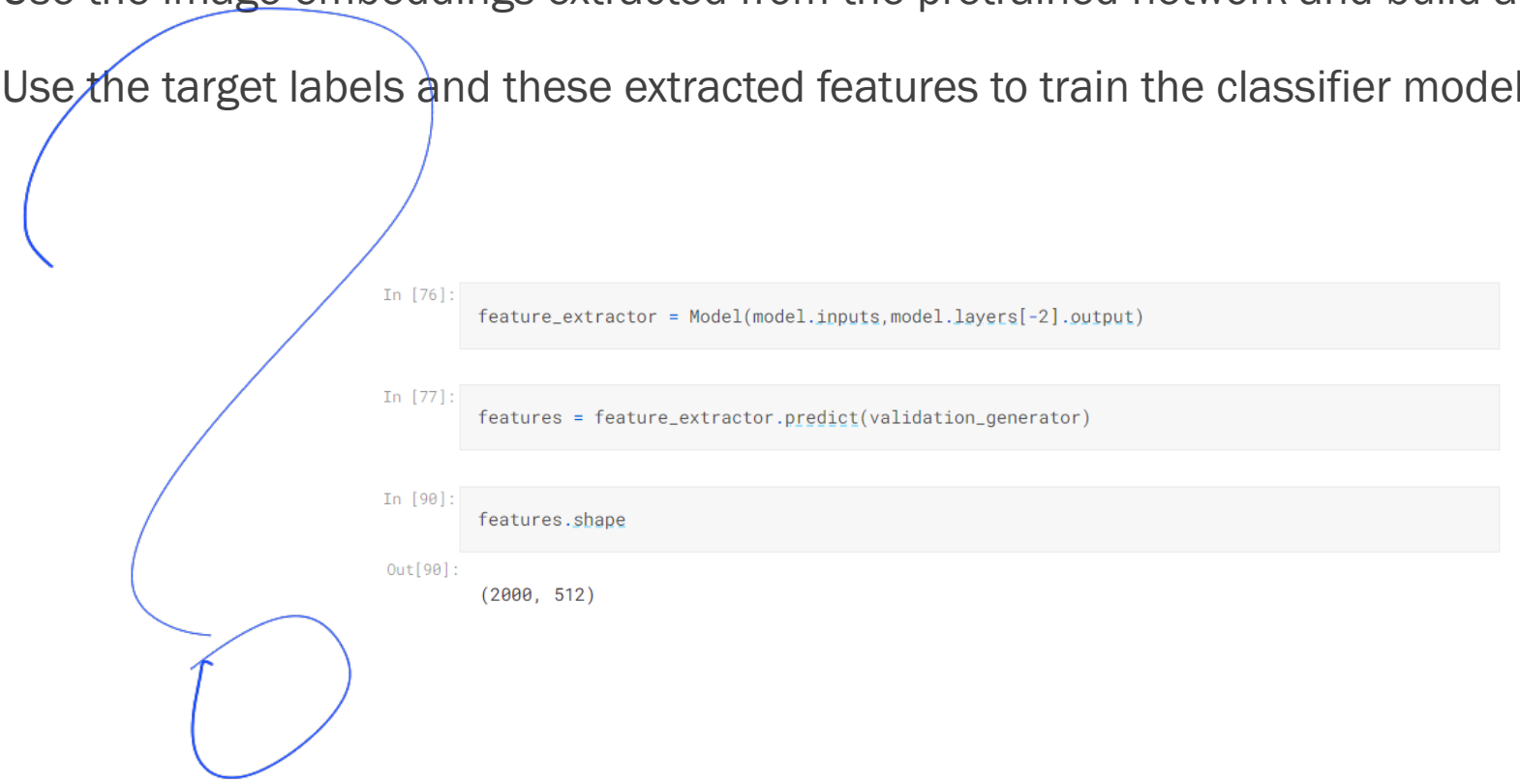
So pre traind models can be used with DL as well as ML algorithms.

Freeze the layers and use pretrained weights

Add a classifier on top

Pretrained Network as a Feature Extractor

1. Use the image embeddings extracted from the pretrained network and build a tabular dataset
2. Use the target labels and these extracted features to train the classifier model like you usually do

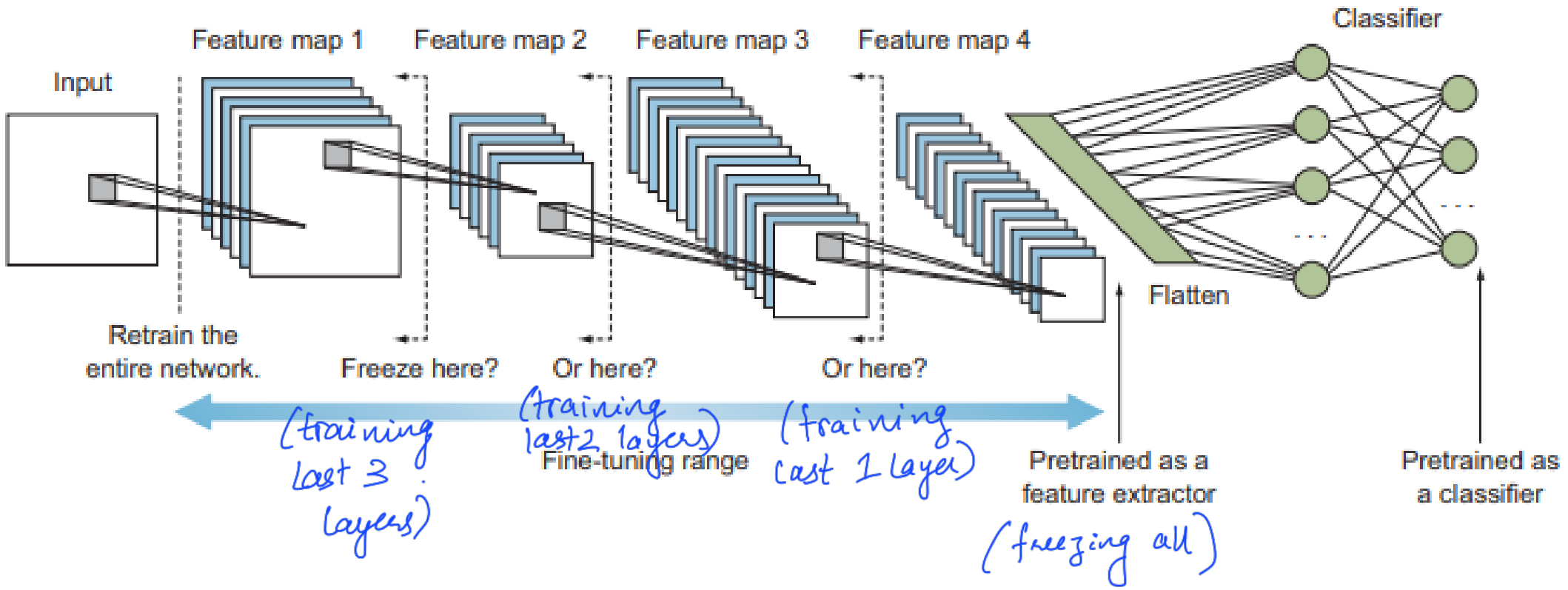


```
In [76]: feature_extractor = Model(model.inputs,model.layers[-2].output)

In [77]: features = feature_extractor.predict(validation_generator)

In [90]: features.shape

Out[90]: (2000, 512)
```

Fine Tuning

1. So far we have seen how to use a pretrained network directly for classification or as a feature extractor. We generally use these approaches when the domain of the target dataset (problem we are interested in solving) is somewhat similar to the source domain (imagenet problem)
2. But what if the target domain is different from the source domain or if its very different ?
3. We just extract the correct feature maps from the source domain pretrained network and fine tune them to the target domain. This is known as fine tuning.
4. Fine tuning can be formally defined as freezing a few of the network layers that are used for feature extraction and transferring all of its high level feature maps (outputs of later convolutional layers) to your domain.

So how many layers should we freeze ?



small dataset → training on too many layers may lead to overfitting → stick to pretrained weights

Choosing the appropriate level of Transfer Learning

Choosing the level of transfer learning depends on two important factors:-

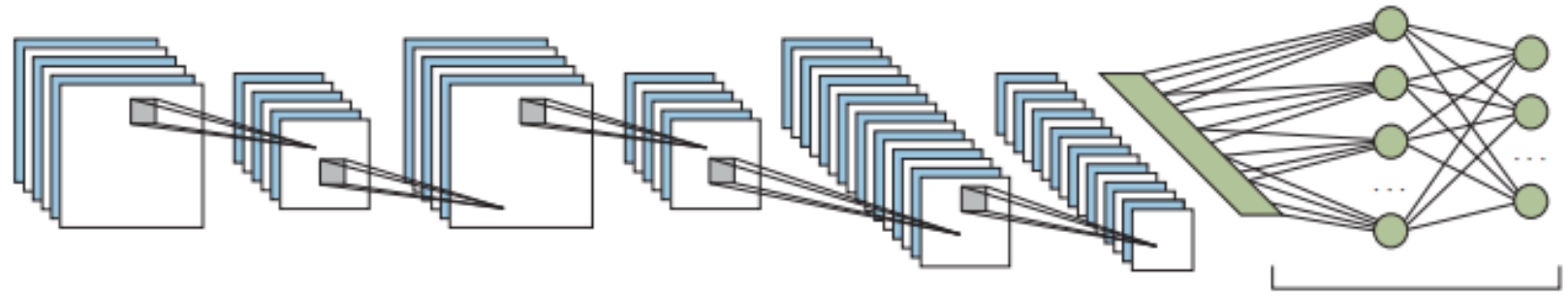
1. Size of the target dataset (small or large): When we have a small dataset, the network probably won't learn much from training on larger number of layers, so it will tend to overfit the new data. In this case we want to rely more on the pretrained weights of the source dataset
2. Domain similarity of the source and target datasets: Is the target problem similar or different than the source problem ?

These two factors lead to 4 different scenarios

Choosing the appropriate level of Transfer Learning

Scenario	Size of the target data	Similarity of the original and new datasets	Approach
1	Small	Similar	Use pretrained network as a feature extractor
2	Large	Similar	Freeze 60-80% of the network and fine tune the rest of the network
3	Small	Very Different	Since target dataset is different it might not be the best to freeze the higher level features of the pretrained network because they contain source dataset specific features. It will be better to retrain layers from somewhere early in the network. Freeze 30-50% of the network.
4	Large	Very Different	Fine tune the entire network/Train the entire network by using the existing weights of the pretrained network

Choosing the appropriate level of Transfer Learning



Scenario #1: You have a small dataset that is similar to the source dataset.

Scenario #2: You have a large dataset that is similar to the source dataset.

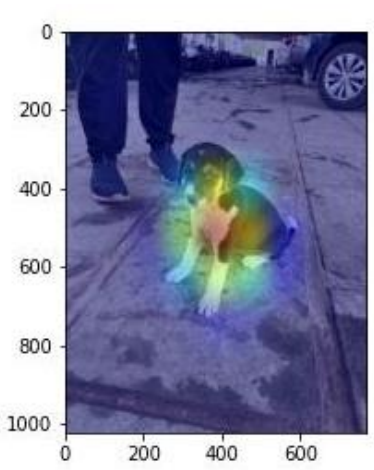
Scenario #3: You have a small dataset that is different from the source dataset.

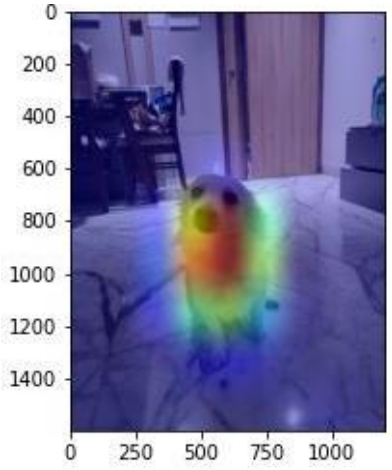
Scenario #4: You have a large dataset that is different from the source dataset.

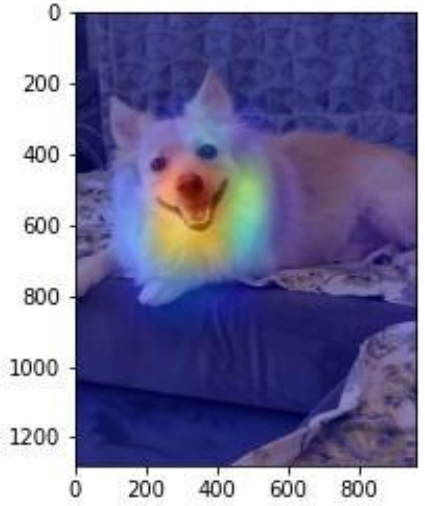
Transfer Learning and Learning Rates

Note: Learning rates should be kept small when transfer learning is applied, reason being you do not want the trainable layers' weights to change drastically during your training process.

Grad-CAM: What do CNNs learn?







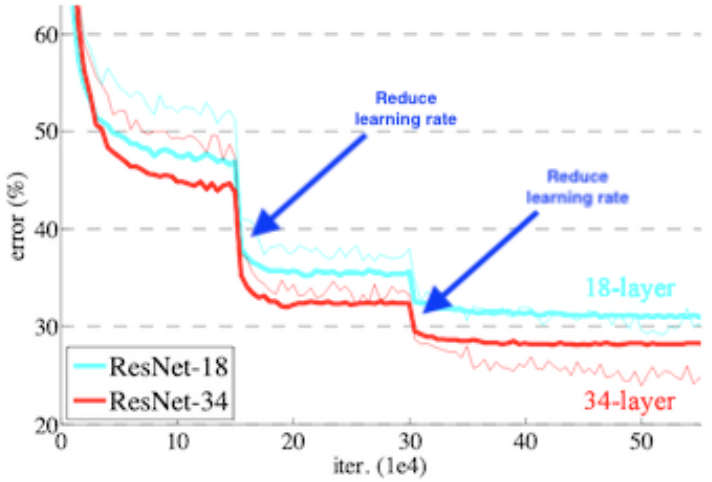
Miscellaneous

Model Callbacks

EarlyStopping

Checkpointing

ReduceLROnPlateau



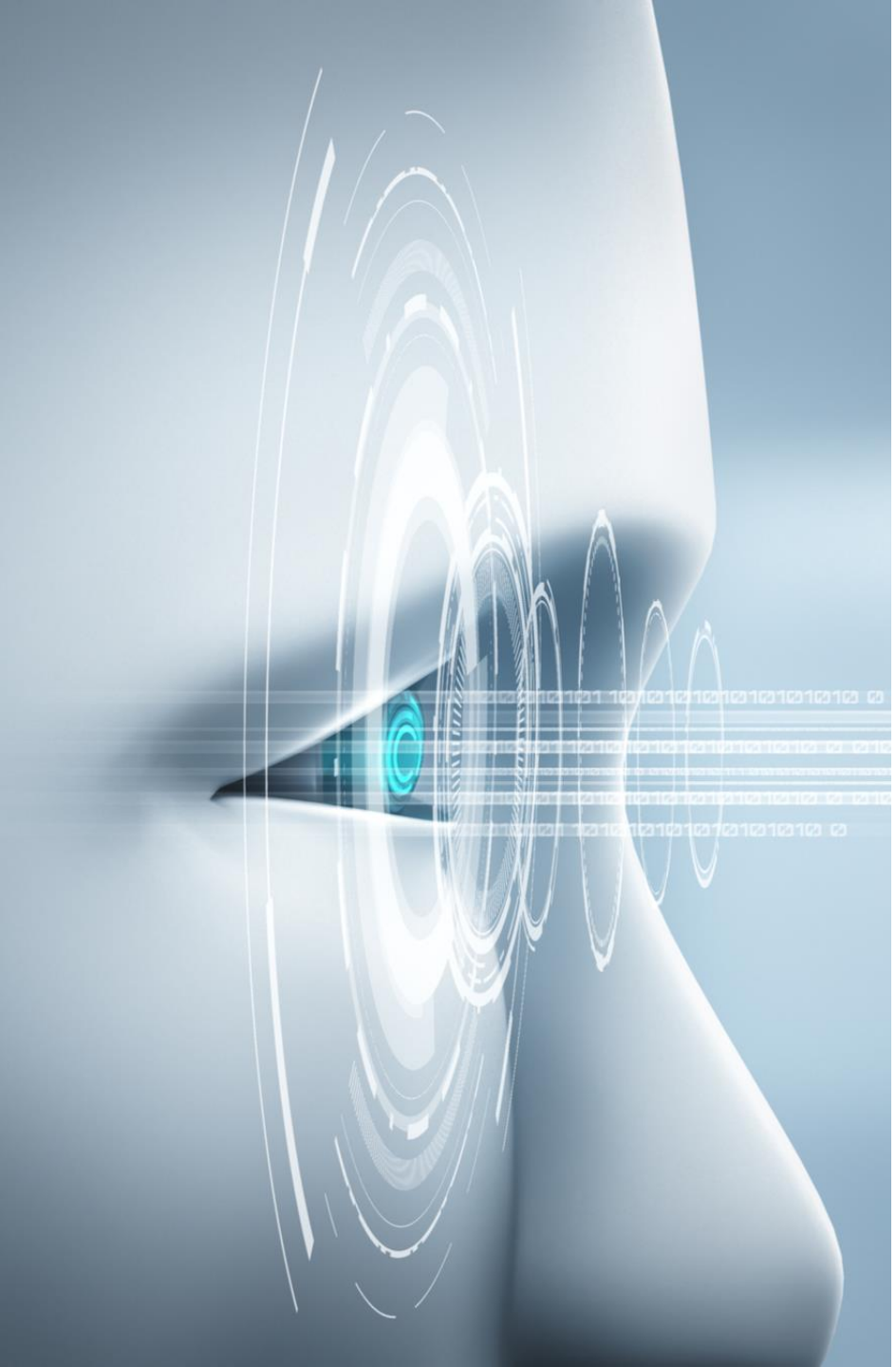
ROC and AUC

https://towardsdatascience.com/demystifying-roc-curves-df809474529a

https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5

Thank you

For any queries drop an email at: quadeershaikh15.8@gmail.com



Computer Vision Applications

BY QUADEER SHAIKH

About me

Work Experience

- Risk Analyst
  - Morgan Stanley (Jan 2023 – Present)
- Data Science Intern
  - AkzoNobel Coatings International B.V. Netherlands (Feb 2022 – Dec 2022)
- Data Science Intern
  - EzeRx Health Tech Pvt. Ltd. (Jan 2022 – July 2022)
- Associate Engineer
  - Tata Communications Ltd. (July 2019 – Aug 2020)
- Network Automation and Analysis Engineer Intern
  - Cisco (June 2018 – July 2018)

Education

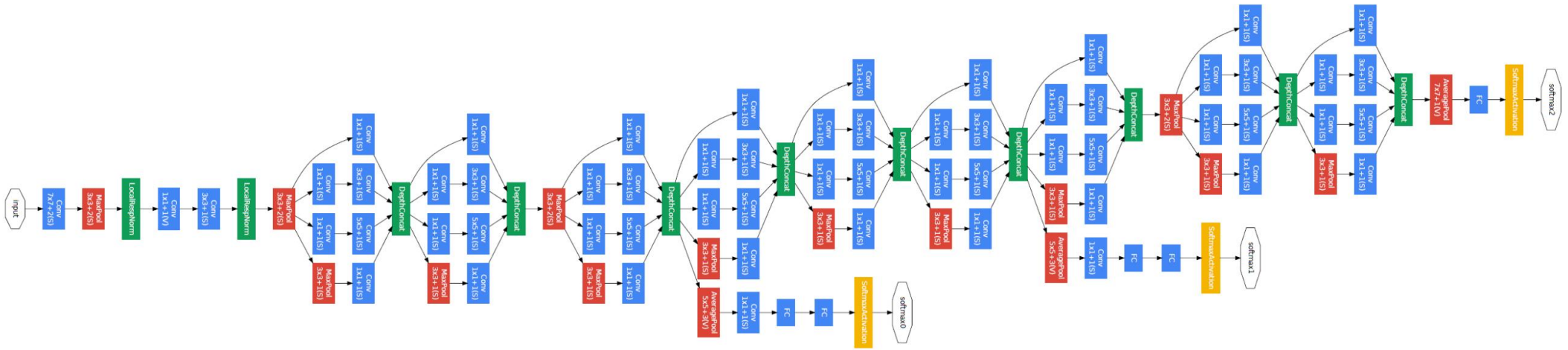
- M.Tech – Artificial Intelligence
  - NMIMS (2021 - 2023, currently pursuing)
- B.E. – Computer Engineering
  - Mumbai University (2015 - 2019)

Advanced CNN Architectures

AN INTUITION TO BUILD YOUR OWN MODELS AND LEVERAGE THE PRETRAINED MODEL

Google's thoughts on building CNNs

GoogleNet/Inception V1



Google's Approach

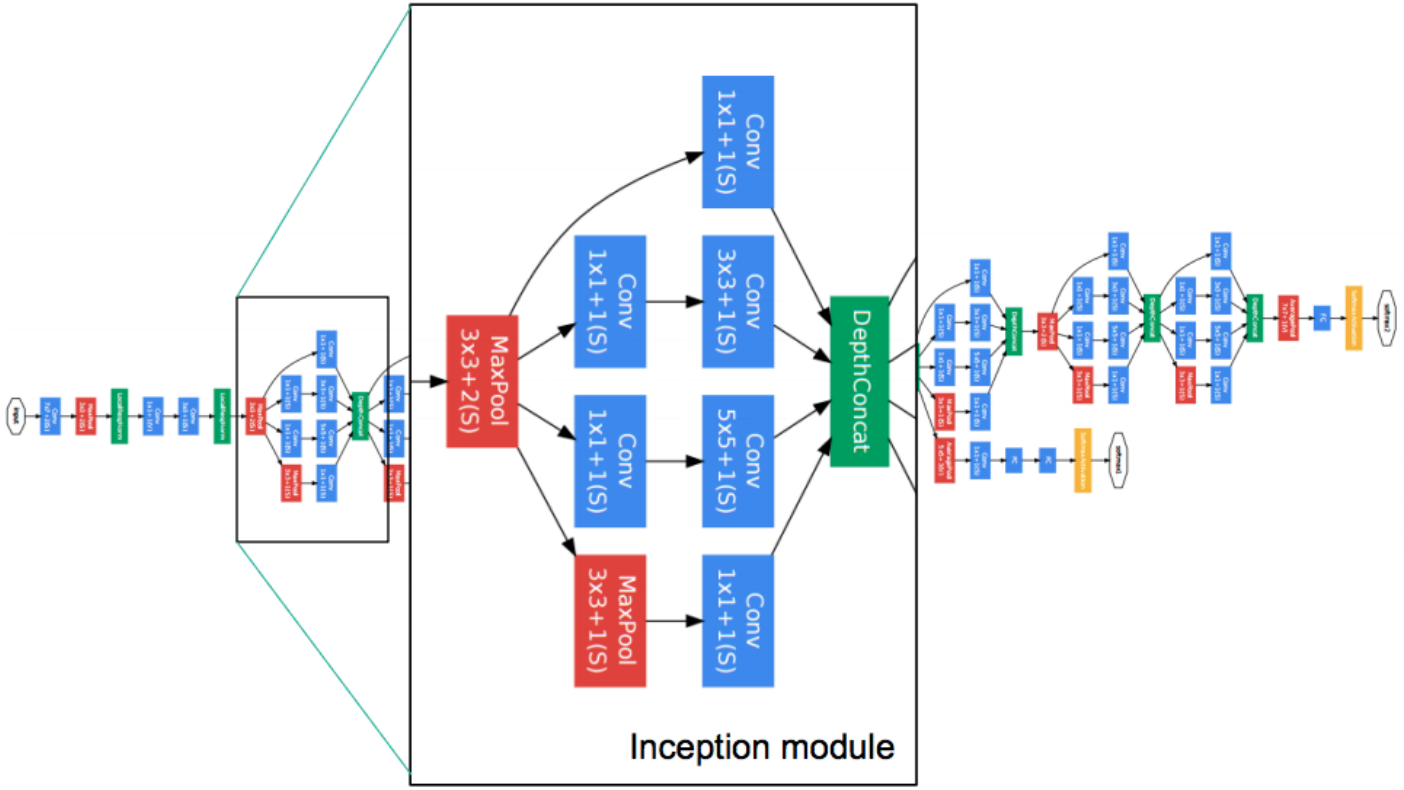
- GoogleNet/Inception V1 uses a network 22 layers deep (deeper than VGGNet) while reducing the number of parameters by 12 times (from ~138 million to ~13 million) and achieving significantly more accurate results.
- **The kernel size of the convolutional layer**—We've seen in previous architectures that the kernel size varies: 1 × 1, 3 × 3, 5 × 5, and, in some cases, 11 × 11 (as in AlexNet). When designing the convolutional layer, we find ourselves trying to pick and tune the kernel size of each layer that fits our dataset. Recall from chapter 3 that smaller kernels capture finer details of the image, whereas bigger filters will leave out minute details.
- **When to use the pooling layer**—AlexNet uses pooling layers every one or two convolutional layers to downsize spatial features. VGGNet applies pooling after every two, three, or four convolutional layers as the network gets deeper

Google's Approach

Configuring the kernel size and positioning the pool layers are decisions that are made by trial and error and experiment

Google RnD Team's idea - ***"Instead of choosing a desired filter size in a convolutional layer and deciding where to place the pooling layers, let's apply all of them all together in one block and call it the inception module."***

GoogleNet/Inception V1



Brainstorming of Inception Module

The inception module is a combination of four layers:

1 × 1 convolutional layer, 3 × 3 convolutional layer, 5 × 5 convolutional layer, 3 × 3 max-pooling layer

Suppose we have an feature map input of size 32x32x200

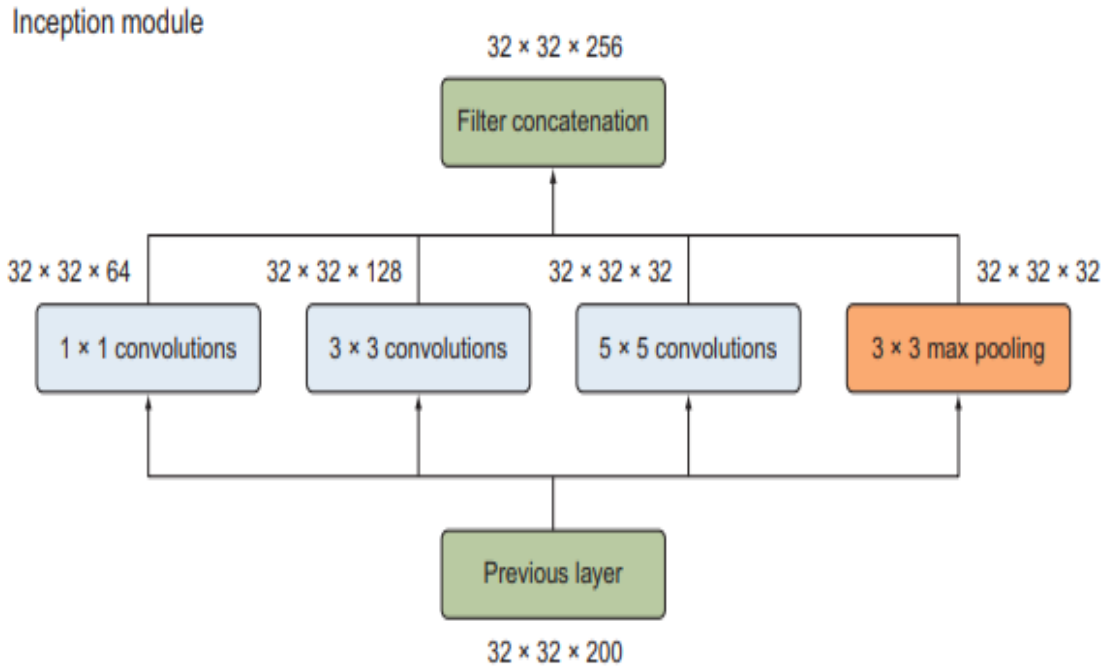
We feed this input to four convolutions simultaneously:

– 1 × 1 convolutional layer with depth = 64 and padding = same.
The output of this kernel = 32 × 32 × 64.

– 3 × 3 convolutional layer with depth = 128 and padding = same.
Output = 32 × 32 × 128.

– 5 × 5 convolutional layer with depth = 32 and padding = same.
Output = 32 × 32 × 32.

– 3 × 3 max-pooling layer with padding = same and strides = 1.
Output = 32 × 32 × 32.



Naïve Version

Brainstorming of Inception Module

The naïve version of Inception module has a big computational cost problem due to the usage and processing of larger filters like 5x5

The input volume with dimensions of 32 × 32 × 200 will be fed to the 5 × 5 convolutional layer of 32 filters with dimensions = 5 × 5 × 32. (5 × 5: filter size; 32: no. of filters)

This means the total number of multiplications that the computer needs to compute is 32 × 32 × 200 multiplied by 5 × 5 × 32, which is more than **163 million** operations. (32x32x200=204800, 5x5x32=800 -> 204800x800 = **163,840,000**

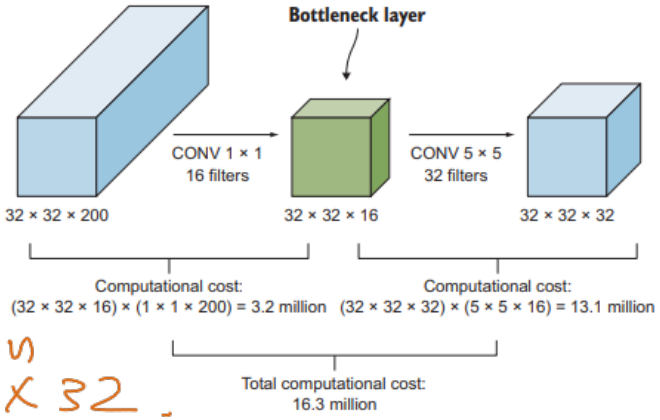
While Google thought that this computation can be performed with modern computers this computation is still expensive.

Dimensionality Reduction Layer (1x1 Conv Layer)

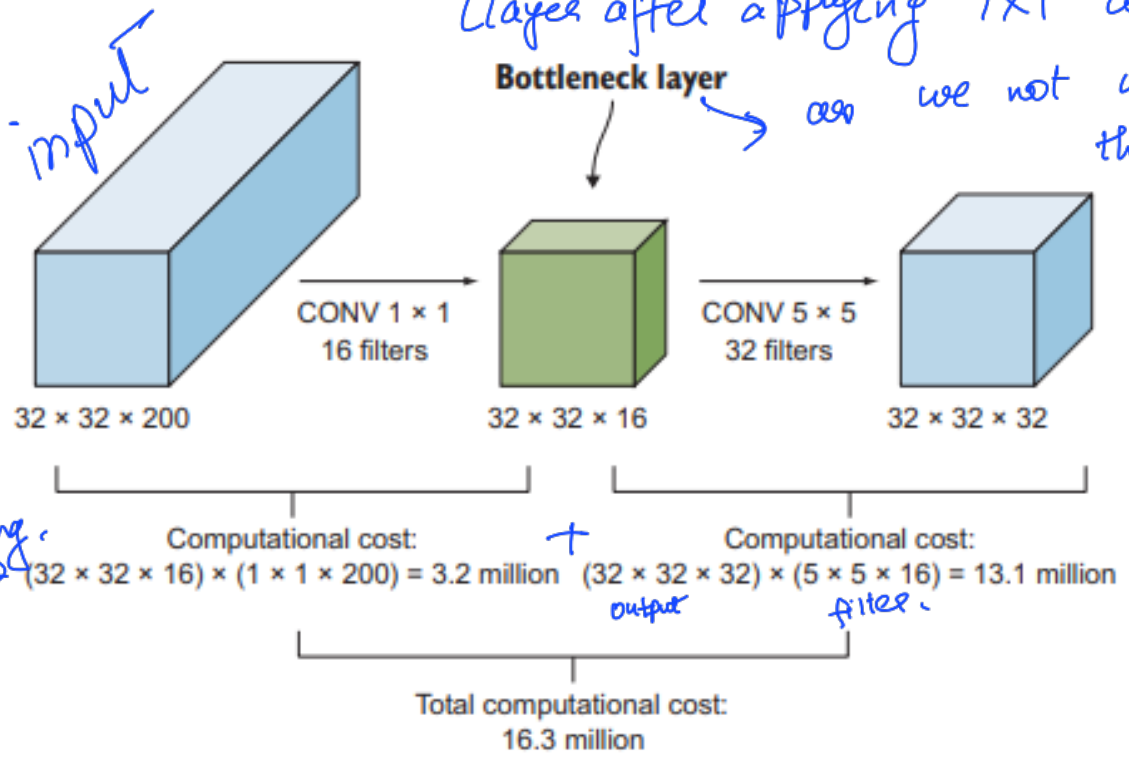
The 1 × 1 convolutional layer can reduce the operational cost of 163 million operations to about a tenth of that. That is why it is called a reduce layer/bottleneck layer.

The idea here is to add a **1 × 1 convolutional layer** before the bigger kernels like the 3 × 3 and 5 × 5 convolutional layers, to reduce their depth, which in turn will reduce the number of operations.

Suppose we have a input feature map of 32x32x200 and we apply a 1x1x16 convolution. This reduces the dimension volume from 200 to 16 channels. We can now apply 5x5 convolution to this output.



Dimensionality Reduction Layer (1x1 Conv Layer)



input

Layer after applying 1X1 conv that reduces the no. of params).

Bottleneck layer → are we not using this in the calculation?

input

its (32 X 32 X 200)

X (1X1 X16) filter.

wrong.

output

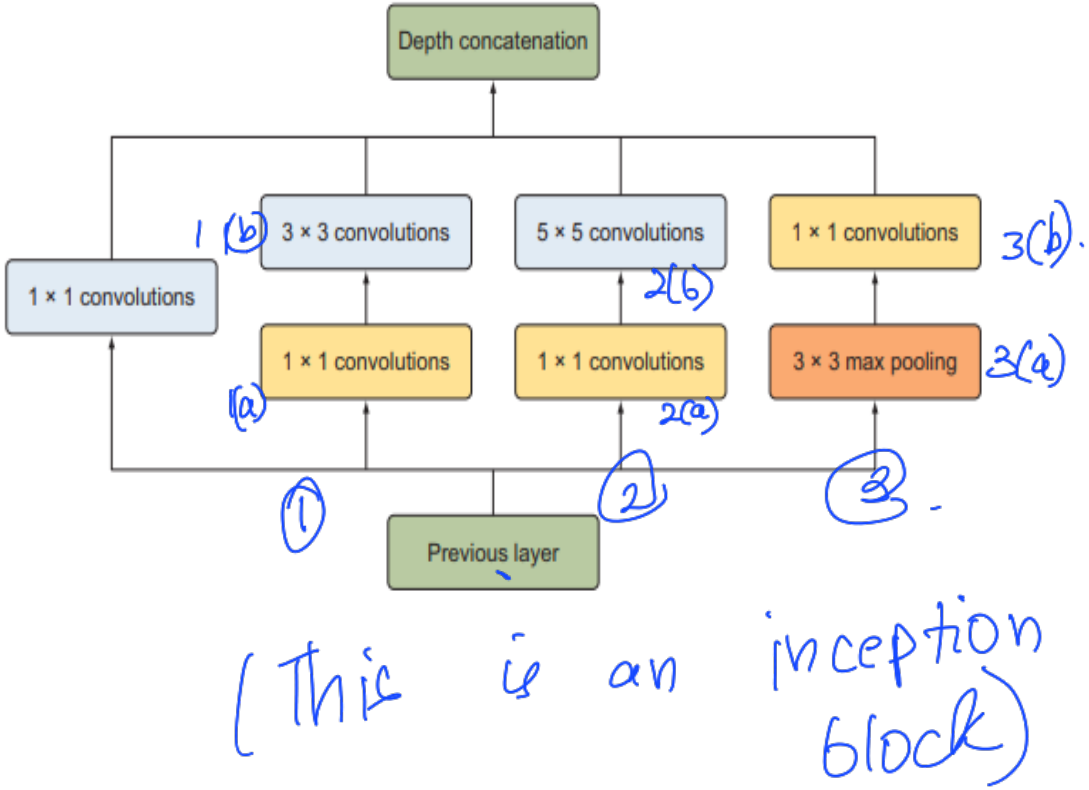
filter.

Inception Module with Dimensionality Reduction

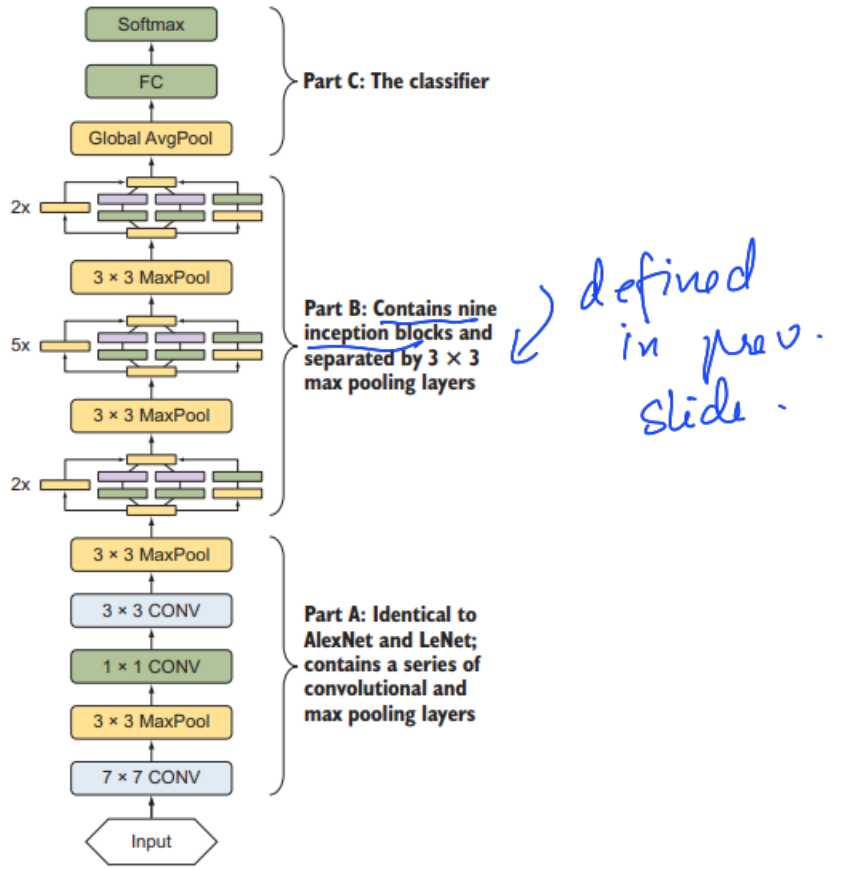
Does shrinking the depth representation size so dramatically hurt the performance of neural network ?

Szegedy et al. ran experiments and found that as long as you implement the reduce layer in moderation, you can shrink the representation size significantly without hurting performance and save a lot of computations

Note: We will add a 1 × 1 convolutional reduce layer before the 3 × 3 and 5 × 5 convolutional layers to reduce their computational cost. We will also add a 1 × 1 convolutional layer after the 3 × 3 max-pooling layer because pooling layers don't reduce the depth for their inputs



GoogleNet/Inception V1



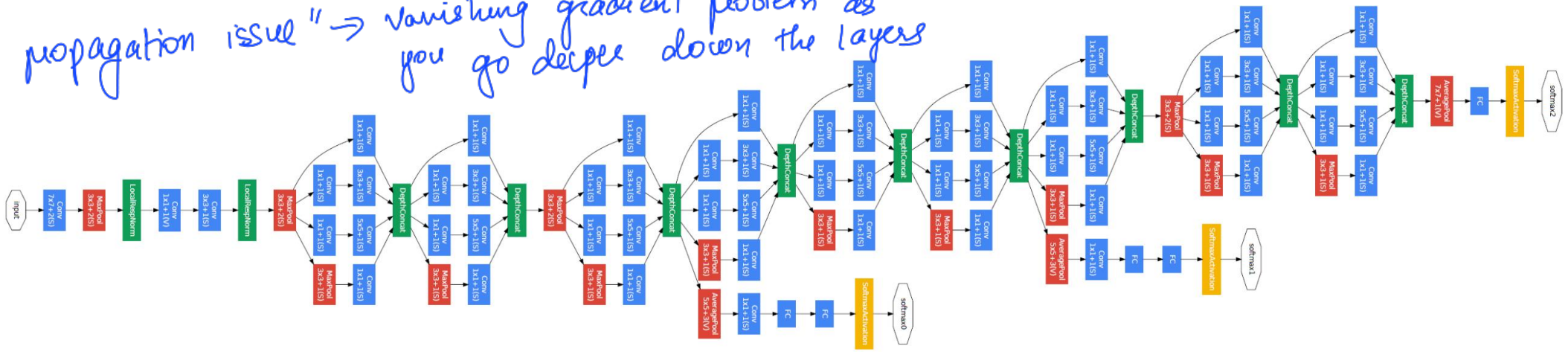
highly discriminating ⇒ very distinct features which make it easy to differentiate one image from another

GoogleNet/Inception V1

In the paper, the authors were concerned about the difficulty of effectively propagating gradients through deep neural networks. They observed that shallower networks were performing well on image classification tasks, indicating that the middle layers in deeper networks were producing highly discriminative features. To help address the gradient propagation issue, they proposed adding auxiliary classifiers to the network that are connected to the intermediate layers. These classifiers are trained to predict the correct class labels, and their loss is added to the overall loss of the network. This encourages the lower stages of the classifier to better discriminate between classes and increases the gradient signal that is propagated back through the network during training. Additionally, it provides extra regularization to help prevent overfitting.

"Given the relatively large depth of the network, the ability to propagate gradients back through all the layers in an effective manner was a concern. One interesting insight is that the strong performance of relatively shallower networks on this task suggests that the features produced by the layers in the middle of the network should be very discriminative. By adding auxiliary classifiers connected to these intermediate layers, we would expect to encourage discrimination in the lower stages in the classifier, increase the gradient signal that gets propagated back, and provide additional regularization."

"gradient propagation issue" ⇒ vanishing gradient problem as you go deeper down the layers



GoogleNet/Inception V1 Summary

Following were the developments

1. Inception Block contains
    1. 1 × 1 convolutional layer
    2. 1 × 1 convolutional layer + 3 × 3 convolutional layer
    3. 1 × 1 convolutional layer + 5 × 5 convolutional layer
    4. 3 × 3 pooling layer + 1 × 1 convolutional layer
2. 1x1 conv layers are used for depth dimensionality reduction thus reducing the floating point operations
3. Uses Global Average Pooling instead of Flatten
4. Uses Auxillary classifiers for prediction and gradient propagation at intermediate parts of the network

Auxiliary classifiers, also known as auxiliary heads, are additional classifiers that are added to a deep neural network at intermediate layers. These classifiers are trained to predict the correct class labels and their loss is added to the overall loss of the network.

The idea behind auxiliary classifiers is to provide additional supervision to the lower stages of the classifier, which can help improve the gradient signal that is propagated back through the network during training. By adding these auxiliary classifiers, the network is encouraged to learn more discriminative features in the lower stages of the network, which can help improve the overall performance of the network.

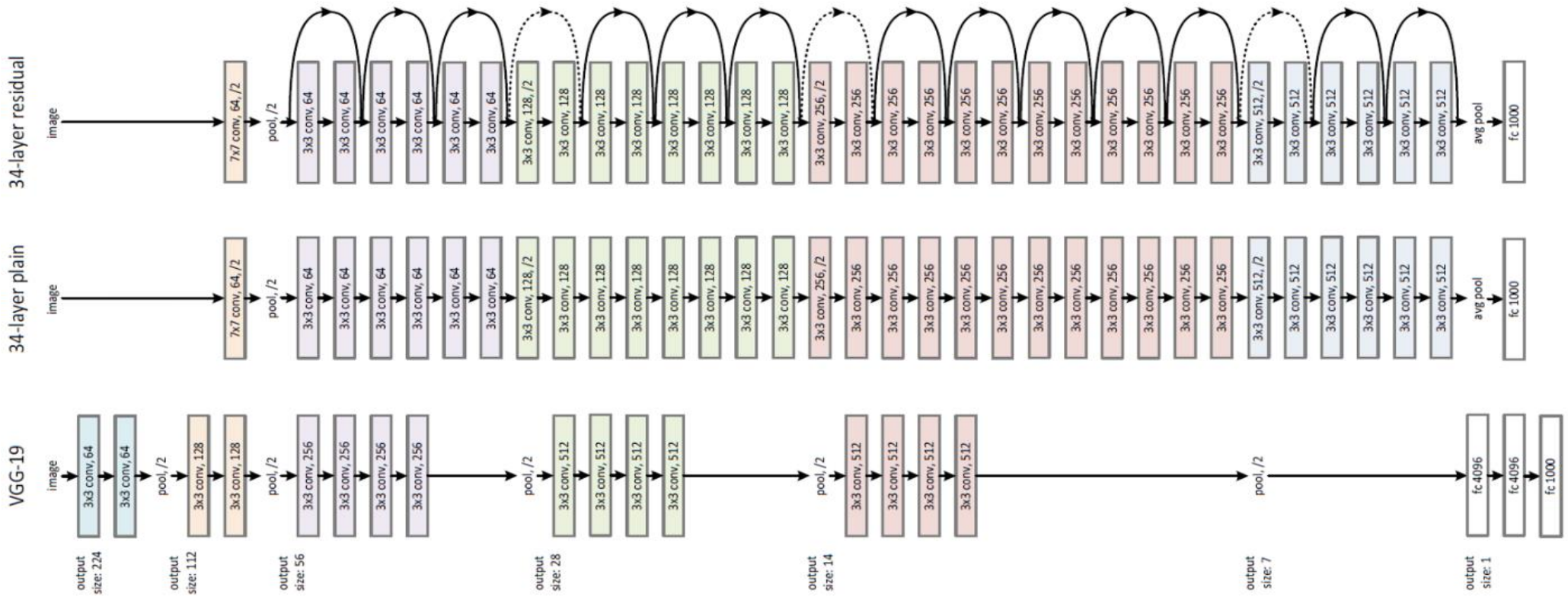
Auxiliary classifiers can be particularly useful in very deep neural networks, where the gradients can become very small during backpropagation, making it difficult to effectively update the weights in the lower stages of the network. By adding auxiliary classifiers, the gradient signal is increased, making it easier to update the weights in the lower stages of the network. Additionally, auxiliary classifiers can help prevent overfitting by introducing additional regularization to the network.

Microsoft's thoughts on building CNNs

Google

Microsoft

ResNet


VGG-19
image
3x3 conv, 64
3x3 conv, 64
pool, /2
3x3 conv, 128
3x3 conv, 128
pool, /2
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
pool, /2
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
pool, /2
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
pool, /2
fc 4096
fc 4096
fc 1000
output size: 224
output size: 112
output size: 56
output size: 28
output size: 14
output size: 7
output size: 1
34-layer plain
image
7x7 conv, 64, /2
pool, /2
3x3 conv, 64
3x3 conv, 128, /2
3x3 conv, 128
3x3 conv, 256, /2
3x3 conv, 256
3x3 conv, 512, /2
3x3 conv, 512
avg pool
fc 1000
34-layer residual
image
7x7 conv, 64, /2
pool, /2
3x3 conv, 64
3x3 conv, 128, /2
3x3 conv, 128
3x3 conv, 256, /2
3x3 conv, 256
3x3 conv, 512, /2
3x3 conv, 512
avg pool
fc 1000

Microsoft's Approach

MS research team introduced a novel **residual module** architecture with skip connections

The network also features heavy batch normalization for the hidden layers

Upon observing AlexNet, VGGNet, Inception you must have noticed the deeper the network, the larger its learning capacity and the better it extracts features from the images.

This happens because **deeper networks** are able to represent **very complex functions,** which allows the network to learn features at many different levels of abstraction, from edges (at the lower layers) to very complex features (at the deeper layers).

VGG-19 was 19 layers, GoogleNet was 22 layers can we go deeper ?

Microsoft's Approach

normalizing the inputs to each layer in a network

normalizing the data in batches so they lie in the same range, duh.

Downsides of Deeper Networks

1. Adding too many layers makes the network prone to overfit on the training data
   - However overfitting can be addressed using regularization, dropout and batch normalization
   - Now can we build networks which are 50/100/150 layers deep and good at learning features ?
2. Vanishing and Exploding Gradients
   1. During backpropagation, in the chained multiplication the gradient from the later layers will become very small by the time it reaches the initial layers of the network – applicable to activation functions which are diminishing in nature like tanh and sigmoid
   2. During backprop, it also might be the case that the gradient grows exponentially quickly during the chained multiplication and takes very large values thus exploding – usually the case with activation functions like ReLU or similar to ReLU (can be solved using Gradient Clipping)

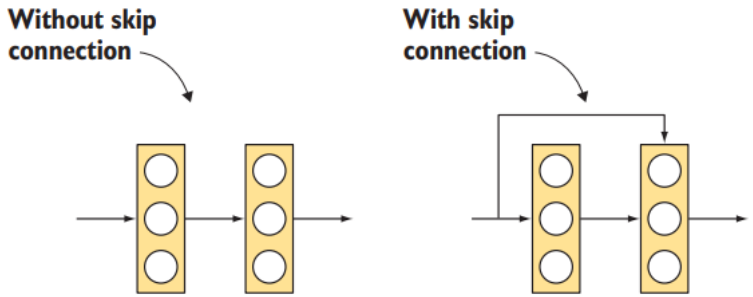
Microsoft's approach to Vanishing Gradient

To solve the vanishing gradient problem, He et al. created a shortcut that allows the gradient to be directly backpropagated to earlier layers. These shortcuts are called as skip connections.

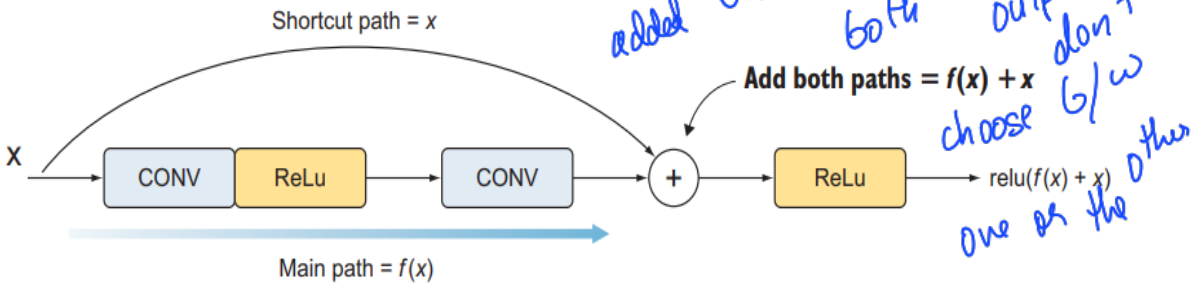
They are used to flow information from earlier layers in the network to later layers, creating an alternate shortcut path for the gradient to flow through.

Skip connections also allow the model to learn an identity function which ensures that the layer will perform at least as well as the previous layer

ANN's Case



CNN's Case



Microsoft's approach to Vanishing Gradient

Unlike traditionally just stacking the convolutions just one after the other they also add the original input to the output of the stacked convolutional blocks

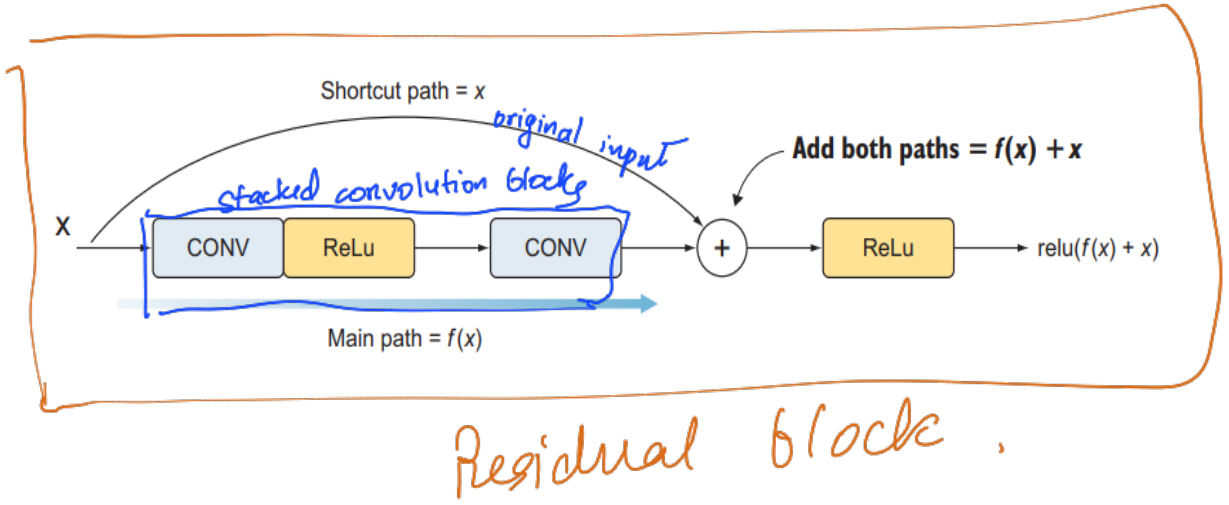
The shortcut arrow/skip connection is added at the end of the convolution layer

The reason is that we add both paths before we apply the ReLU activation function of this layer

We apply the ReLU activation to f(x) + x to produce the output signal: ReLU(f(x) + x).

Note: This combination of the skip connection and convolutional layers is called a residual block. Similar to the Inception network, ResNet is composed of a series of these residual block building blocks that are stacked on top of each other

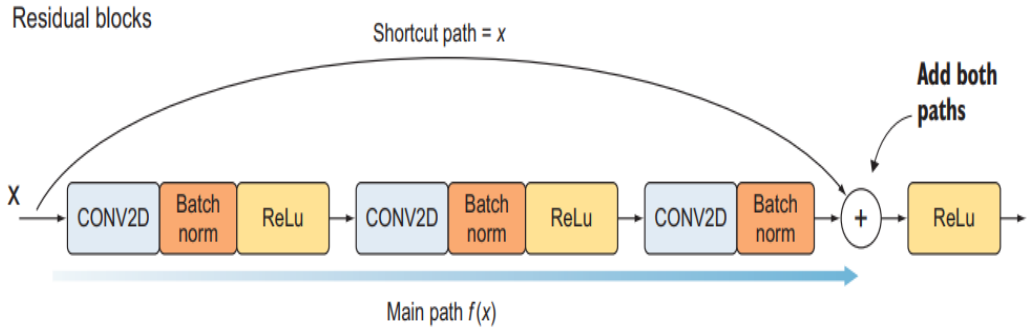
CNN's Case



Residual Blocks

Residual blocks consist of two branches

1. **Shortcut path:** Connects the input to an addition of the second branch
2. **Main path:** A series of convolutions and activations. The main path consists of three convolutional layers with ReLU activations. We also add batch normalization to each convolutional layer to reduce overfitting and speed up training. The main path architecture looks like this: [CONV -> BN -> ReLU] × 3.



As mentioned earlier the shortcut path is added to the main path right before the activation function of the last convolution layer. Then we add ReLU function after adding the two paths.

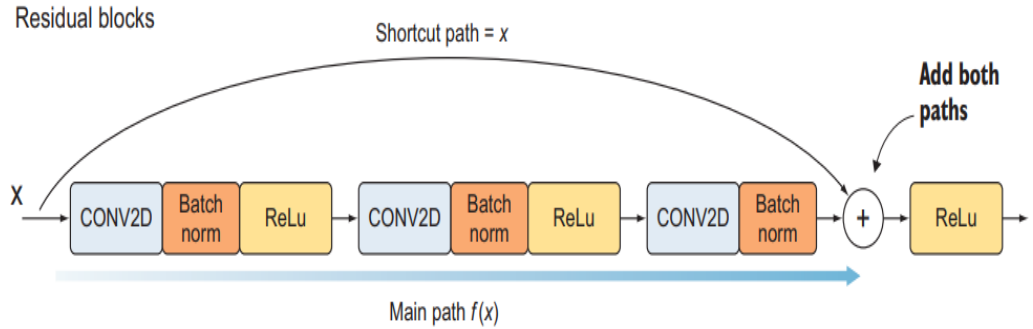
Residual Blocks

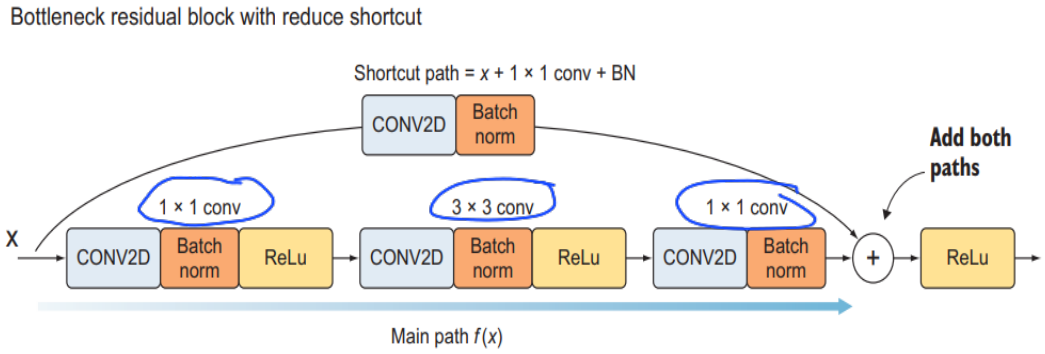
1. He et al. decided to do dimension downsampling using bottleneck 1 × 1 convolutional layers, similar to the Inception network
2. Residual blocks starts with a 1 × 1 convolutional layer to downsample the input dimension volume, and a 3 × 3 convolutional layer and another 1 × 1 convolutional layer to downsample the output.
3. This is a good technique to keep control of the volume dimensions across many layers. This configuration is called a bottleneck residual block

Keeps the volume in check.

4. Since the volume dimensions are changing due to the bottleneck layers, we also sometimes downsample the shortcut path as well using the bottleneck layer.

Note: ResNets do not have a pooling layer in the network, they instead use strides=2 to downsample the width and height of the image in the reduce layer itself.





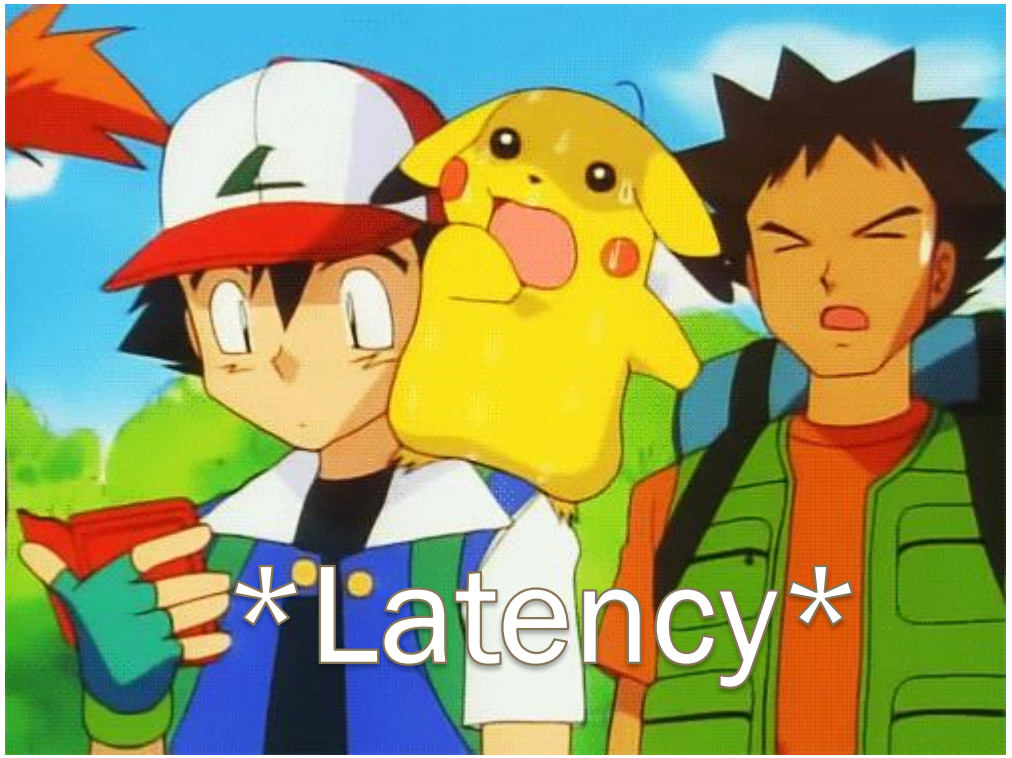
ResNet Summary

Residual blocks contain two paths

1. The shortcut path and the main path.
2. The main path consists of three convolutional layers, and we add a batch normalization layer to them:
    1. 1 × 1 convolutional layer
    2. 3 × 3 convolutional layer
    3. 1 × 1 convolutional layer
3. There are two ways to implement the shortcut path:
    1. **Regular shortcut:** Add the input dimensions to the main path.
    2. **Reduce shortcut:** Add a convolutional layer in the shortcut path before merging with the main path.

(downsampling the shortcut by adding a 1x1 conv)

Google’s thoughts on Models running on mobile devices



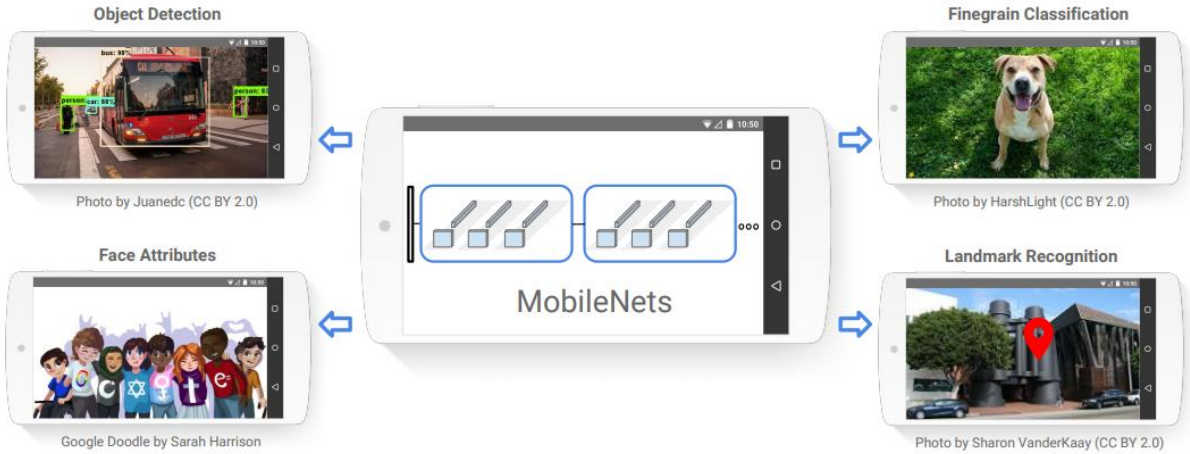
idgi LoL

MobileNet: Google's approach at light weight models

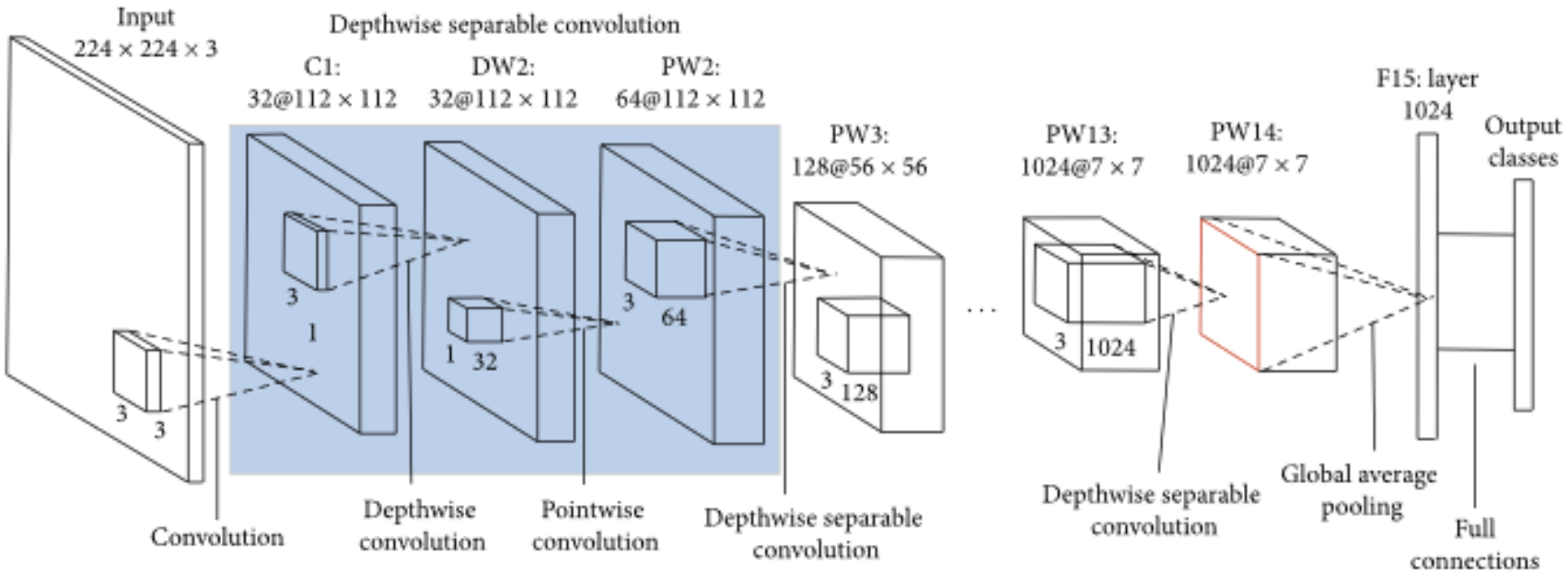
A class of efficient models for mobile and embedded vision applications

Mobilenets make use of Depth Wise Separable Convolutions along with reduce layer (1x1 convolution) to build light weight deep neural networks

Two simple hyperparameters introduced that efficiently trade off between latency and accuracy



MobileNet


Input
224 × 224 × 3
Depthwise separable convolution
C1:
32@112 × 112
DW2:
32@112 × 112
PW2:
64@112 × 112
PW3:
128@56 × 56
PW13:
1024@7 × 7
PW14:
1024@7 × 7
F15: layer
1024
Output
classes
3
3
3
1
1
32
3
64
3
128
3
1024
Convolution
Depthwise
convolution
Pointwise
convolution
Depthwise separable
convolution
Depthwise separable
convolution
Global average
pooling
Full
connections

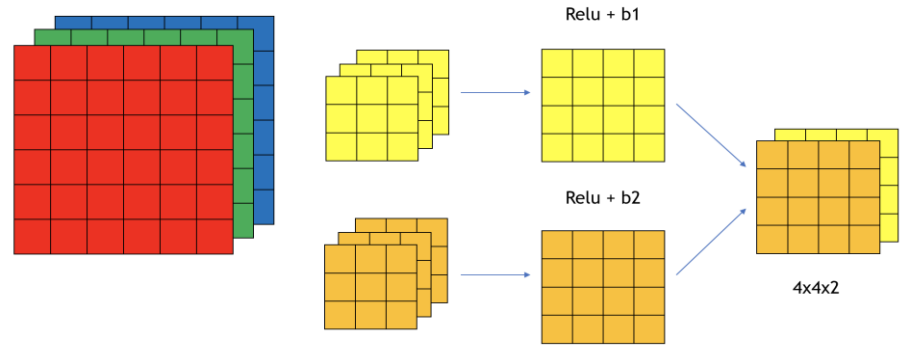
MobileNet Approach: Depthwise Separable Convolutions

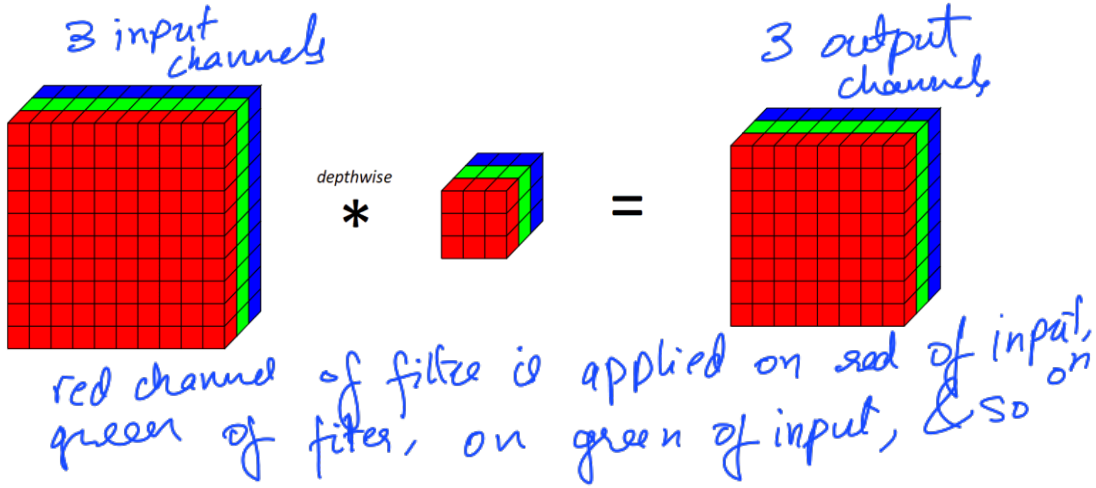
In regular convolution a kernel is applied across all channels of the image

A depthwise convolution is basically a convolution along only one spatial channel of the image.

The key difference between a normal convolutional layer and a depthwise convolution is that the depthwise convolution applies the convolution along only one spatial dimension (i.e. channel) while a normal convolution is applied across all spatial dimensions/channels at each step.

Note: Since we are applying one convolutional filter for each output channel, the number of output channels is equal to the number of input channels. After applying this depthwise convolutional layer, we then apply a pointwise convolutional layer. Here the pointwise convolution will not be known as reduce but as expand layer.



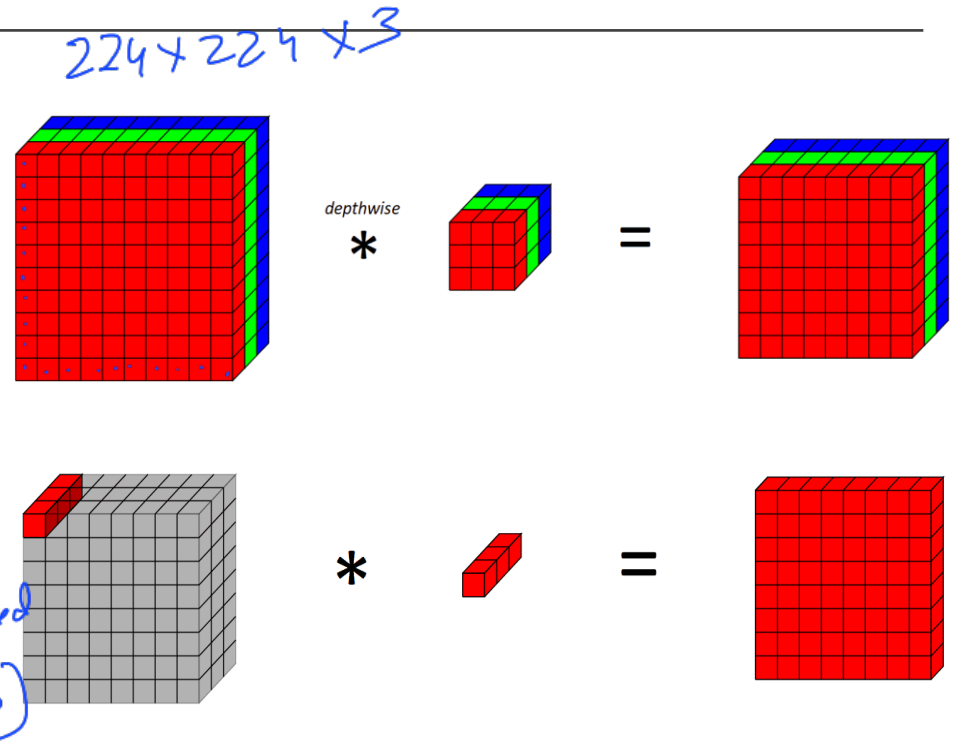


MobileNet Approach: Depthwise Separable Convolution + Pointwise Convolution

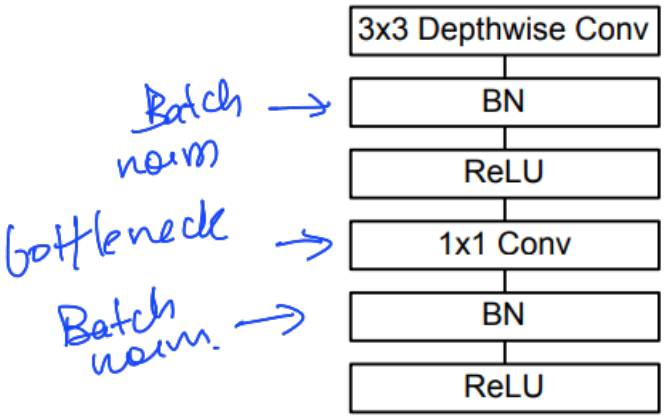
Suppose we have a RGB image of size 224x224x3. You want to apply the convolutional layer with 3x3 size kernel with 64 total kernels

1. 3x3x3x64+64=1792 parameters (if directly 64 3x3 kernels are applied)
2. DSC+ PWC
   1. 3x3x1x3+3 = 30 params (DSC)
   2. 1x1x3x64+64 = 256 params (PWC)
3. 30 + 256 = 286 parameters

Same operation performed with significantly lesser number of parameters !



MobileNet Architecture



Note: MobileNet does not use pooling either. It uses strides of 2 for downsampling width and height. This has been the case for most of the modern architectures

Table 1. MobileNet Body Architecture

Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
$5\times$ Conv dw / s1	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
$5\times$ Conv / s1	$1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool 7×7	$7 \times 7 \times 1024$
FC / s1	1024×1000	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

MobileNet's Two Model Lightening Hyperparameters

1. **Width Multiplier:** Although the base MobileNet architecture is already small and low latency, many times a specific use case or application may require the model to be smaller and faster.
   - The role of the width multiplier α is to thin a network uniformly at each layer.
   - For a given layer and width multiplier α, the number of input channels M becomes αM and the number of output channels N becomes αN. The value of α is typically between [0,1] If the width isn't getting increased in size, it decreases.
2. **Resolution Multiplier:** A resolution multiplier ρ is used on the input image and the internal representation of every layer is subsequently reduced by the same multiplier. In practice we implicitly set ρ by setting the input resolution. Value of ρ is typically between [0,1] but as mentioned, in practice the input resolution itself is set typically - 224, 192, 160 or 128

MobileNet Summary

Mobilenet makes use of special type of convolution combined with pointwise convolution

1. Depthwise Separable Convolutions used along with Pointwise Convolutions to reduce the number of parameters as well as computations
2. Width Multiplier and Resolution Multiplier hyperparameters used for further making the model more light weight

EfficientNet

To be covered in the next class with Visual Embeddings and Face Recognition

What do CNNs see ?

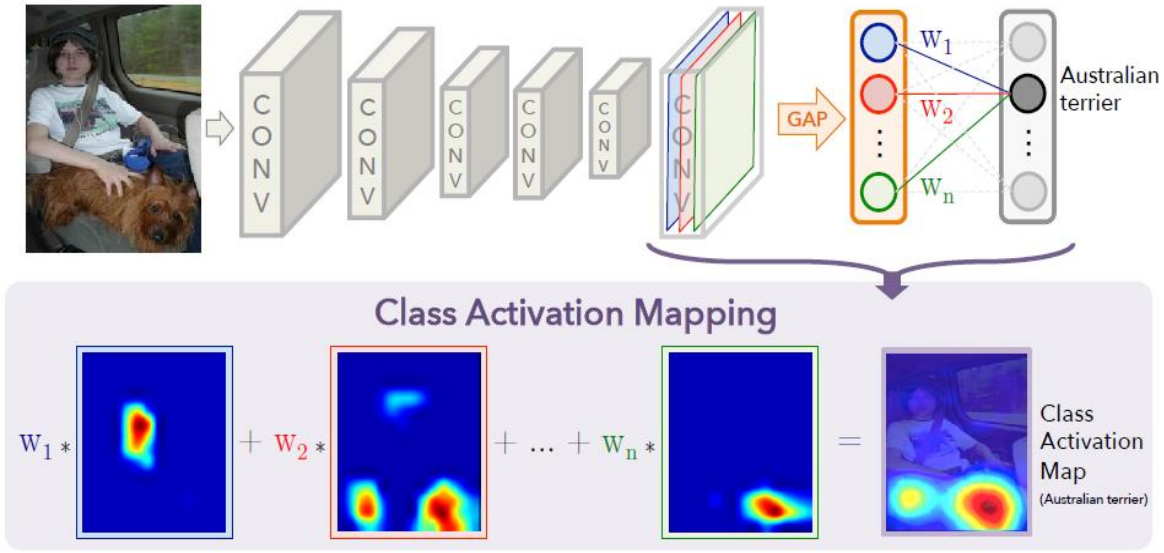
1. Class Activation Maps – Naïve and Restricted Approach
2. Gradient Class Activation Maps (Grad-CAM) – Appropriate for all kinds of tasks
    - Image Classification
    - Image Captioning
    - Visual QnA
    - Image Segmentation

Class Activation Maps

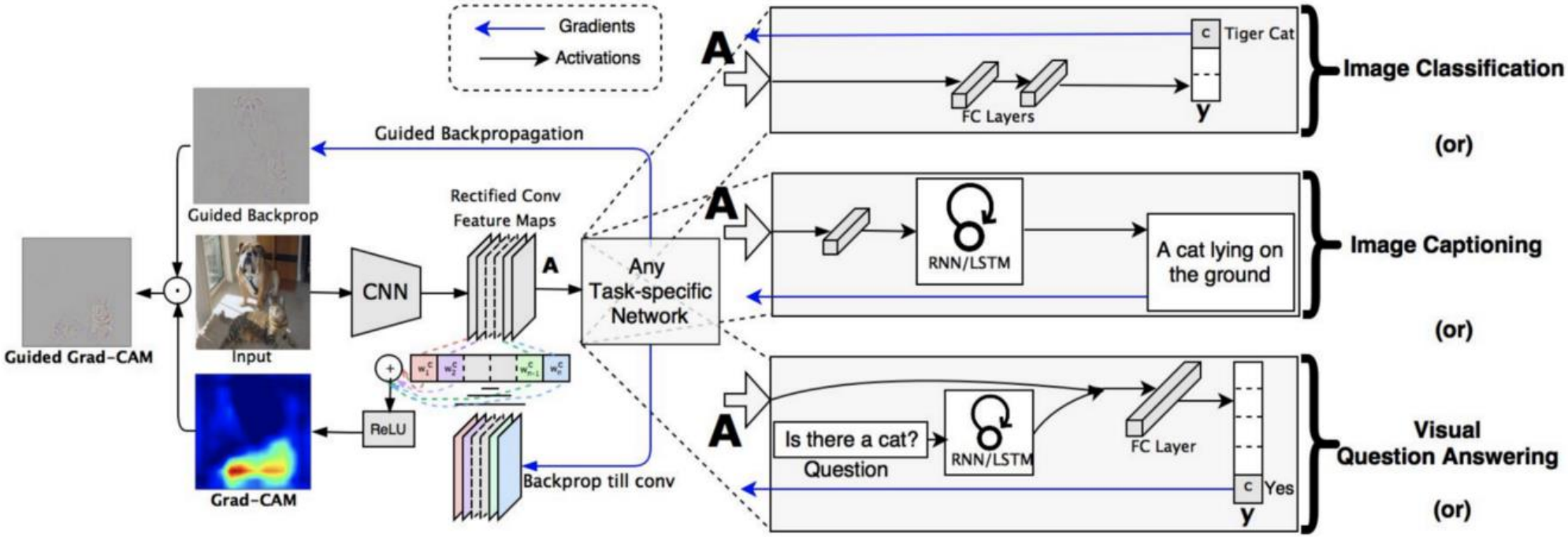
Similar to how you find importance of input features by looking at weights

Weights associated with each Average value of its respective feature map is taken and projected on the image to get its importance

Disadvantage: What if you have fully connected layers instead of directly having a softmax/sigmoid layer after global average pooling

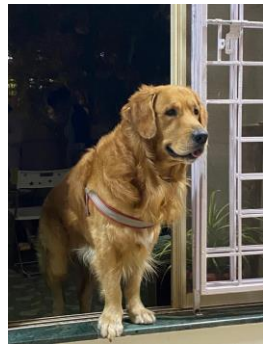


Grad-CAMs


Gradients
Activations
Guided Backpropagation
Guided Backprop
Guided Grad-CAM
Input
Grad-CAM
CNN
Rectified Conv
Feature Maps
A
Any
Task-specific
Network
ReLU
Backprop till conv
C
Tiger Cat
FC Layers
y
Image Classification
(or)
RNN/LSTM
A cat lying on
the ground
Image Captioning
(or)
Is there a cat?
Question
RNN/LSTM
FC Layer
C
Yes
y
Visual
Question Answering
(or)

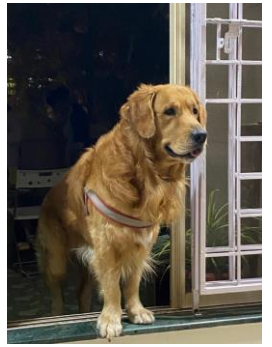
Grad-CAMs: Binary Image Classification

1. Create a custom model in and deep learning framework tf.keras where
    1. Input: image
    2. Output: Gives last Convolution Output and Classification Probability score
2. Get the Classification probability score: Sigmoid activation value
3. Calculate the gradient of this classification activation value wrt to the last Convolution Output
4. If class probability score/sigmoid activation >= 0.5
    1. Take the average of gradient matrix
    2. Else if its < 0.5 put a negative sign on the gradient matrix and then take its average
5. Project this averaged gradient on the last convolutional feature map to form a image importance gradient heatmap
6. We only want the positive impact of this gradient heatmap
    1. Pass the gradient heatmap through a relu function
    2. Normalize the gradient heatmap
7. Resize the gradient heatmap and project it on to the image



Grad-CAMs: Multiclass Image Classification

1. Create a custom model in and deep learning framework tf.keras where
    1. Input: image
    2. Output: Gives last Convolution Output and Classification Probability score
2. Get the Classification probability score: Softmax activation value
3. Calculate the gradient of this classification activation value wrt to the last Convolution Output
4. Get the highest voted class probability/softmax activation using max function
    1. Take the average of gradient matrix
5. Project this averaged gradient on the last convolutional feature map to form a image importance gradient heatmap
6. We only want the positive impact of this gradient heatmap
    1. Pass the gradient heatmap through a relu function
    2. Normalize the gradient heatmap
7. Resize the gradient heatmap and project it on to the image



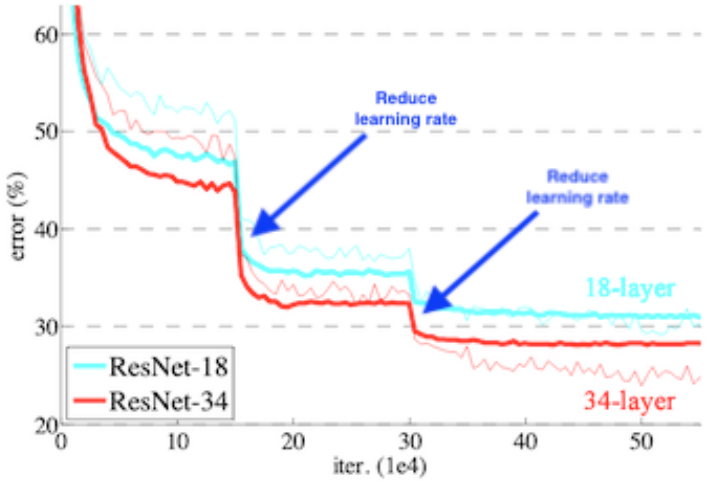
Miscellaneous

Model Callbacks

EarlyStopping

Checkpointing

ReduceLROnPlateau



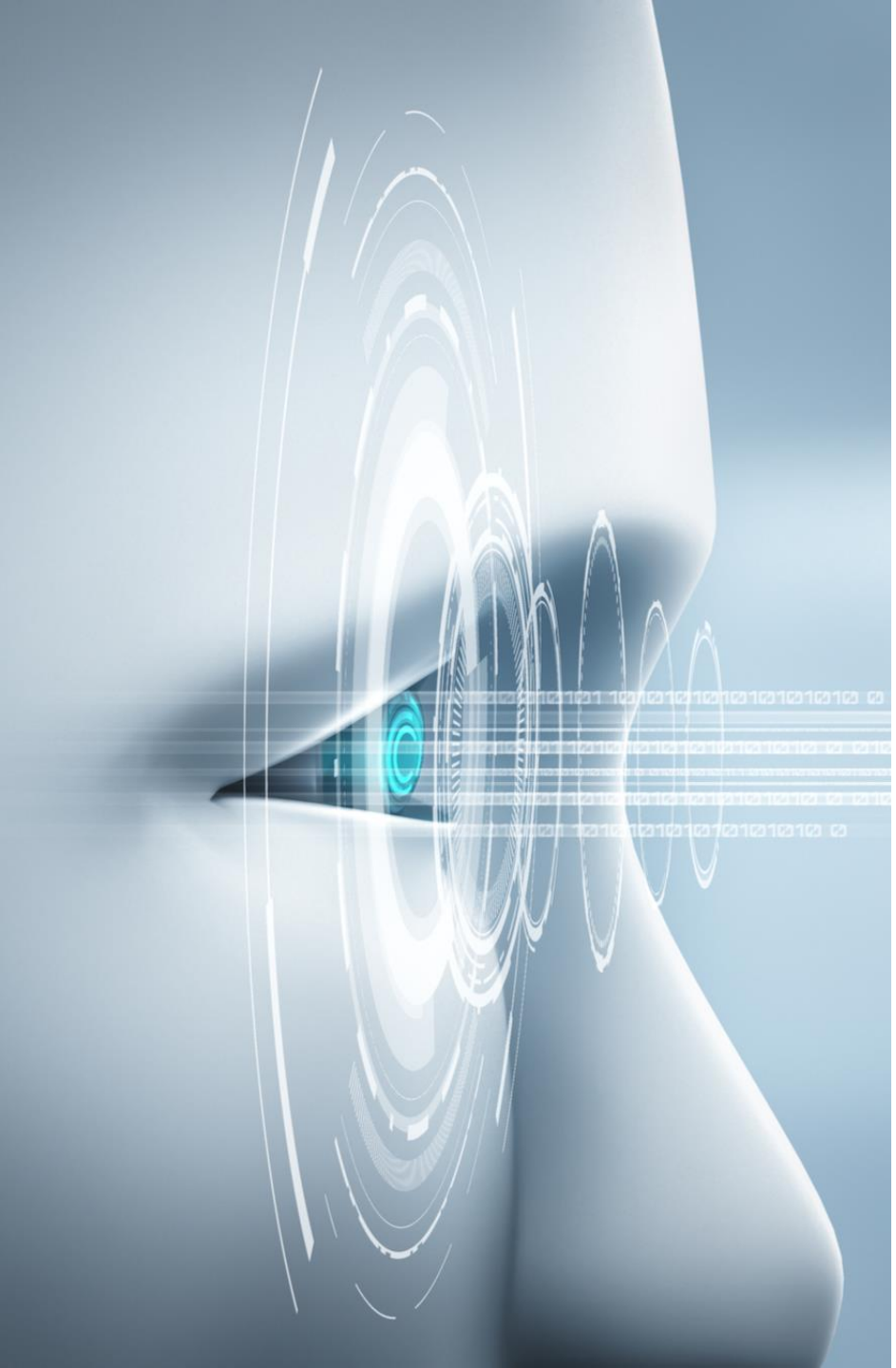
ROC and AUC

https://towardsdatascience.com/demystifying-roc-curves-df809474529a

https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5

Thank you

For any queries drop an email at: quadeershaikh15.8@gmail.com



Computer Vision Applications

BY QUADEER SHAIKH

About me

Work Experience

- Risk Analyst
  - Morgan Stanley (Jan 2023 – Present)
- Data Science Intern
  - AkzoNobel Coatings International B.V. Netherlands (Feb 2022 – Dec 2022)
- Data Science Intern
  - EzeRx Health Tech Pvt. Ltd. (Jan 2022 – July 2022)
- Associate Engineer
  - Tata Communications Ltd. (July 2019 – Aug 2020)
- Network Automation and Analysis Engineer Intern
  - Cisco (June 2018 – July 2018)

Education

- M.Tech – Artificial Intelligence
  - NMIMS (2021 - 2023, currently pursuing)
- B.E. – Computer Engineering
  - Mumbai University (2015 - 2019)

Visual Embeddings and One Shot Learning

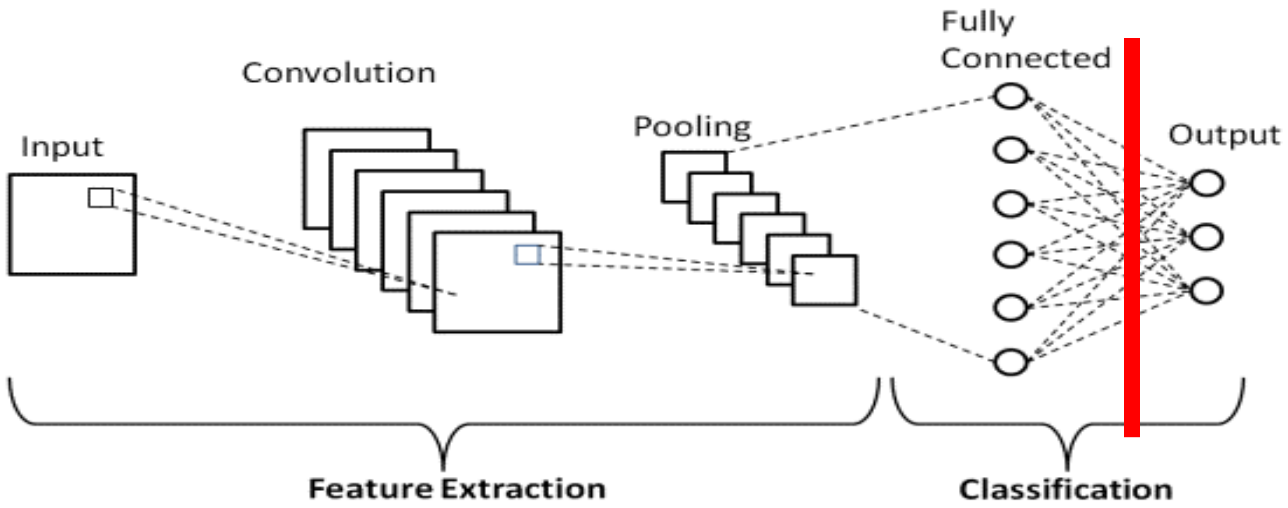
Converting img/vids to numerical vectors. Set of numerical values that represent the visual content of an image. They essentially allow computers to interpret visual data the way humans do.

Visual Embeddings

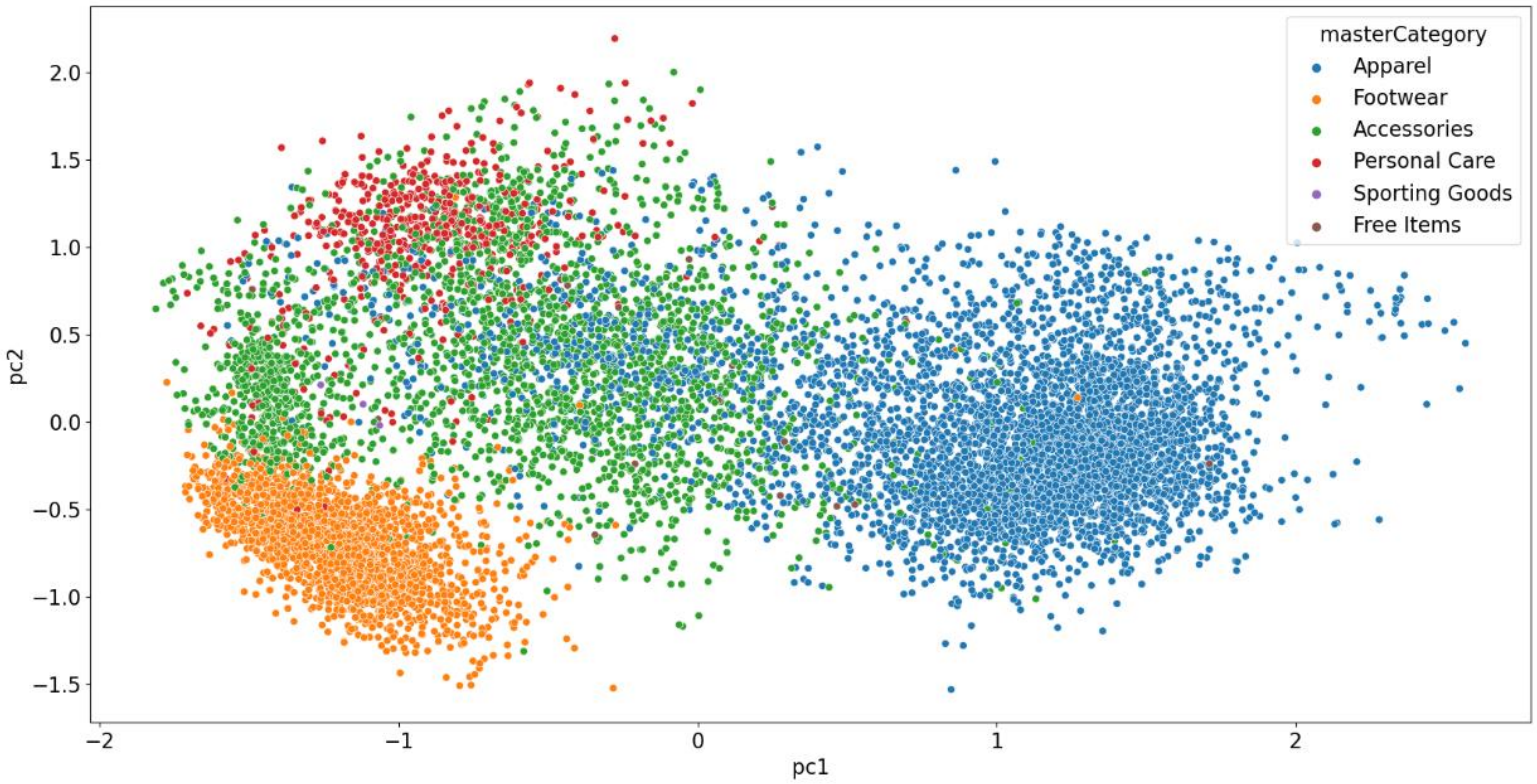
What are image embeddings? An image embedding is a lower-dimensional representation of the image. In other words, it is a dense vector representation of the image which can be used for different computer vision tasks. E.g. Classification

Usually is the output of the layer before the final layer (last fully connected layer)

This way, comparisons between images can be performed by measuring their pairwise distances in this embedding space.

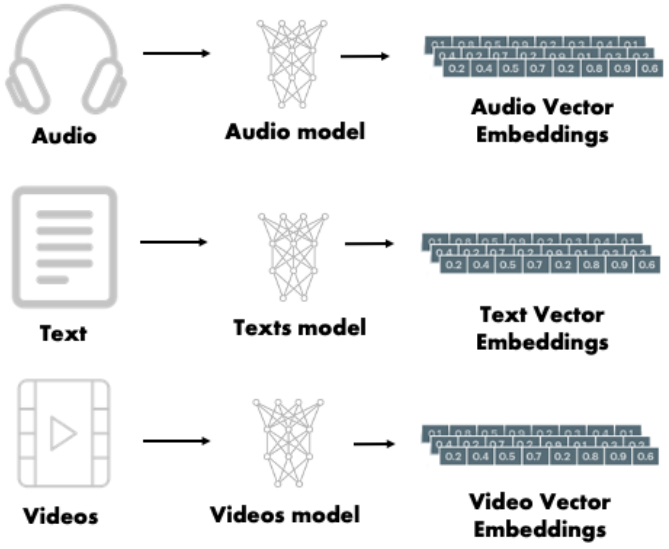


Visual Embeddings: Projected in 2D using Dimensionality Reduction



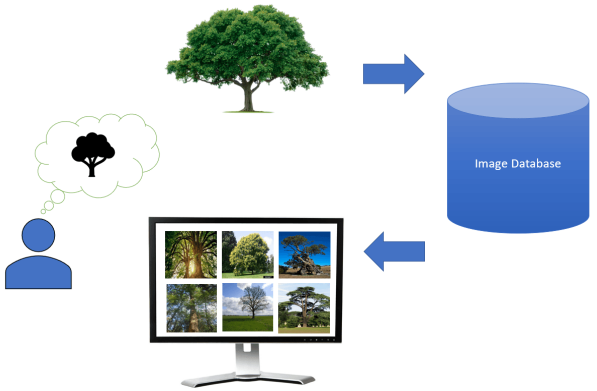
Embeddings: Formal Definition

An embedding is a vector space, typically of lower dimension than the input space, which preserves relative dissimilarity (in the input space). The terms vector space and embedding space are used interchangeably. Embeddings can be extracted for any kind of data form.



Applications of Visual Embeddings

1. Image Recommendation Systems
2. Reverse Image Search Engines
3. Face Recognition
4. Object Re-Identification



Reverse Image Search/Image Recommendation Approach

1. Extract visual embeddings using a pretrained network
2. Store the vector embeddings in a database
3. Provide input image -> calculate visual vector embedding
4. Calculate distance between input image embeddings and all the image embeddings in the database
5. Display the k-Nearest Neighbours based on the distance calculated

Note: In the examples shown, k = 5

Subjective metrics such as hit-rate and A/B tests are used to evaluate the success of these tasks

Face Recognition

One Shot Learning: Face Recognition

One-shot learning is an **object categorization problem**, found mostly in **computer vision**. Whereas most **machine learning**-based object categorization algorithms require training on hundreds or thousands of examples, one-shot learning aims to classify objects from one, or only a few, examples. The term **few-shot learning** is also used for these problems, especially when more than one example is needed.

Tasks like signature or face verification require the recognition of objects or faces. Deep learning algorithms typically require a lot of training data for these recognition based tasks. But, for a setup like face recognition we do not possess a large variety of photos for each person.

Face Recognition: Modes

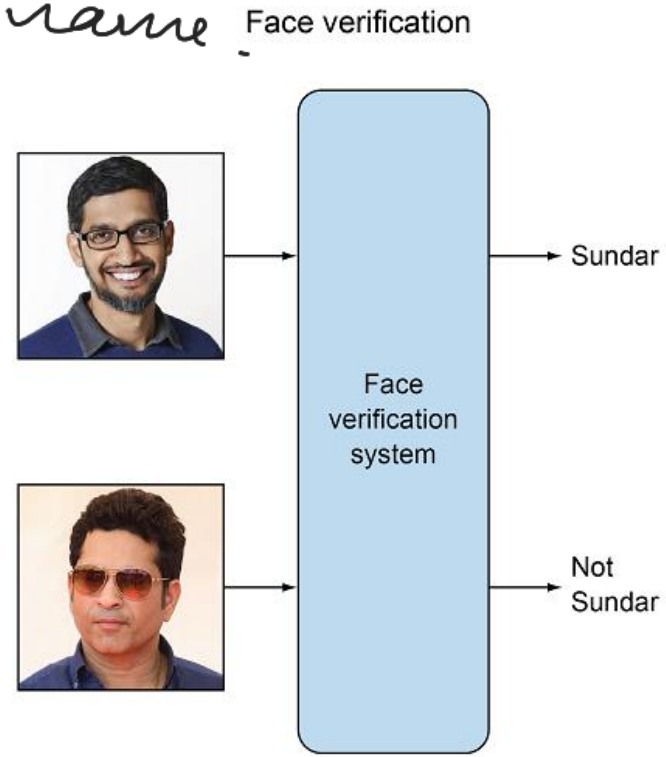
1. **Face Identification:** One-to-many matches that compare a query face image against all the template images in the database to determine the identity of the query face
   - Person A's image is compared with all the other people images present in the database
   - One to Many comparison
2. **Face Verification:** One-to-one match that compares a query face image against a template face image whose identity is being claimed.
   - Person A's image is saved in the database, Person A's new input image is compared with the existing image in the database
   - One to one comparison

confirms whether the input person is who he claims to be by cross-checking with the image in the DB associated to that name.

Face Verification

1. Face image embeddings of users stored in a database
2. User provides and input image, saying he is person A
3. Distance = L2(User's face embeddings, Person A's face embeddings)
4. If Distance < Distance threshold
   - The person is verified as person A
5. Else:
   - It is not person A

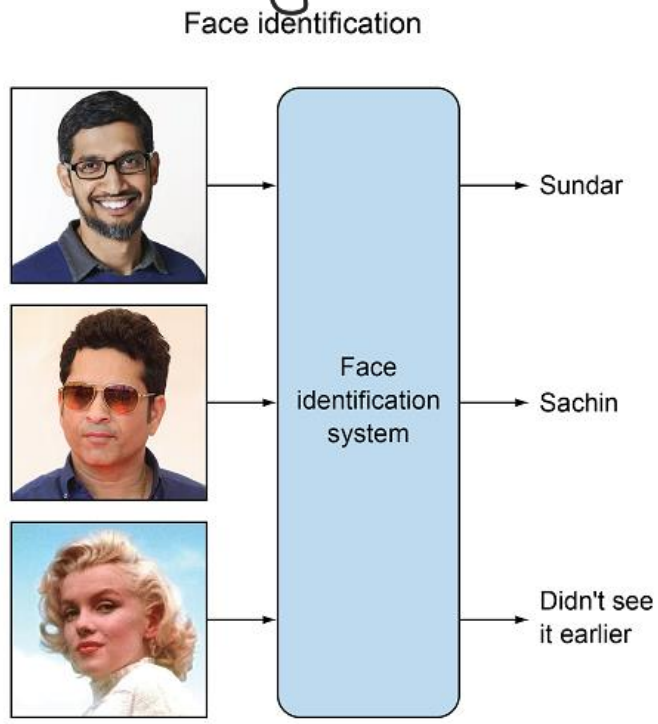
extract visual embeddings of both user, & the already existing image in the database.



Face Identification

tells you who's face is being inputted after matching with every face in the system

1. Face image embeddings of all users stored in a database
2. Person of interest identified
3. Person's face embeddings calculated
4. Distance = L2(person's face embeddings, face embeddings in DB)
5. Find the person id from the database with lowest distance
6. If distance < distance thresh
   - Person is person id with the lowest distance
7. Else
   - Person's image is not present in the database



Face Recognition

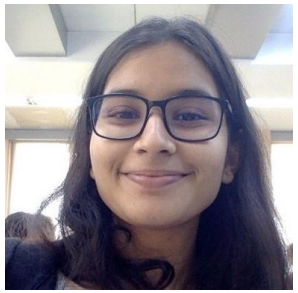
1. Face Detection
   - Haar Cascades
   - Object Detection based models – MTCNN – Covered in Object Detection
   - Mediapipe – To be covered in next lecture
2. Face Alignment
3. Embedding Extraction
4. Face Identification/Verification

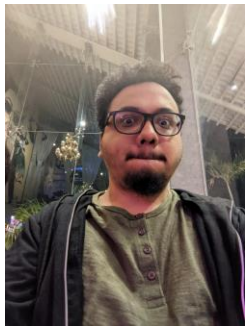
Face Detection: RoI

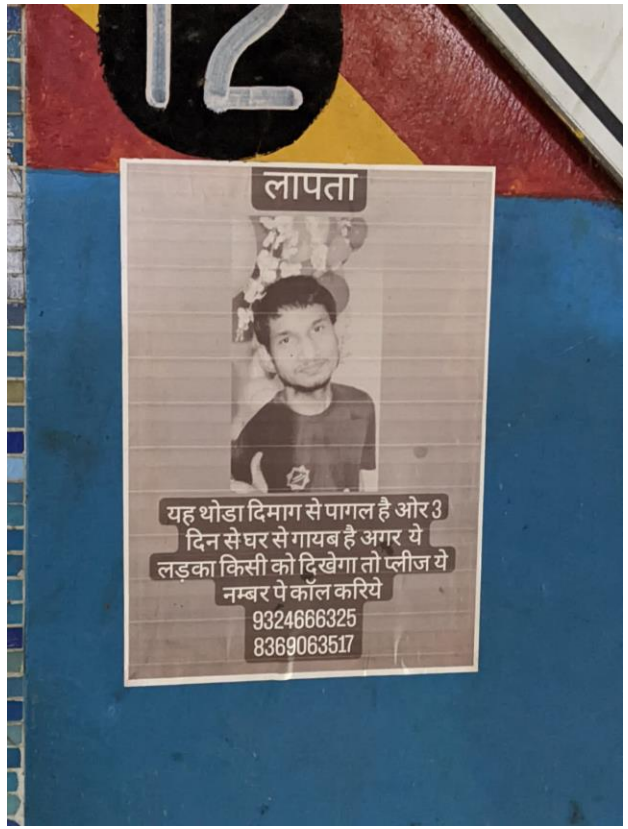





Face Detection: Full Images





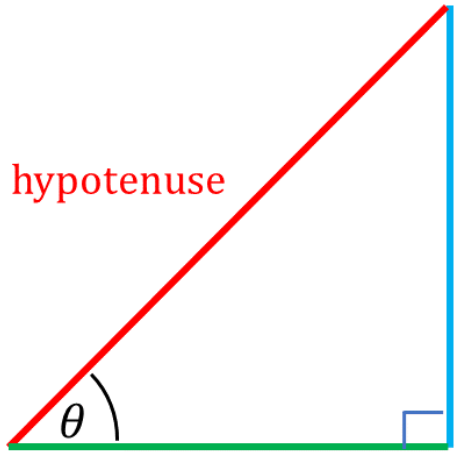


position & orientation of a face are normalized to a standard position & scale. Typically done by detecting facial landmarks such as corners of eyes/nose/mouth, then warping the image so that these landmarks are in fixed position.

Face Alignment: 2D Alignment

Some math.....

Note: 3D alignment also takes into account the shape of the face.



$$\sin\theta = \frac{\text{opposite}}{\text{hypotenuse}}$$

$$\cos\theta = \frac{\text{adjacent}}{\text{hypotenuse}}$$

This corrects variation in pose/lighting/etc. and once aligned, if can be compared to other faces using processing techniques like feature extraction.

Face Alignment: 2D Alignment

Some math.....

Face Alignment: 2D Alignment

Some math.....

Embedding Extraction and Face Recognition

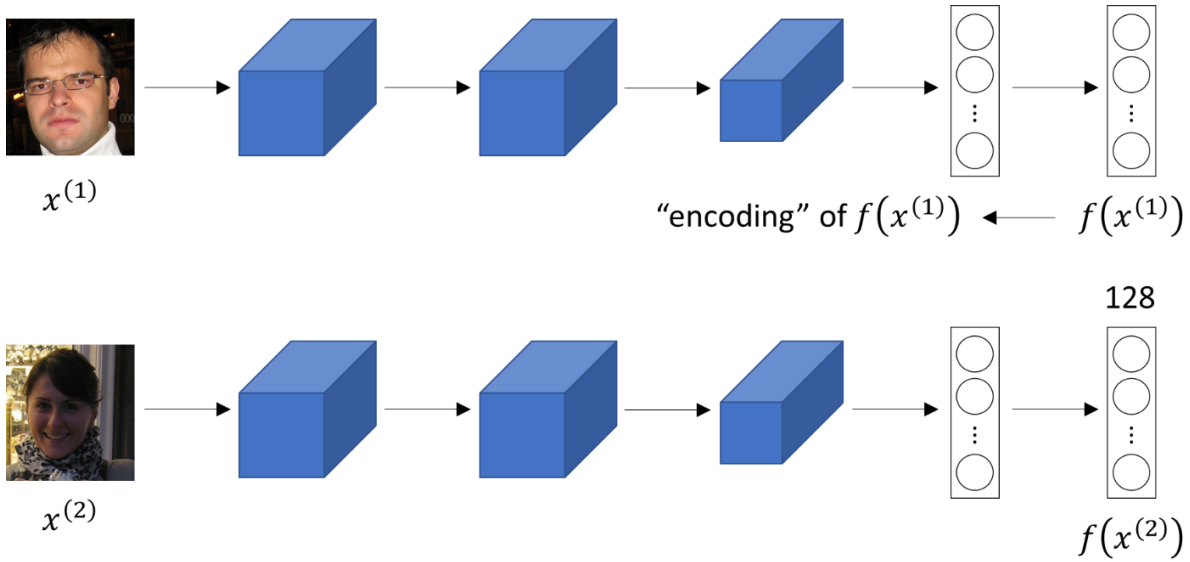
What kind of network should we train for this one shot learning/few shot learning task ?

1. A model from scratch ?
2. A pretrained model ?

compares 2 inputs & determines their similarity / dissimilarity.

Siamese Network

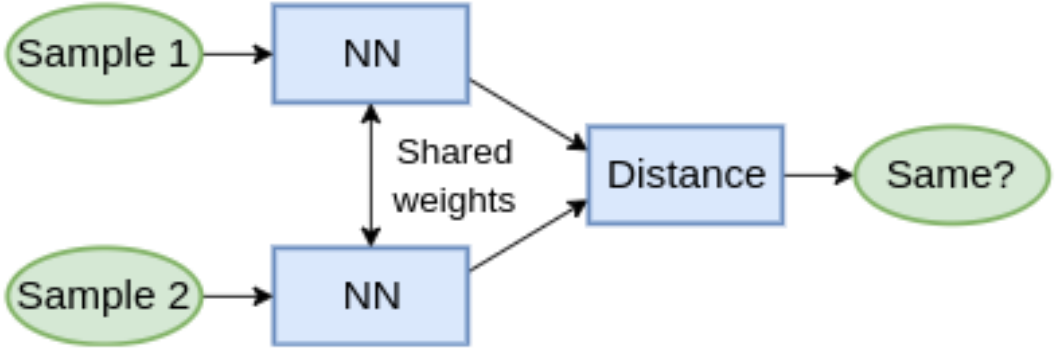
A **Siamese neural network** (sometimes called a **twin neural network**) is an artificial neural network that uses the same weights while working in tandem on two different inputs to compute comparable output vectors.



Siamese Network: L1/L2 Distance and Classification

Calculate the distance between the feature vectors

Pass this distance vector to the classification layer

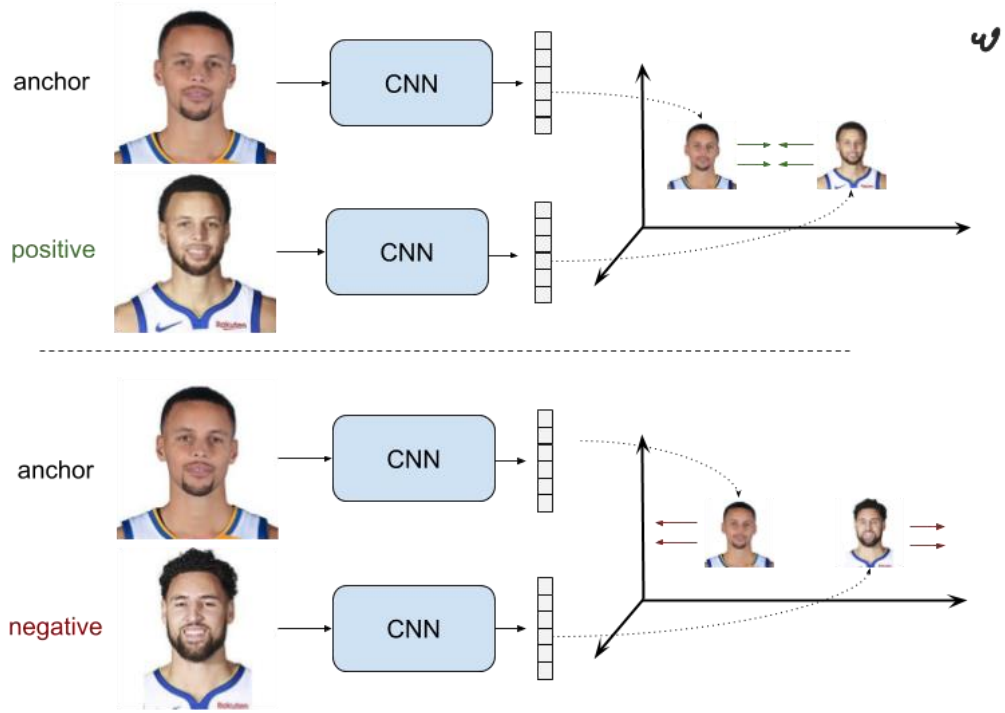


Use of a loss function that is tailored for a specific task.

Siamese Network: Custom Loss

1. Contrastive Loss
2. **Triplet Loss**

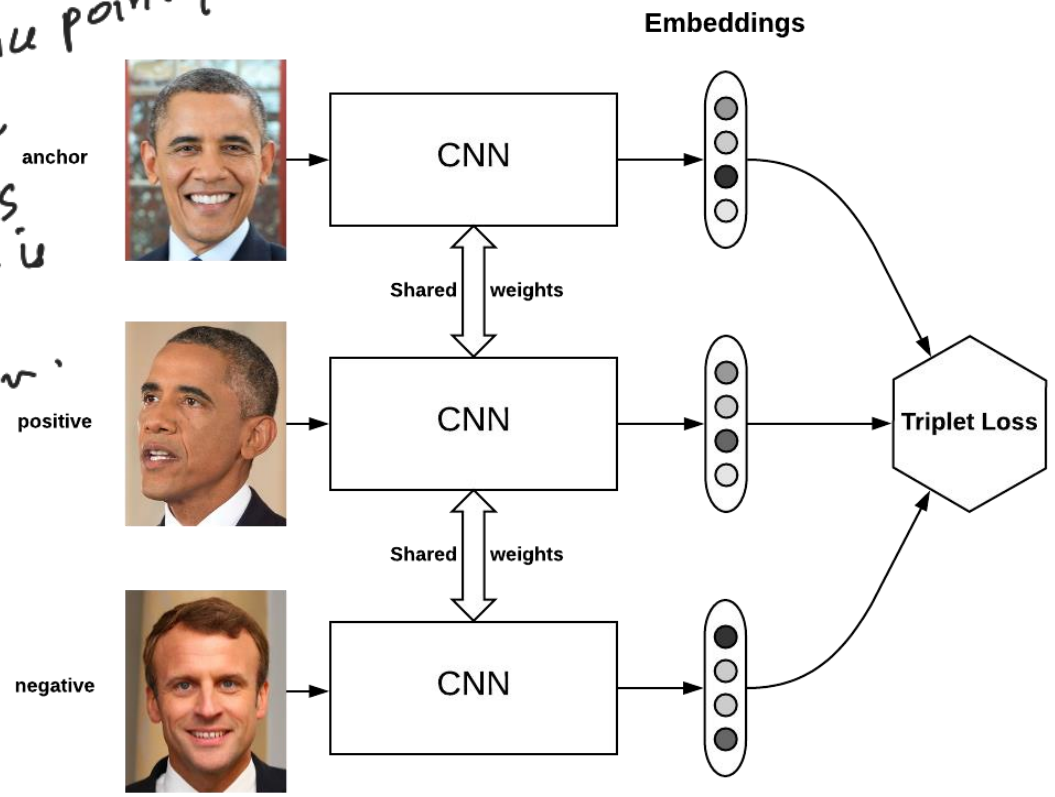
learn embeddings or feature representations for inputs such that similar inputs are mapped close to each other in embedding space while dissimilar ones are far apart.



objective of Loss function : minimise dist b/w anchor & positive,
maximise dist b/w anchor & negative.

Siamese Network: Triplet Loss

anchor is the reference point / template that exists in DB & is used for comparison.

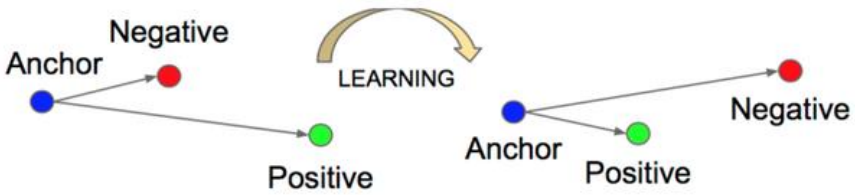


dist b/w anchor & positive input

hyperparameter that controls min dist. b/w anchor & negative input.

$$\mathcal{L} = max(d(a,p) - d(a,n) + margin, 0)$$

dist b/w anchor & negative input

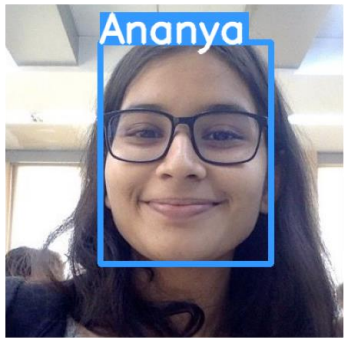


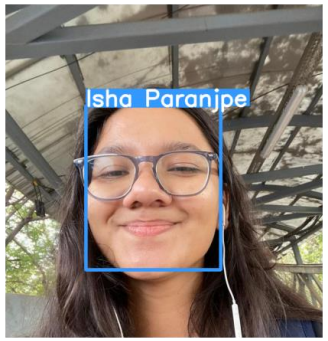
Important Aspects of Face Recognition Systems

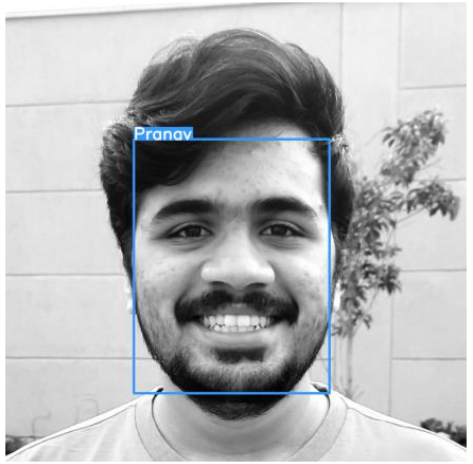
1. Fairness
2. Transparency
3. Accountability
4. Non-discrimination
5. Notice and Consent
6. Lawful Surveillance

Applications of Face recognition

1. Auto Tagging on social media platforms
2. Authentication via Face Verification
3. Missing Person Identification



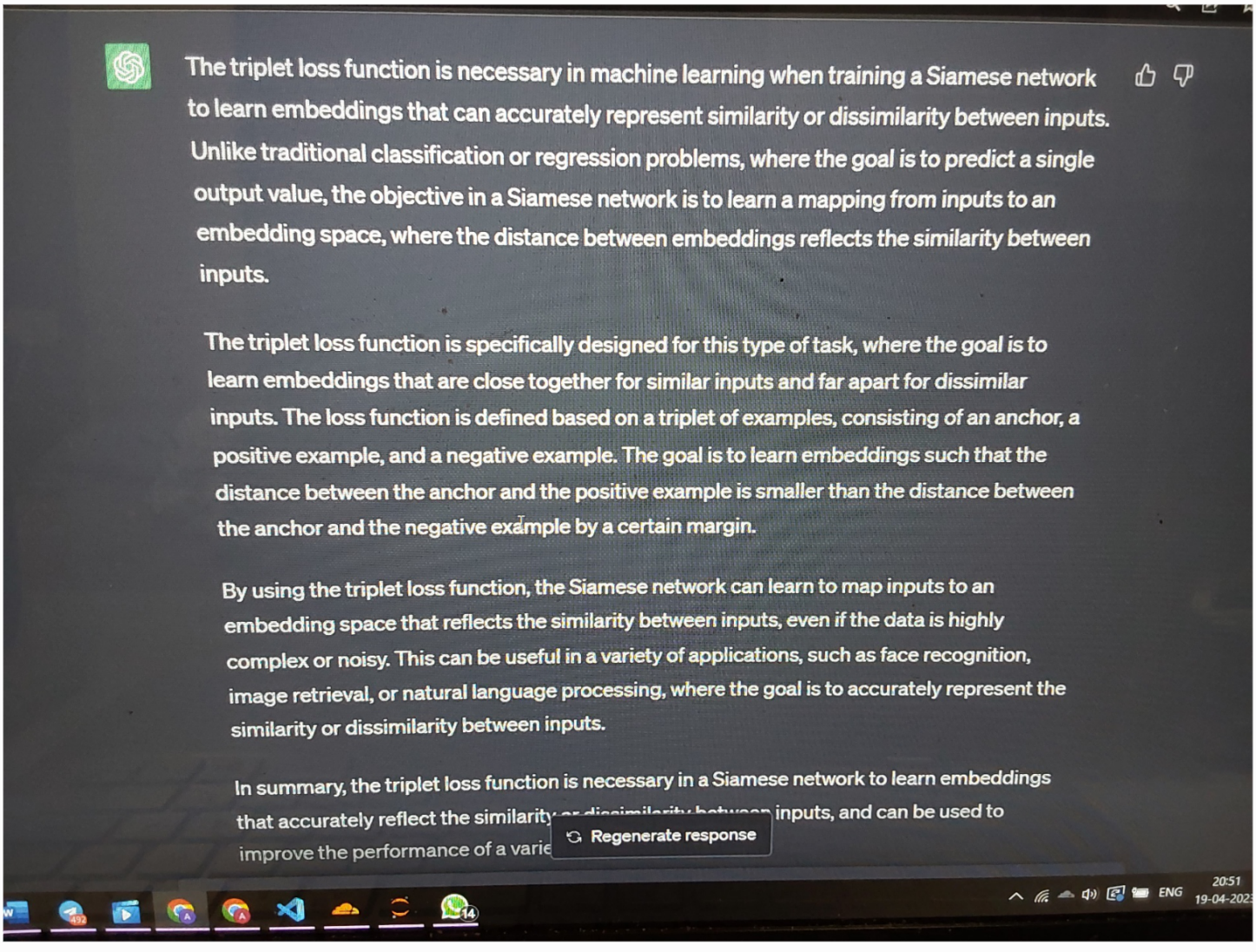




Thank you

For any queries drop an email at: quadeershaikh15.8@gmail.com

Triplet loss:



The triplet loss function is necessary in machine learning when training a Siamese network to learn embeddings that can accurately represent similarity or dissimilarity between inputs. Unlike traditional classification or regression problems, where the goal is to predict a single output value, the objective in a Siamese network is to learn a mapping from inputs to an embedding space, where the distance between embeddings reflects the similarity between inputs.

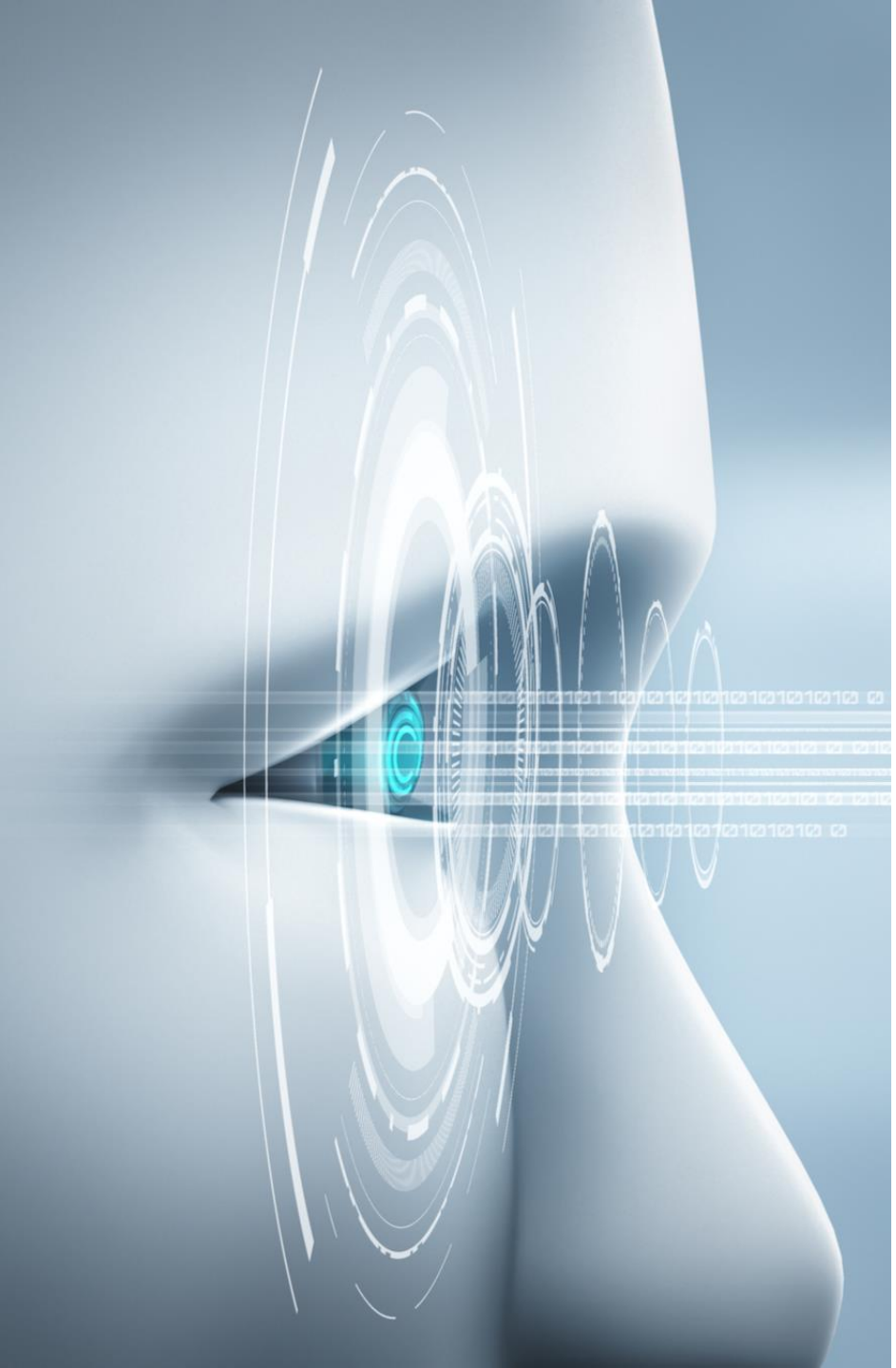
The triplet loss function is specifically designed for this type of task, where the goal is to learn embeddings that are close together for similar inputs and far apart for dissimilar inputs. The loss function is defined based on a triplet of examples, consisting of an anchor, a positive example, and a negative example. The goal is to learn embeddings such that the distance between the anchor and the positive example is smaller than the distance between the anchor and the negative example by a certain margin.

By using the triplet loss function, the Siamese network can learn to map inputs to an embedding space that reflects the similarity between inputs, even if the data is highly complex or noisy. This can be useful in a variety of applications, such as face recognition, image retrieval, or natural language processing, where the goal is to accurately represent the similarity or dissimilarity between inputs.

In summary, the triplet loss function is necessary in a Siamese network to learn embeddings that accurately reflect the similarity or dissimilarity between inputs, and can be used to improve the performance of a varie

Regenerate response

ENG 20:51 19-04-202



Computer Vision Applications

BY QUADEER SHAIKH

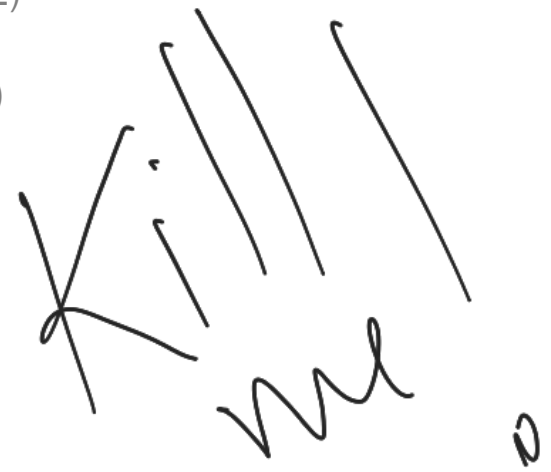
About me

Work Experience

- Risk Analyst
  - Morgan Stanley (Jan 2023 – Present)
- Data Science Intern
  - AkzoNobel Coatings International B.V. Netherlands (Feb 2022 – Dec 2022)
- Data Science Intern
  - EzeRx Health Tech Pvt. Ltd. (Jan 2022 – July 2022)
- Associate Engineer
  - Tata Communications Ltd. (July 2019 – Aug 2020)
- Network Automation and Analysis Engineer Intern
  - Cisco (June 2018 – July 2018)

Education

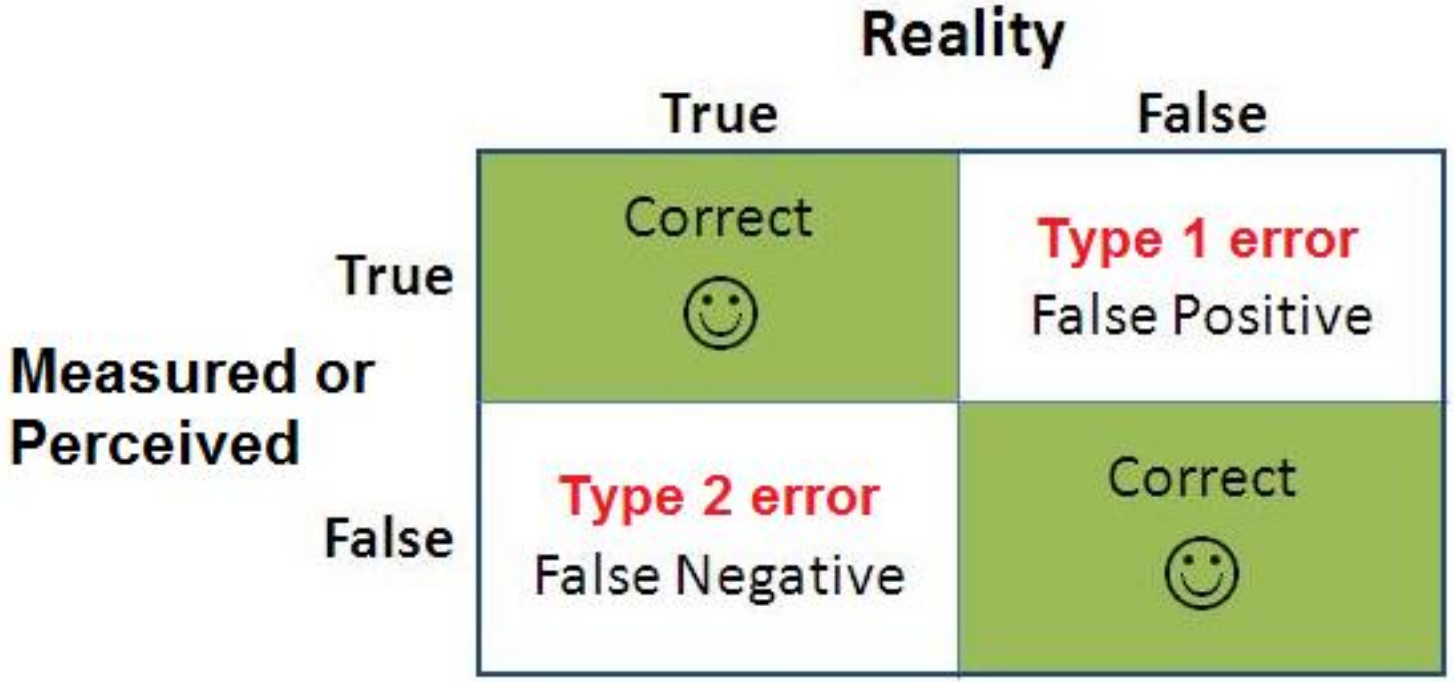
- M.Tech – Artificial Intelligence
  - NMIMS (2021 - 2023, currently pursuing)
- B.E. – Computer Engineering
  - Mumbai University (2015 - 2019)



Classification Models: Improvement and Usage in Video Processing Pipelines

Type I and Type II Error

Which is more dangerous ?



Type I and Type II Errors: Context Matters

You decide to get tested for COVID-19 based on mild symptoms. There are two errors that could potentially occur:

- **Type I error (false positive):** the test result says you have coronavirus, but you actually don't.
- **Type II error (false negative):** the test result says you don't have coronavirus, but you actually do.

more dangerous

You build a model for predicting if a credit card user is going to default or not. The bank that uses this model might want to take some precautionary measures based on the prediction.

- **Type I error (false positive):** the model result says user will default, but he actually doesn't.
- **Type II error (false negative):** the model result says user will not default, but he actually does.

Both can be dangerous but for different reasons.

Confusion Matrix

Decide on the threshold value for classification based on the confusion matrix

TP → correctly classified as True

FN → incorrectly classified as false.

Acquired Class	True Class T	True Class F
Y	True Positives (TP)	False Positives (FP)
N	False Negatives (FN)	True Negatives (TN)

$$\text{True Positive Rate (TPR)} = \frac{TP}{TP + FN}$$ ← all positive values

$$\text{False Positive Rate (FPR)} = \frac{FP}{FP + TN}$$ ← all negative values.

$$\text{Accuracy (ACC)} = \frac{TP + TN}{TP + FP + TN + FN}$$

Different Metrics for Classification

Recall Sensitivity True positive rate (TPR)	$\frac{TP}{FN+TP} = \frac{TP}{P}$
False positive rate (FPR) False alarm rate	$\frac{FP}{TN+FP} = \frac{FP}{N}$
Specificity True negative rate (TNR)	$\frac{TN}{TN+FP} = \frac{TN}{N} = 1 - FPR$
Precision	$\frac{TP}{TP+FP}$
False negative rate (FNR)	$\frac{FN}{FN+TP} = \frac{FN}{P}$
Accuracy	$\frac{TP+TN}{P+N} = \frac{TP+TN}{TP+TN+FP+FN}$

ROC (Receiver Operating Characteristic) Curve (Binary Classification)

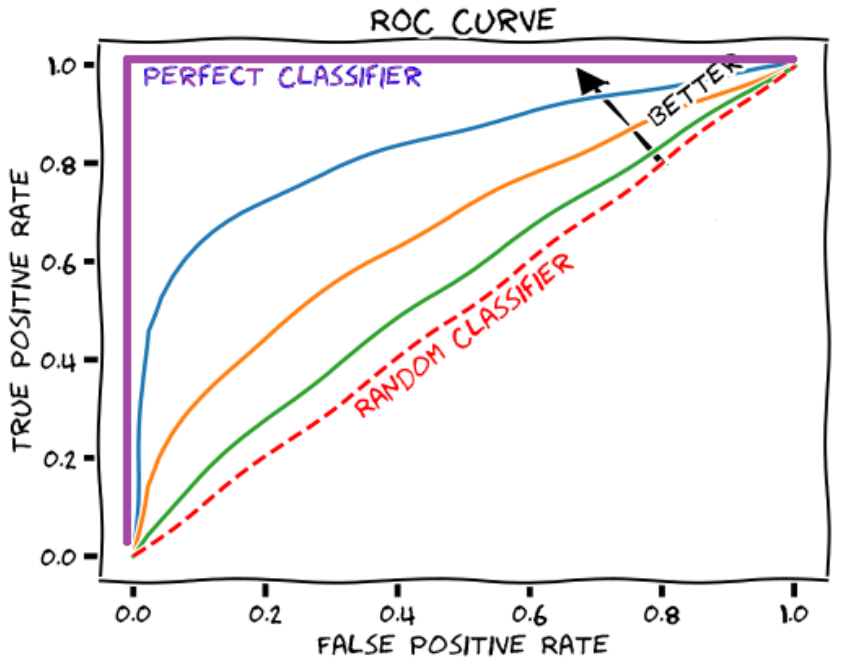
An **ROC curve** (**receiver operating characteristic curve**) is a graph showing the performance of a classification model at all classification thresholds. This curve plots two parameters:

- True Positive Rate
- False Positive Rate

predicted → real ↓	*Class_pos*	*Class_neg*
Class_pos	TP	FN
Class_neg	FP	TN

$$\text{TPR (sensitivity)} = \frac{TP}{TP + FN}$$

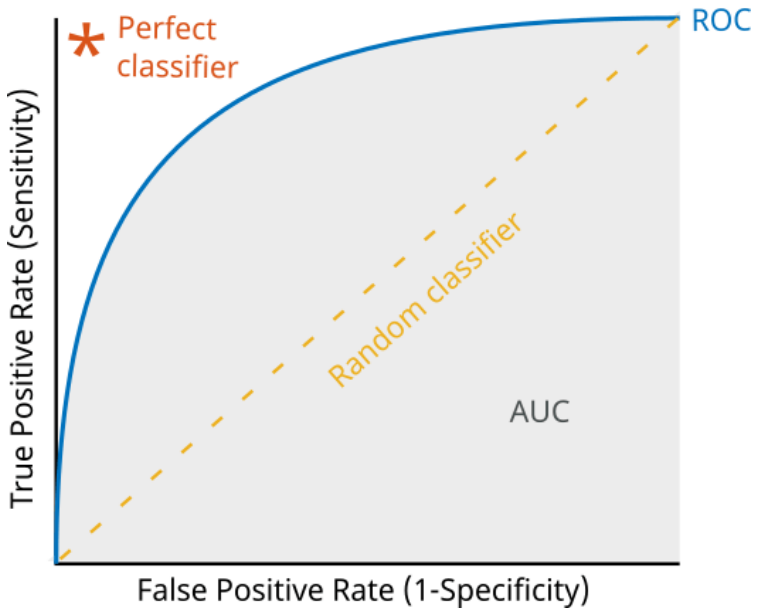
$$\text{FPR (1-specificity)} = \frac{FP}{TN + FP}$$



AUC (Area Under ROC Curve)

AUC provides an aggregate measure of performance across all possible classification thresholds.

AUC ranges in value from 0 to 1. A model whose predictions are 100% wrong has an AUC of 0.0; one whose predictions are 100% correct has an AUC of 1.0.



Losses in Classification Models

Binary Cross Entropy

$$L_{BCE} = -\frac{1}{n}\sum_{i=1}^{n}(Y_i \cdot \log \hat{Y}_i + (1 - Y_i) \cdot \log (1 - \hat{Y}_i))$$

Categorical Cross Entropy

$$\text{Loss} = -\sum_{i=1}^{\substack{\text{output} \\ \text{size}}} y_i \cdot \log \hat{y}_i$$

Why should one monitor validation loss while saving the best model instead of accuracy?

Which Model is better ?

Model 1		P(A)	P(B)	P(C)		GT(A)	GT(B)	GT(C)		Actual label	Predicted Label
	1	0.55	0.35	0.1		1	0	0		A	A
	2	0.3	0.5	0.2		0	1	0		B	B
	3	0.6	0.35	0.05		1	0	0		A	A
	4	0.3	0.3	0.4		0	0	1		C	C

Model 2		P(A)	P(B)	P(C)		GT(A)	GT(B)	GT(C)		Actual label	Predicted Label
	1	0.8	0.1	0.1		1	0	0		A	A
	2	0.5	0.4	0.1		0	1	0		B	A
	3	0.75	0.2	0.05		1	0	0		A	A
	4	0.05	0.2	0.75		0	0	1		C	C

Which Model is better ?

Space for Quadeer’s lethal mathematical skillzzzz

How to get more confident predictions ?

Different Ensembling techniques

1. Bagging
2. Boosting
3. Stacking
4. Voting
    i. Hard Voting: Takes the mode of predictions of different classifiers
    ii. Soft Voting: Takes the probability average of different classifiers and then decides a class

Video Processing Pipelines using Classification Models

Naïve Video Processing Pipeline

1. Give a video input source
2. Read each frame of the video
3. Classify each frame in the video (Bottleneck)
4. Display the prediction on the video

Video Processing Pipelines using Classification Models

MULTI THREADING (AKA CONCURRENT PROCESSING)

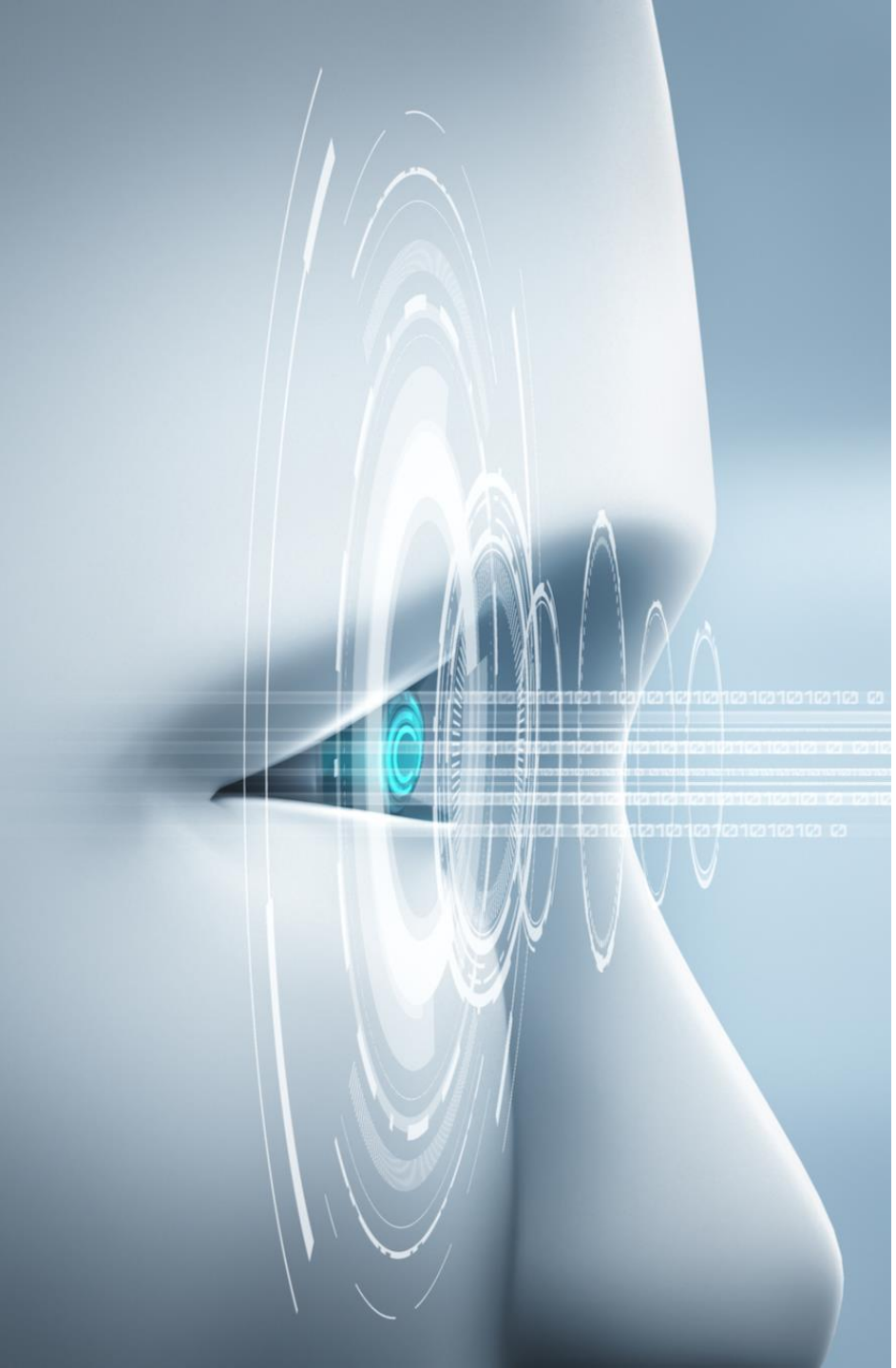
1. Multiple segments of a program/process is created to speed up the task at hand
2. Should be used when your task is I/O bound i.e. receiving an input continuously, processing it and then displaying it is a bottleneck.
3. E.g. You type sentences in a word file, and your spell check runs concurrently without blocking you from typing the next word.
4. E.g. Your model classifies each frame of the video and your program should not have to wait for fetching the next video frame thus blocking and slowing down the whole process

MULTIPROCESSING

1. A program/process is subdivided into multiple processes that can speed up a task by running this multiple subdivided processes on different processors/CPU cores.
2. Should be used when your task is computationally heavy and requires a lot of processors for computation.
3. E.g. Utilizing multiple cores of a GPU to train your deep learning models

Thank you

For any queries drop an email at: quadeershaikh15.8@gmail.com



Computer Vision Applications

BY QUADEER SHAIKH

About me

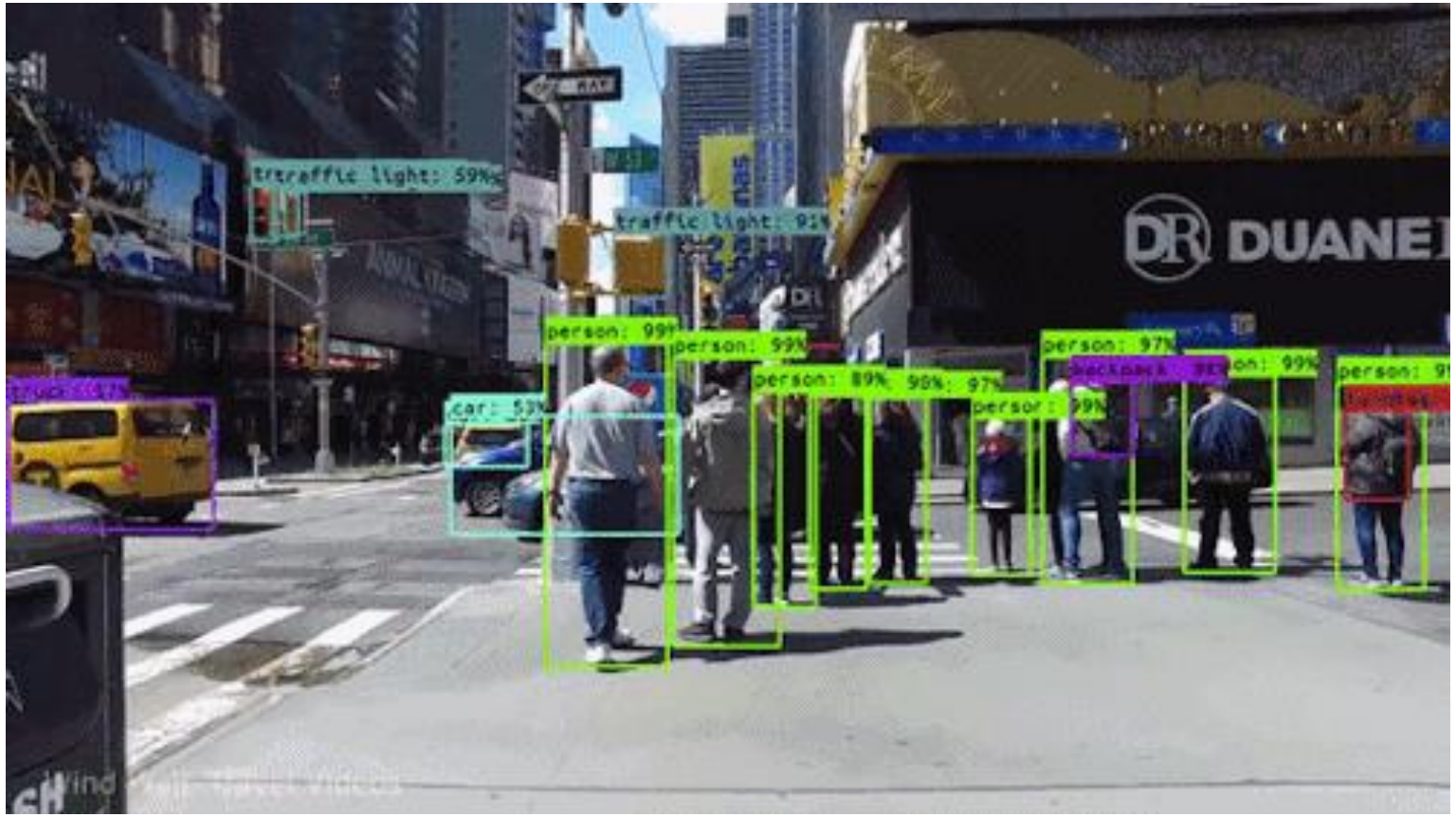
Work Experience

- Risk Analyst
  - Morgan Stanley (Jan 2023 – Present)
- Data Science Intern
  - AkzoNobel Coatings International B.V. Netherlands (Feb 2022 – Dec 2022)
- Data Science Intern
  - EzeRx Health Tech Pvt. Ltd. (Jan 2022 – July 2022)
- Associate Engineer
  - Tata Communications Ltd. (July 2019 – Aug 2020)
- Network Automation and Analysis Engineer Intern
  - Cisco (June 2018 – July 2018)

Education

- M.Tech – Artificial Intelligence
  - NMIMS (2021 - 2023, currently pursuing)
- B.E. – Computer Engineering
  - Mumbai University (2015 - 2019)

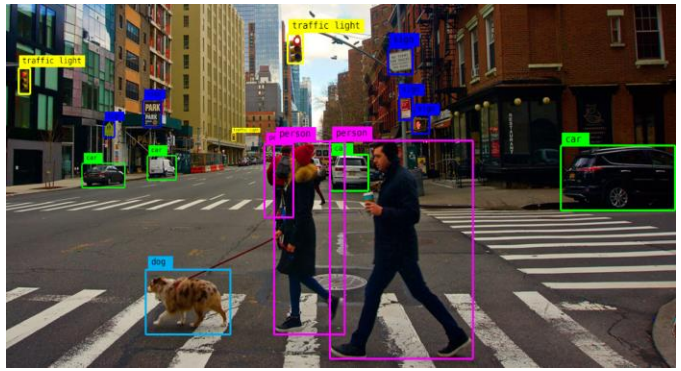
Object Detection

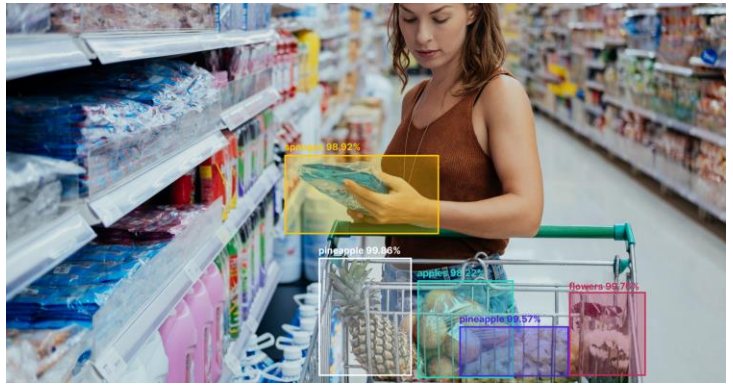


classification & localization ⇒ Identify what class an object belongs to, but also its position.

What is object detection ?

- Object detection is a computer technology related to computer vision and image processing that deals with detecting instances of semantic objects of a certain class in digital images and videos.
- It is a computer vision technique for locating instances of objects in images or videos. Object detection algorithms typically leverage machine learning or deep learning to produce meaningful results.
- 17 interesting applications: https://alwaysai.co/blog/object-detection-for-businesses

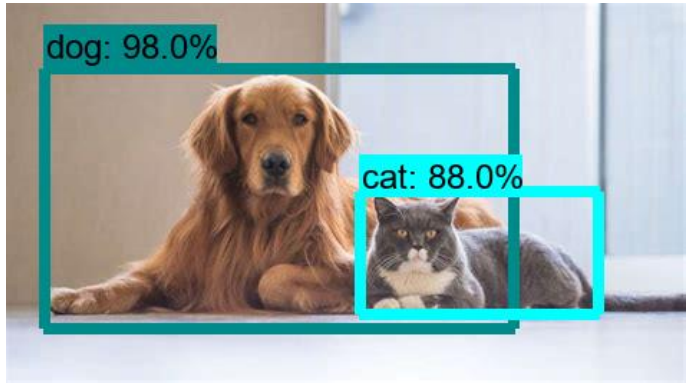




output of object detection → probability of object belonging to a class, with 4 continuous values that signify position of object.

How do you localize an object in an image ?

- Ermm..... Maybe we can plot a rectangle that bounds an object of interest ? And hence, **Bounding Box**
- We know how to predict a class of an image, but how do we go about predicting a bounding box ?
- How do we create a rectangle/bounding box in computer vision using tools like opencv at all ?

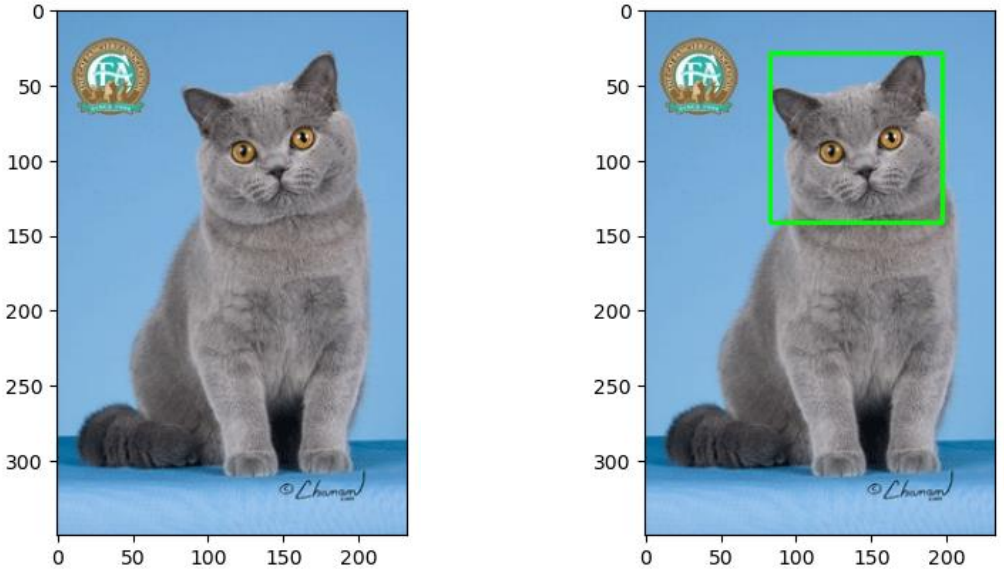


Bounding Box Regression

- ***Syntax:*** *cv2.rectangle(image, start_point, end_point, color, thickness)*
- *Where, start_point = (xmin, ymin) and end_point = (xmax, ymax)*
- So we perform a regression task to predict 4 values of 2 pairs of coordinates
  - xmin, ymin, xmax, ymax

Bounding Box Regression

- Example: This is an image of Sir. Jim Radicliffe, a British Shorthair
- Image Dimensions: Height is 350, Width is 233, Colored image so Depth is 3
- Rectangle Coords: 83,29,197,142

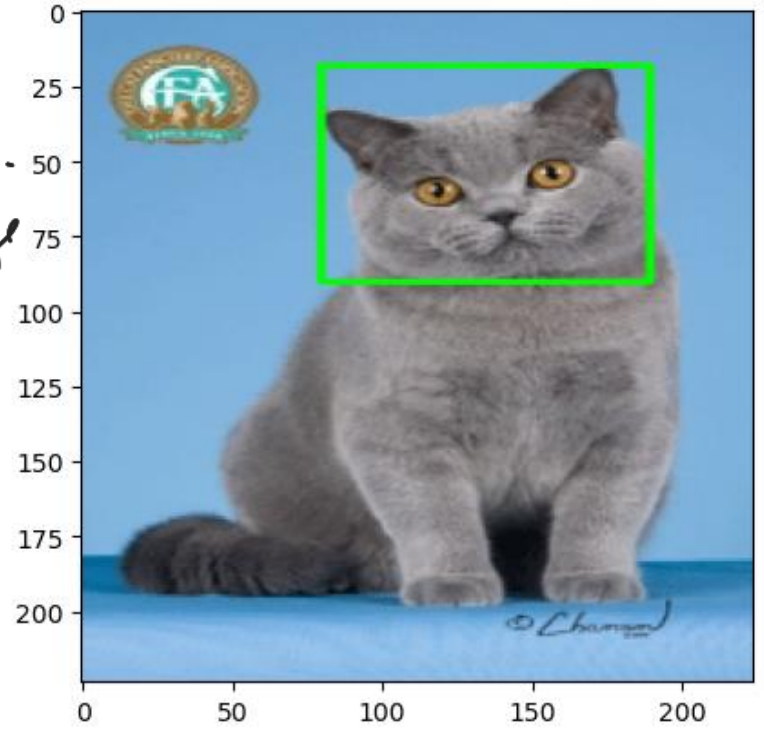


Bounding Box Regression

But don't we feed a fixed size image input to our CNNs ? Maybe something like 224x224x3

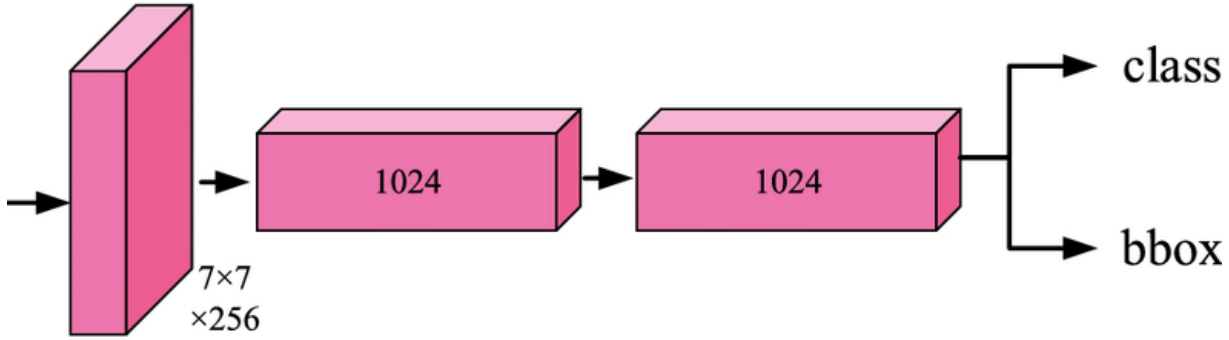
- Example: This is an image of Sir. Jim Radicliffe, a British Shorthair
- Image Dimensions: Height is 350, Width is 233, Colored image so Depth is 3
- Rectangle Coords: 83,29,197,142
- That is why we normalize the coordinates
  - xmin: 83/233 (xmin/width)
  - ymin: 29/350 (ymin/height)
  - xmax: 197/233 (xmax/width)
  - ymax: 142/350 (ymax/height)
- If we resize the above image to 224x224x3
  - xmin = int(xmin*224)
  - ymin = int(ymin*224)
  - xmax = int(xmax*224)
  - ymax = int(ymax*224)

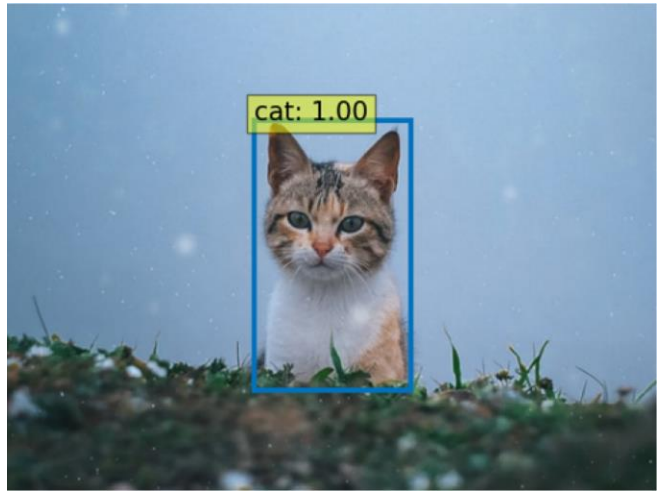
↳ scaling them relatively to the size of the image.
- done to introduce scale-invariance.
- better training → network can converge faster.
- ensures consistency between different models / datasets.



Classification with Bounding Box Regression

Class predictor maybe a sigmoid or a softmax depending on the number of class of objects that we are predicting

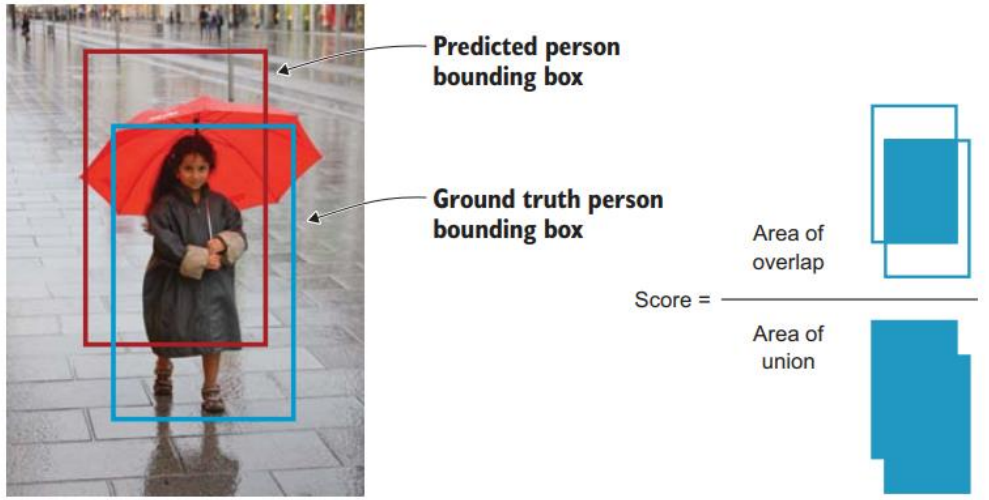




How to assess the prediction of Bbox ?

Intersection over Union

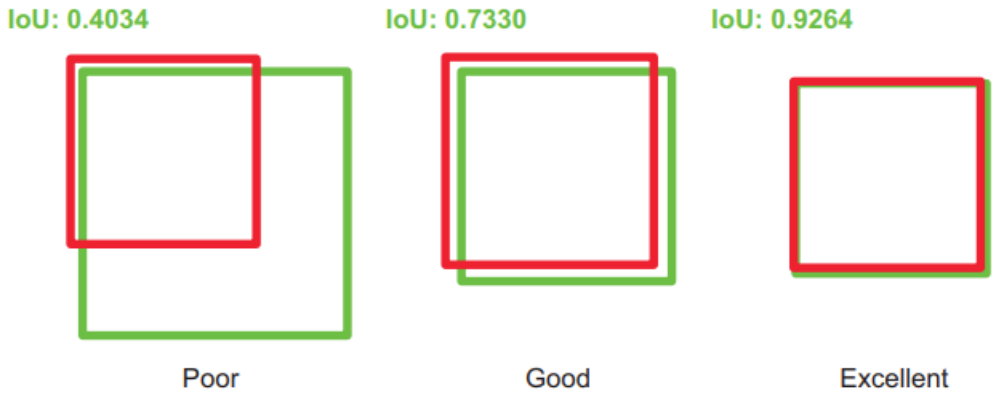
This measure evaluates the overlap between two bounding boxes: the ground truth bounding box (Bground truth) and the predicted bounding box (Bpredicted). By applying the IoU, we can tell whether a detection is valid (True Positive) or not (False Positive)



How to assess the prediction of Bbox ?

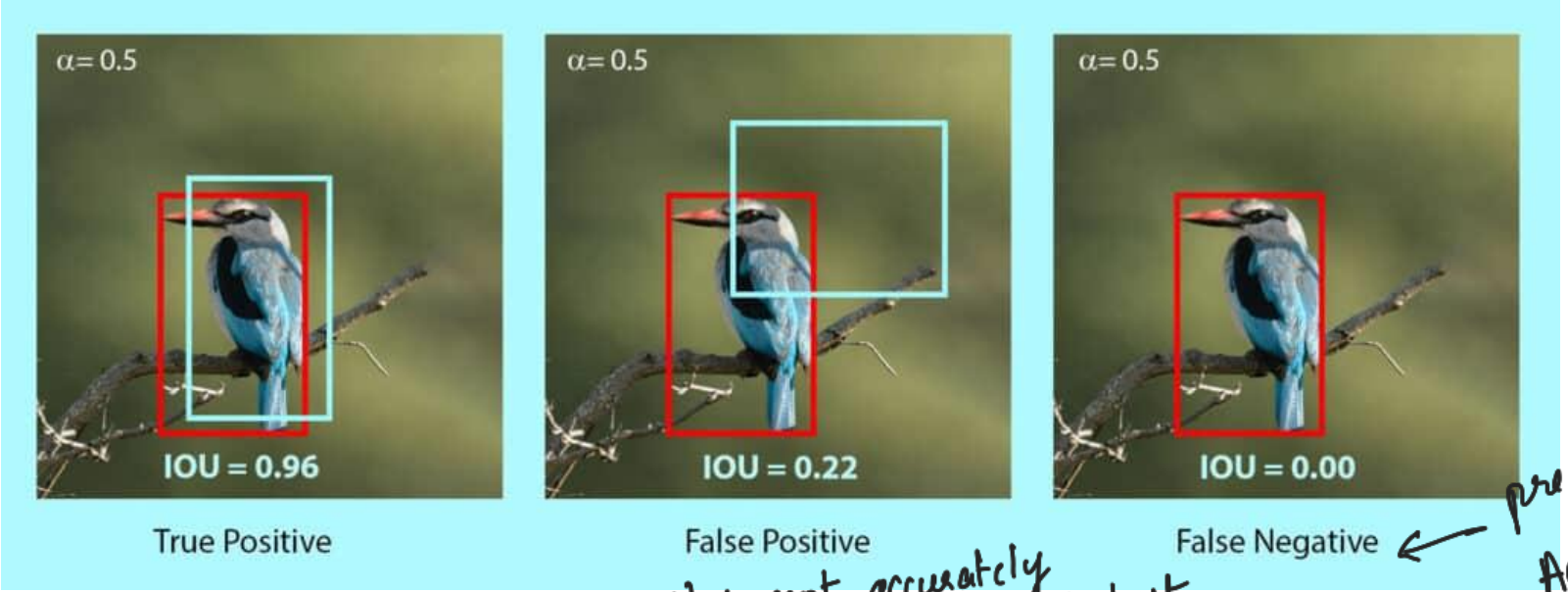
Intersection over Union

The intersection over the union value ranges from 0 (no overlap at all) to 1 (the two bounding boxes overlap each other 100%). The higher the overlap between the two bounding boxes (IoU value).



Confusion Matrix for IoU

Note: True Negative is not applicable to object detection. True Negative means it is correctly detecting the background of an object as background. Which is equivalent to not detecting anything.

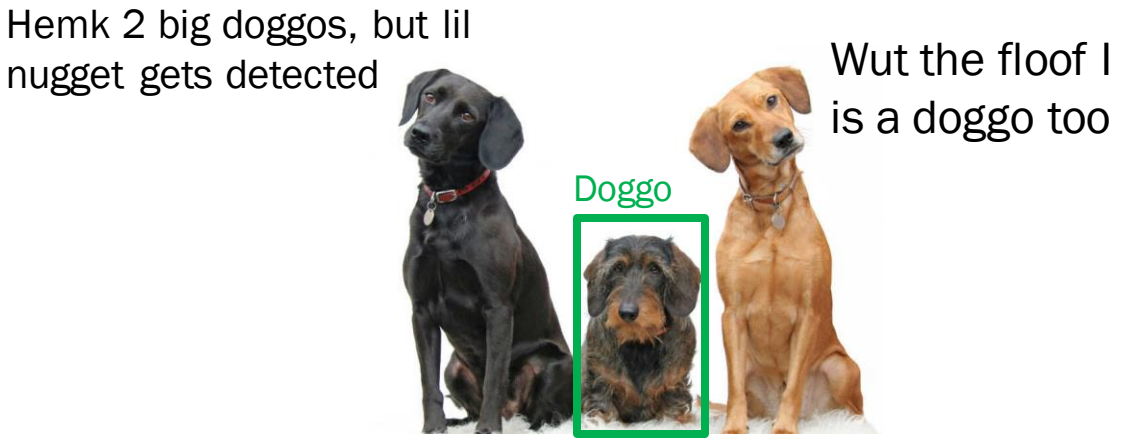


prediction → there is no bird
Actual → there is a bird.

↳ it is not accurately detecting the bird but is saying it is a bird.

Is object detection really that simple ?

- Bounding box regression with classification module works out fine if you have just one object in an image and you want to detect one object
- If you have multiple objects of a similar class in an image it will only detect one of them which it is most confident about
- How do we address this issue ?



Take a trip down the memory lane

What did we stumble upon in our fundamental computer vision modules that could help us find the objects of interest ?

Take a trip down the memory lane

What did we stumble upon in our fundamental computer vision modules that could help us find the objects of interest ?

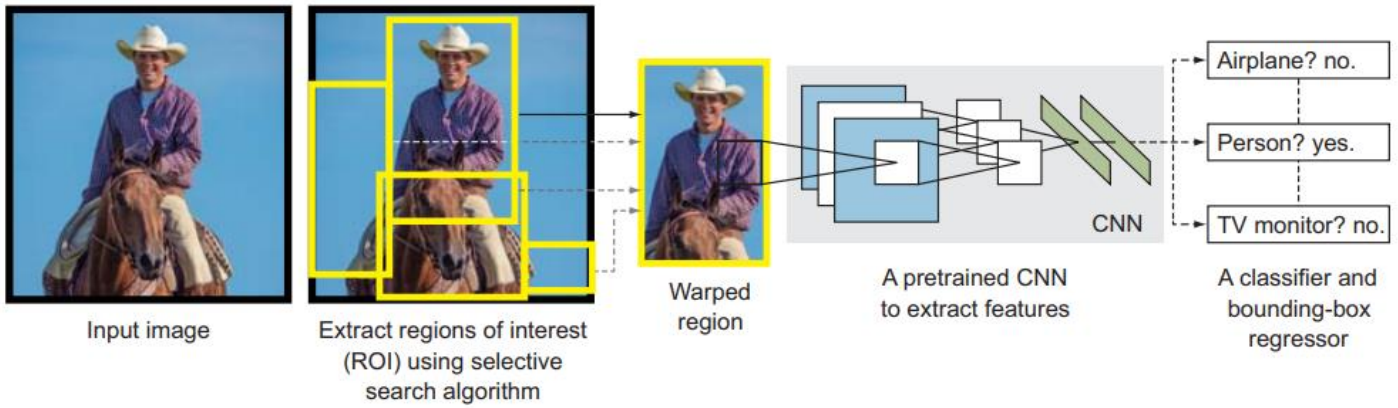
Objects of interest.....mmm..... **Regions of Interest (ROI)** ?!

Region based CNN Models

1. RCNN (2013)
2. Fast RCNN (2015)
3. Faster RCNN (2015)

RCNN

1. It is the first region based architecture in the region proposal based object detection family
2. It was one of the first large, successful applications of convolutional neural networks to the problem of object detection and localization
3. The approach was demonstrated on benchmark datasets, achieving then-state-of-the-art results on the PASCAL VOC-2012 dataset and the ILSVRC 2013 object detection challenge



RCNN Components

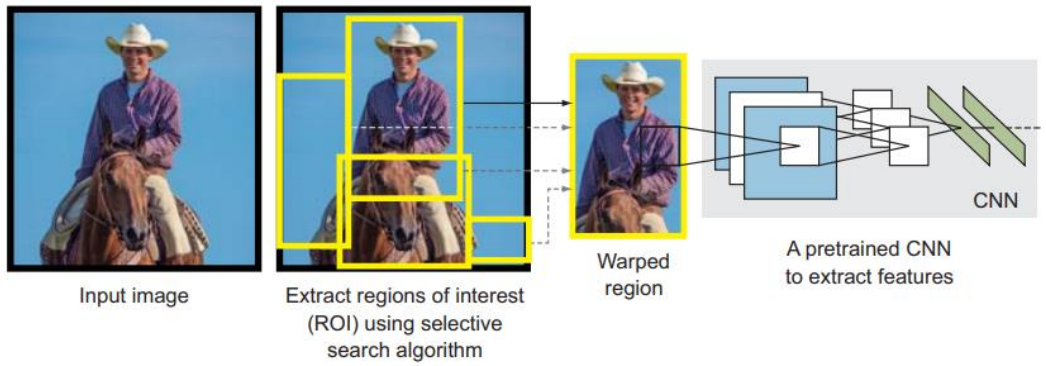
1. Region Proposal: Extract Regions of Interest (ROI)
2. Feature Extraction Module
3. Classification Module
4. Localization Refinement Module (optional component)

RCNN – Region Proposal

- The regions proposed in this step have a very high probability of containing an object
- An algorithm called *selective search* scans the input image to find regions that contain blobs and proposes RoIs (2000 RoIs in fast mode, approx. 9000 in regular mode) to be processed by the next components of the RCNN.
- The proposed RoIs are of different sizes
- But since a CNN is fed an image of fixed size we warp the RoIs to a fixed size
- **Note:** Since the selective search algorithm proposes region proposal bounding boxes, bounding box regression becomes an optional component

RCNN – Feature Extraction

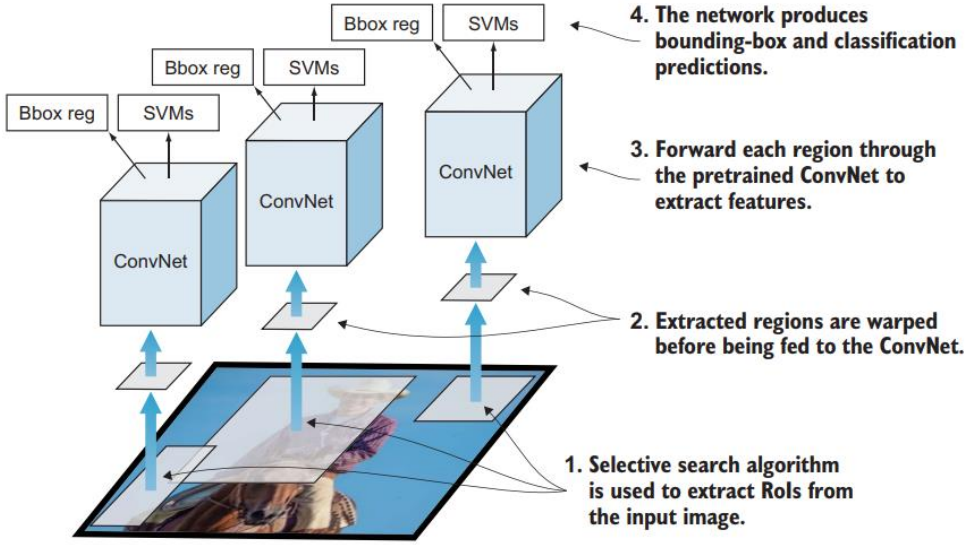
- We run a pretrained CNN (on imagenet) on top of the region proposals to extract features from each of these candidates
- However, since the proposed regions vary in size and are typically warped to fixed size, an imagenet pretrained CNN's feature extraction offers a mediocre/poor performance
- Therefore, the pretrained network is finetuned on the warped region proposals using the softmax classification output layer
- AlexNet/VGG-Net (initial version of VGG-16/19) was used as a pretrained network



usual use of pretrained model + fine tuning with softmax

RCNN – Classification Module & Bbox Refinement

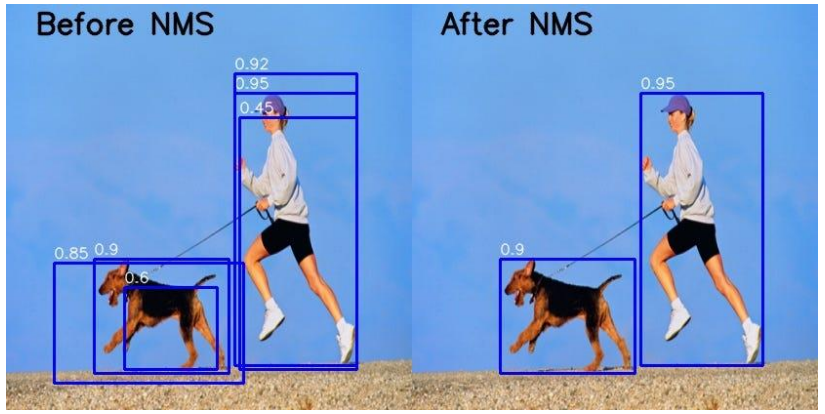
- We train a Linear SVM to perform classification on the features extracted using the pretrained network on the candidate region proposals
- Linear SVM performs a one vs rest classification to classify an object into a particular class
- Since Selective Search takes care of the localization using its own proposed regions bounding box we do not explicitly perform the bounding box regression step
- But to improve the localization step further we include the bounding box regression to give more refined coordinates of the bounding box

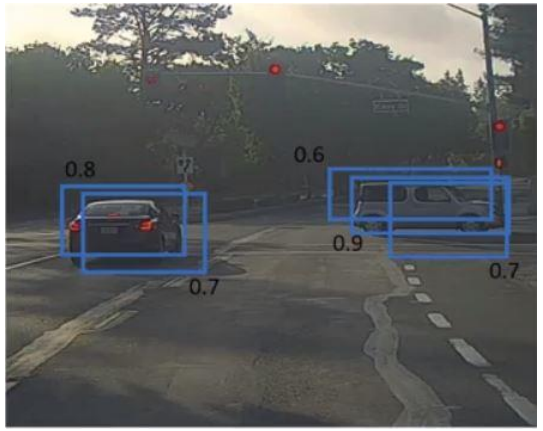


• selecting a single entity out of many overlapping entities
• discard the entities that are below a probability bound.

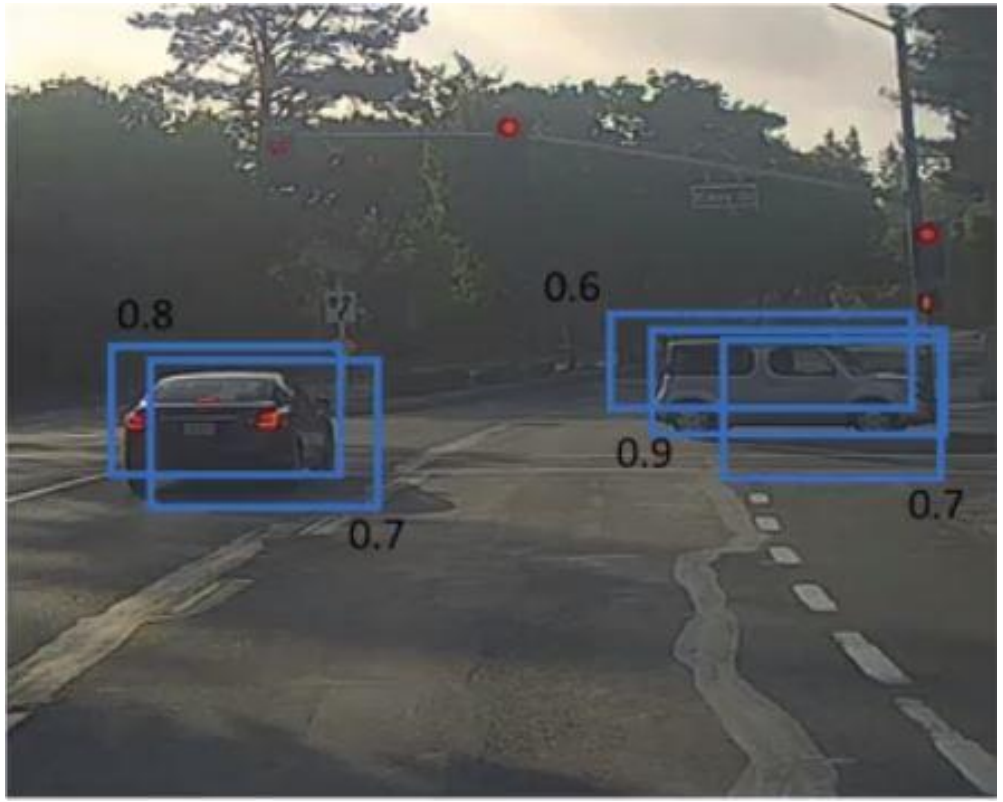
Multiple Bbox single object: NMS

- Non max suppression is a technique used mainly in object detection that aims at selecting the best bounding box out of a set of overlapping boxes
- Sort the bboxes in descending order of class confidence/probability score
- remove the boxex which have a confidence score < threshold confidence score (e.g. remove boxes with score < 0.5)
- Loop over the remaining boxes
  - Start with the box which has the highest confidence
  - Discard other boxes if they have an IoU > predefined IoU thresh with the highest confidence box
  - Repeat the steps until all the boxes are either taken as output prediction or discarded





Multiple Bbox single object: NMS

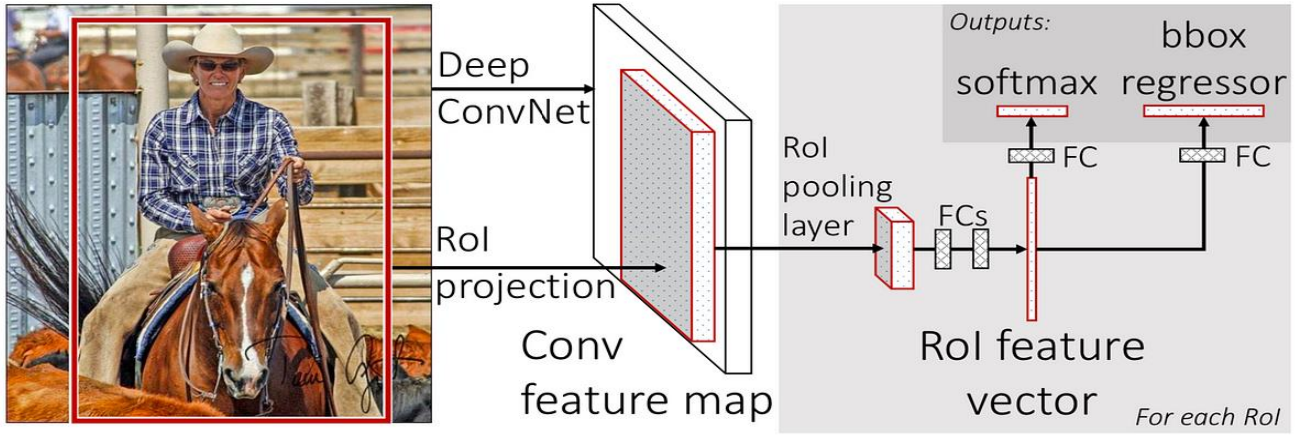


Disadvantages of RCNN

1. Object Detection is very slow
   - For each image selective search proposes 2000 RoIs to be examined by the entire pipeline. These 2000 RoIs are then passed on to the CNN 2000 times for a single image i.e. 2000 forward propagations for a single image. This process makes it computationally very expensive. During inference object detection in a single image takes 50 seconds (47.5 seconds for CNN + time taken for selective search)
2. Multiple Training stages (3 stages)
   - Finetuning of CNN feature extraction
   - Training of object classes using SVMs
   - Training of Bbox regressors for bounding box refinement
3. Two stage pipeline
   - First stage is for Region Proposals
   - Second stage is for Object Detection on Region Proposals
4. Training is expensive in terms of both time and space
   - 2000 RoIs per image need to finetuned
   - 2000 RoIs per image need to be stored on the disk

Fast RCNN (Inspired by SPPNet: Spatial Pyramid Pooling Network)

- Since feeding 2000 RoIs to the CNNs 2000 times for a single image was a bad idea the author of the paper on RCNN came up with a new architecture
- Bounding box coordinate format: (x_c, y_c, w, h)
  - x_c – x center, y_c – y center, w – width, h - height

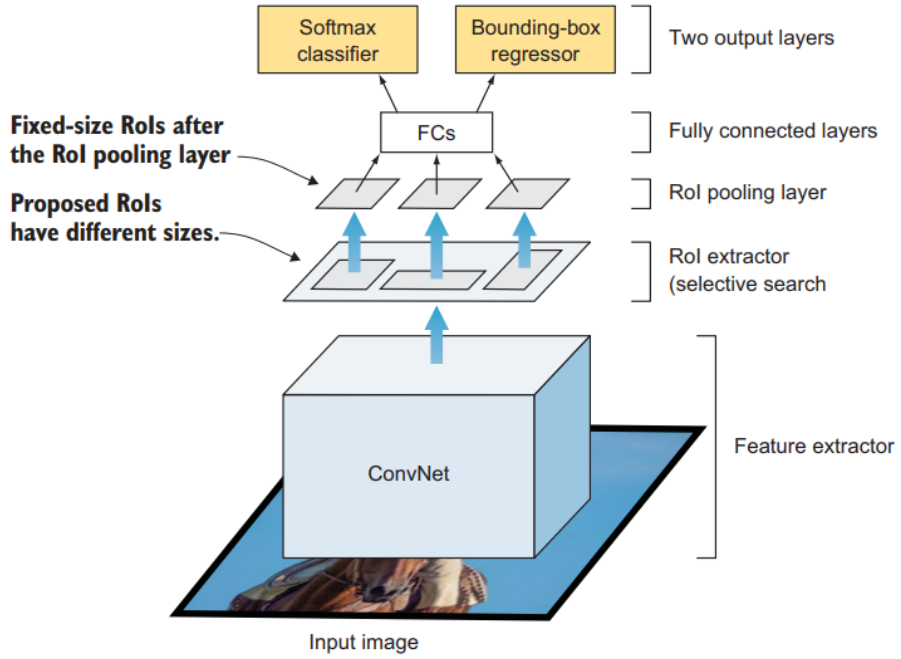


Check from theory notes.

ROI Pooling:

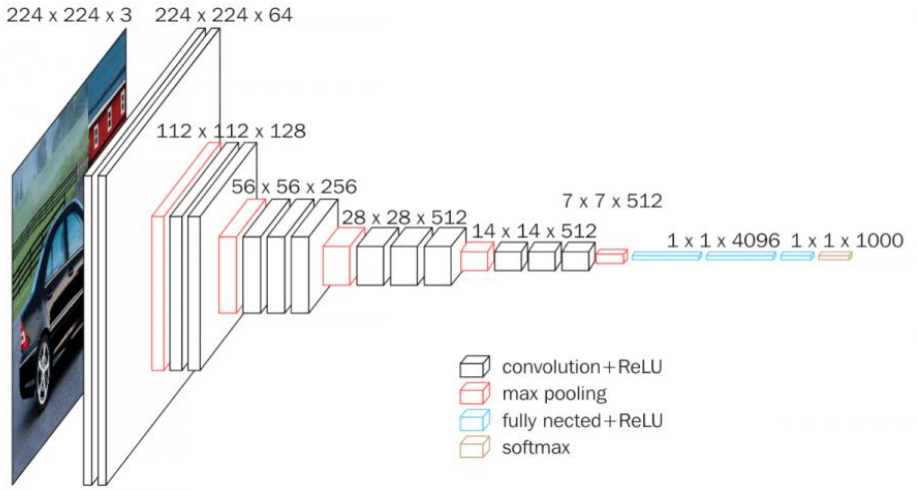
Fast RCNN

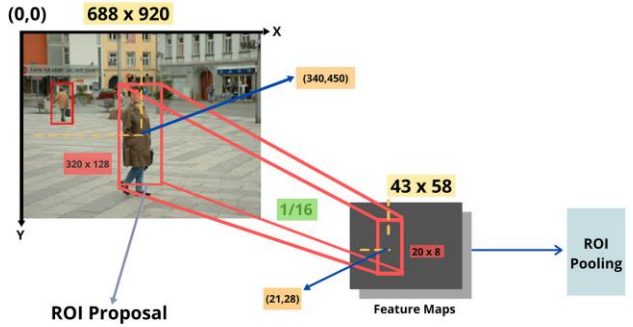
1. Pass the input image to the feature extractor just once
2. Pass the input image to selective search algorithm to get RoIs/Region Proposals
3. Feature maps are produced by the pooling layer of the pretrained network
4. RoIs from step 2 are projected onto the feature maps
5. RoI pooling is performed on the projected RoIs
6. RoI pooling output is passed to the fully connected layers
7. Fully connected layer is connected to the multi-head output
    1. Classifier
    2. Bbox Regressor



RoI Projection

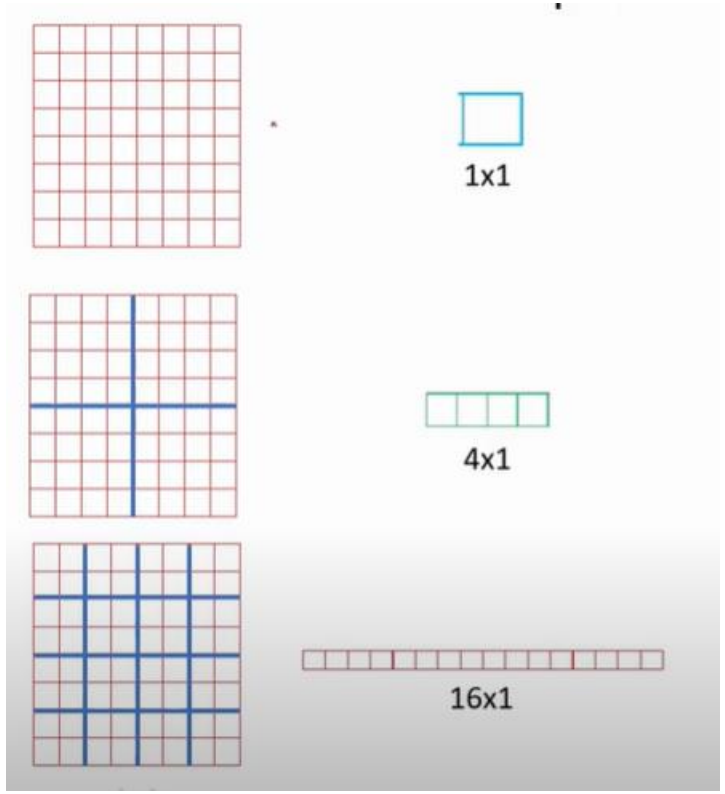
- The final max pooling layer produces a feature map of size 7x7x512
- An image of size 224x224 gets downsampled to 7x7, therefore we can calculate its **subsampling ratio** by 224/7 = **32**
- Any object that is present in the image of size 224x224 can be projected to the feature map of size 7x7 if we divide the height and width of that object by **32.**





ROI Pooling

- The projected RoIs on the feature maps are then pooled and passed on to the fully connected layers
- But since the projected RoIs will vary in size we need to do something about it
- We perform a special type of pooling known as ROI pooling.
- ROI pooling takes the max value from each grid
- We use a 7x7 grid to perform ROI pooling on a feature map which will produce a vector of size 49x1
  - 1x1 grid ROI pooling will produce 1x1 vector
  - 2x2 grid will produce a 4x1 vector
  - 4x4 grid will produce a 16x1 vector



Classification and Bbox Regression Module

1. Classification: Softmax classifier is used for classification prediction
2. Bbox Regression: Offsets between the selective search region proposals and ground truth bounding boxes are predicted for each class (this method is known as Relative Bounding Box Regression)
    1. tx = x_center_gt – x_center_ss
    2. ty = y_center_gt – y_center_ss
    3. tw = w_gt – w_ss
    4. th = h_gt – h_ss

Loss function

Fast R-CNN

- Multi-task loss

$$L(p, u, t^u, v) = L_{\text{cls}}(p, u) + \lambda[u \geq 1] L_{\text{loc}}(t^u, v)$$

Where $$L_{\text{loc}}(t^u, v) = \sum_{i \in \{\text{x,y,w,h}\}} \text{smooth}_{L_1}(t_i^u - v_i)$$

$$\text{smooth}_{L_1}(x) = \begin{cases} 0.5x^2 & \text{if } |x| < 1 \\ |x| - 0.5 & \text{otherwise} \end{cases}$$

Advantages and Disadvantages of Fast RCNN

ADVANTAGES

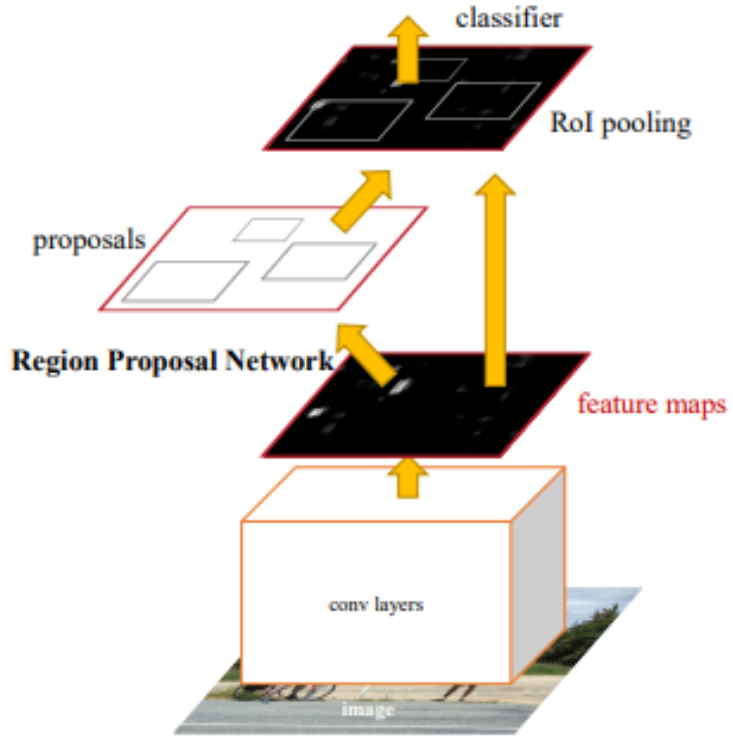
1. Requires only one propagation for each image unlike 2000 in RCNN
2. Three stage training from RCNN reduced to one stage training
3. Faster in inference time than RCNN: 2 seconds (0.32 seconds for CNN + time required for selective search)

DISADVANTAGES

1. It is still a two stage pipeline
    1. Selective Search Region Proposal
    2. CNN
2. Selective search still remains to be a bottleneck as it is very slow to generate region proposals. 2 seconds of inference time is too much for real time object detection in videos.

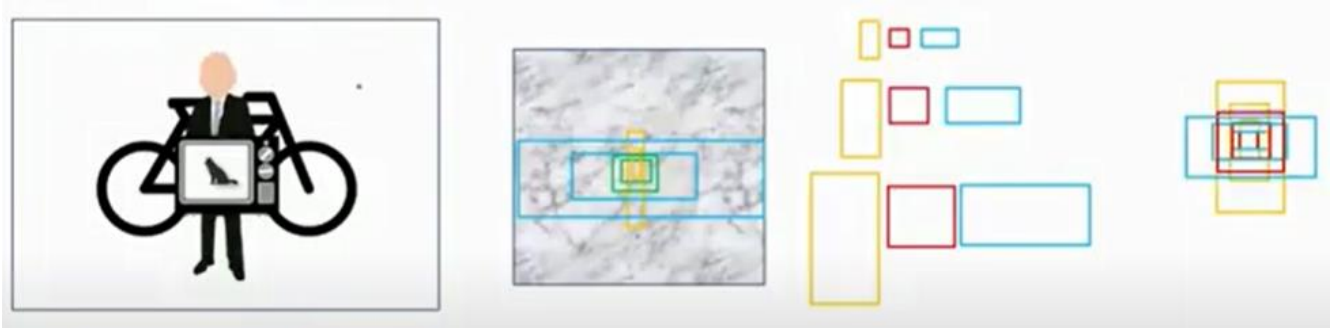
Faster RCNN

- We now know that the major bottleneck with region based CNNs till now is the traditional and slow region proposal method (selective search)
- We need to replace the selective search method with some other region proposal method based on the following criteria
  - < 2000 region proposals
  - As fast as SS or better
  - As accurate as SS or better
  - Should be able to propose overlapping ROIs with different aspect ratios and scales



Faster RCNN: Region Proposal Aspect Ratios

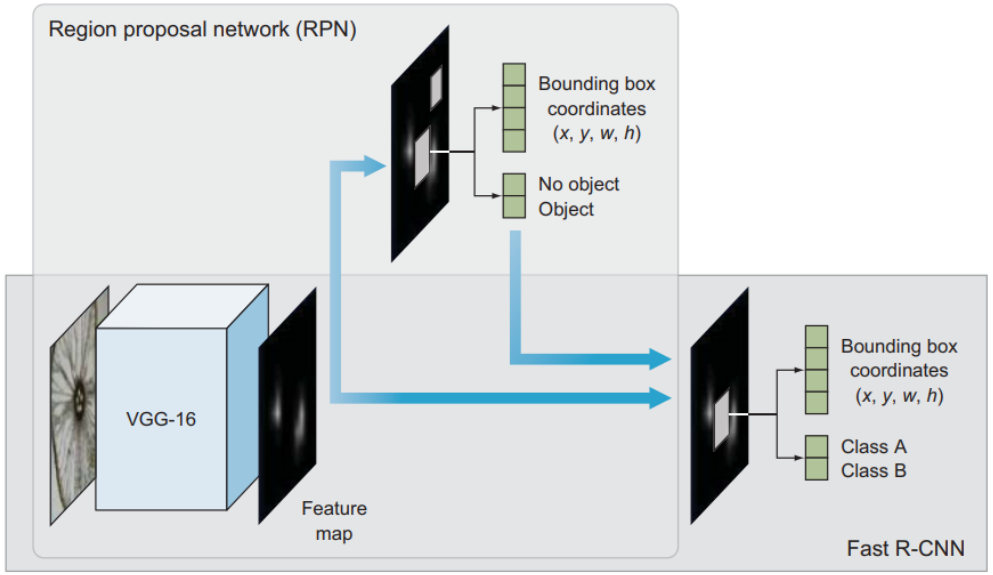
- To detect most kinds of objects you typically use one of the 3 aspect ratios given below
  - Vertical rectangle
  - Square
  - Horizontal rectangle
- Most of the object will be obstructing the one behind and hence you need overlapping ROIs
- Objects maybe of different size therefore you need ROIs of different scale



Faster RCNN: Components

To address the constraint of selective search for region proposals we have the following addition in Faster RCNN

1. Region Proposal Network (RPN): Selective Search is replaced by a ConvNet that proposes RoIs from the last feature maps of the feature extractor that are considered for investigating objects of interest. RPN has two outputs
    1. Objectness Score: whether there is an object present (1) or absent(0)
    2. Box location
2. Fast RCNN: same components
    1. Base network for feature extraction
    2. RoI pooling layer
    3. Output layer that contains two fully connected layers: softmax classifier and bounding box regression



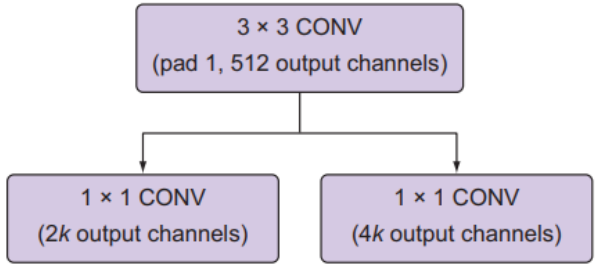
Region Proposal Network (RPN)

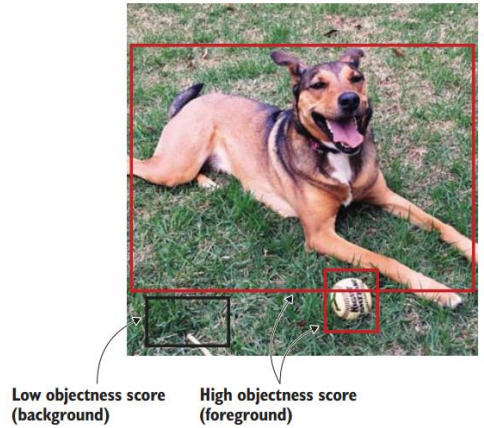
RPN identifies RoIs based on the last feature map of the pretrained CNN. Faster RCNN uses RPN to perform the region proposals instead of using selective search like its earlier versions

The architecture of RPN is composed of two layers

1. A 3x3 conv layer with 512 filters
2. Two parallel 1x1 conv layers
    1. Classification layer: Predicts whether the is an object present in the proposed region (or just random background
    2. Bounding box regression layer: bounding box of proposed RoIs

Note: Remember that RPN is not doing object detection but it is only finding objects of interest that the later components can both localize and classify





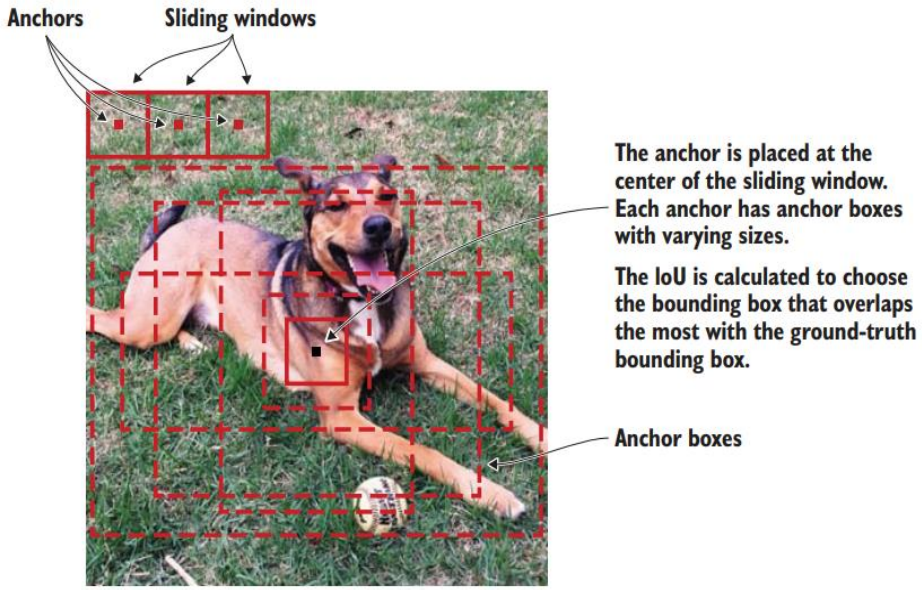
How are the region proposed bounding boxes predicted ?

Bounding box can be defined as (x,y,w,h) where x and y are the centers of the bounding box and w and h are the widths and heights respectively

We perform relative bounding box regression i.e. we create reference boxes called as anchor boxes in the image and make the regression layer predict the offsets from these boxes called deltas (Δx, Δy, Δw, Δh) to adjust the anchor boxes to better fit the object to get final proposals

Anchor Boxes

- RPN generates k RoIs for each location in the feature map using a sliding window approach. These regions are represented using anchor boxes.
- The anchors are centered in the middle of their corresponding sliding window and are of different aspect ratios and scales to cover a wide variety of different sized objects
- They are fixed bounding boxes that are placed throughout the image to be used for reference when first predicting the region proposals for objects of interest
- In the Faster RCNN paper, anchor boxes with 3 different aspect ratios and 3 different scales were proposed



Training the RPN: Need to ensure there are less than 2000 RoIS

RPN is used to classify whether an object is present or not and propose the bounding box for an object

How does one prepare the data for whether there is an object present or not inside the anchor box?

For each anchor the overlap value is computed which indicates how much these anchor boxes overlap with the ground truth boxes (or how useful these anchor boxes are going to be to help us find the region proposals)

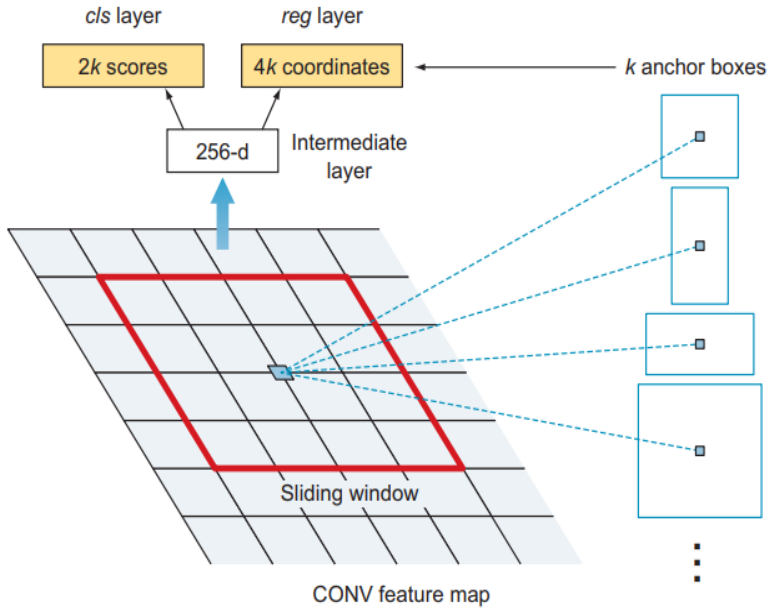
Only the following anchor inputs are taken forward for the objectness classification

1. Anchor boxes having high IoU with ground truth boxes IoU > 0.7 (positive anchor boxes)
2. Anchor boxes having low IoU with ground truth boxes IoU < 0.3 (negative anchor boxes)
3. Rest of the anchor boxes are ignored 0.3< IoU <0.7

Training the RPN Contd.

During the training process the positive and negative anchors are passed as input to two fully connected layers corresponding to the classification of anchors as containing an object or no object, and to the regression of location parameters (four coordinates)

Corresponding to the k number of anchors from a location, the RPN network outputs 2k scores and 4k coordinates. Thus, for example, if the number of anchors per sliding window (k) is 9, then the RPN outputs 18 objectness scores and 36 location coordinates



Training the Fast RCNN component of Faster RCNN

The final fully connected layers of the network take two inputs:

1. Feature maps from the feature extractor
2. RoIs coming from the RPN

The RoIs are projected on to the feature maps in the same way like Fast RCNN

Softmax Classifier and Bounding Box regressor are used to predict the object class and bounding box offsets respectively for the detected object

RPN: Loss Function

$$\mathscr{L}(p_i, t_i) = \frac{1}{N_{cls}} \sum_i \mathscr{L}_{cls}(p_i, p_i^*) + \frac{\lambda}{N_{reg}} \sum_i p_i^* \mathscr{L}_{reg}(t_i, t_i^*)$$

$p_i^* = 1$ if anchor box contains ground truth object

$= 0$ otherwise

$p_i =$ predicted probability of anchor box containing an object

$N_{cls} =$ batch-size

$N_{reg} =$ batch-size $\times k$

$k =$ anchor boxes

Faster RCNN Training in a Nutshell

1. Finetune RPN using a pretrained network
2. Finetune Fast RCNN using a pretrained network on the proposed images from step 1
3. Finetune RPN using the pretrained network used in step 2
4. Now finetune the Fast RCNN using the pretrained network used in step 3 on the images proposed in step 3

Review of RCNN Family

1. mAP on Pascal VOC 2007
    1. RCNN: 66%
    2. Fast RCNN: 66.9%
    3. Faster RCNN: 66.9%
2. Limitations
    1. RCNN: 3 stage training, high computation time and memory requirements
    2. Fast RCNN: Selective search is slow and hence computation time is still high
    3. Faster RCNN: It is still a 2 stage network
3. Test Time Per Image
    1. RCNN: 50 seconds
    2. Fast RCNN: 2 seconds
    3. Faster RCNN: 0.2 seconds
4. Speedup from RCNN
    1. Fast RCNN: 25x
    2. Faster RCNN: 250x

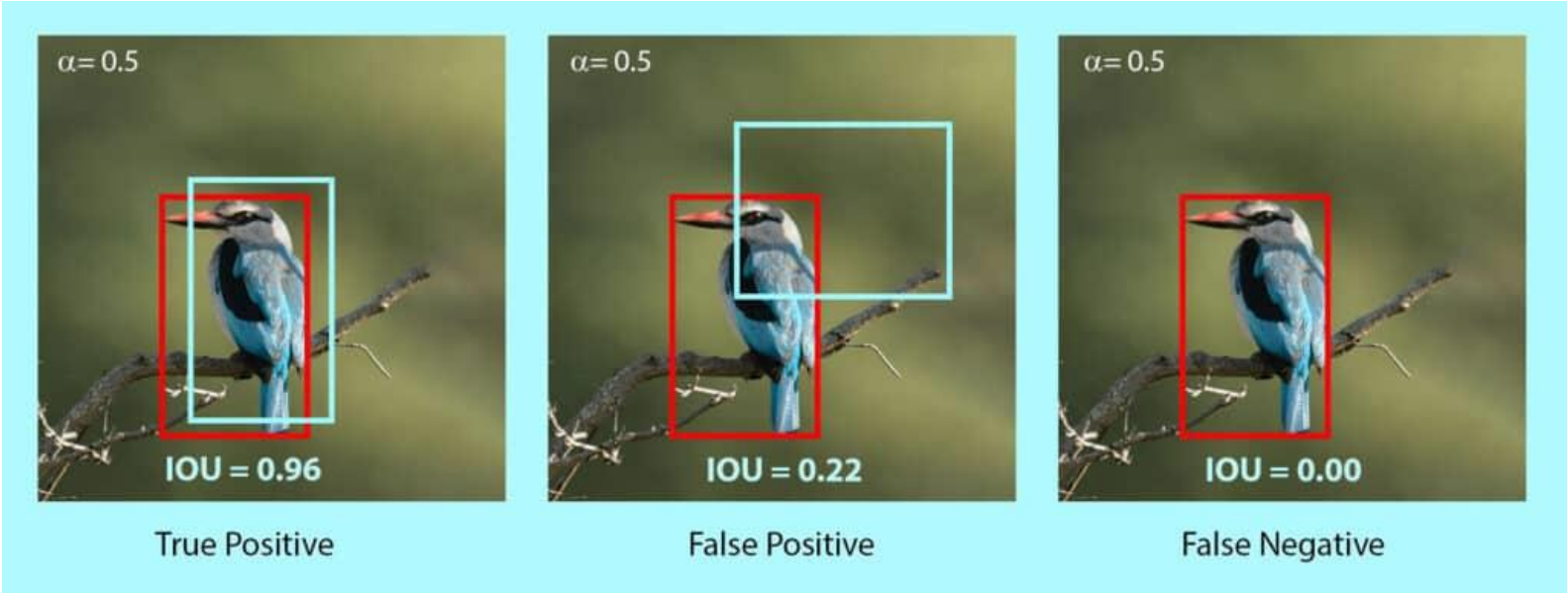
mAP: mean Average Precision

Mean Average Precision (mAP) is a performance metric popularly used for object detection models. It is the most popular metric that is used by benchmark challenges for object detection such as PASCAL VOC, COCO, ImageNET challenge, Google Open Image Challenge, etc.

Is it just average of precisions ? Its not that simple.

mAP: mean Average Precision

Remember IoU ?



mAP: mean Average Precision

Remember Precision ?

Precision measures the proportion of predicted positives that are actually correct. If you are wondering how to calculate precision, it is simply the **True Positives** out of **total detections**. Mathematically, it's defined as follows.

P = TP/(TP + FP)

= TP / Total True Predicted

The value ranges from 0 to 1.

mAP: mean Average Precision

Remember Recall ?

Recall measures the proportion of actual positives that were predicted correctly. It is the True Positives out of all **Ground Truths**. Mathematically, it is defined as follows.

R = TP / (TP + FN)

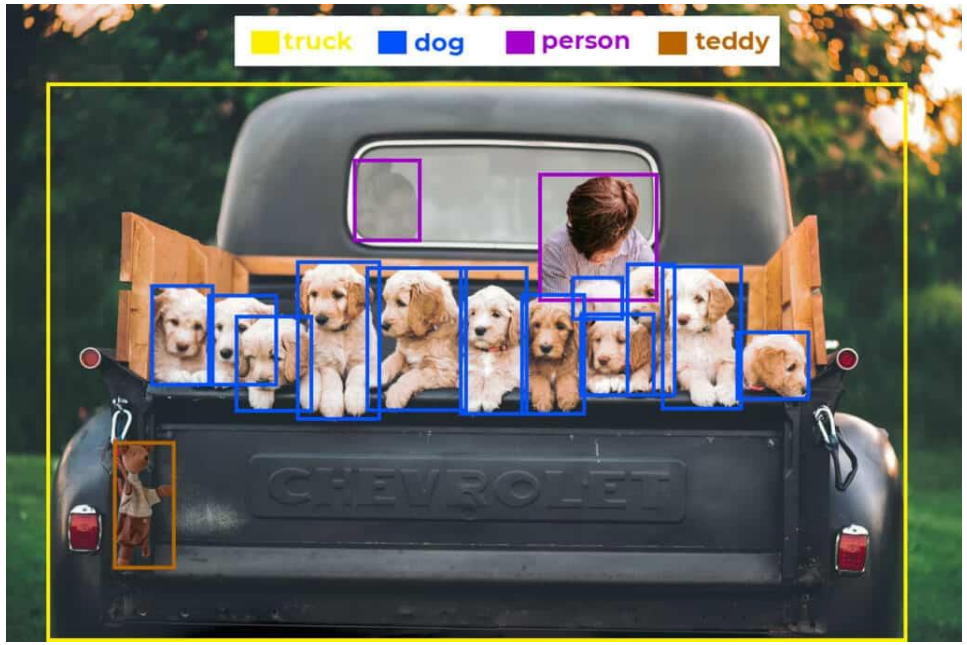
= TP / Total Ground Truths

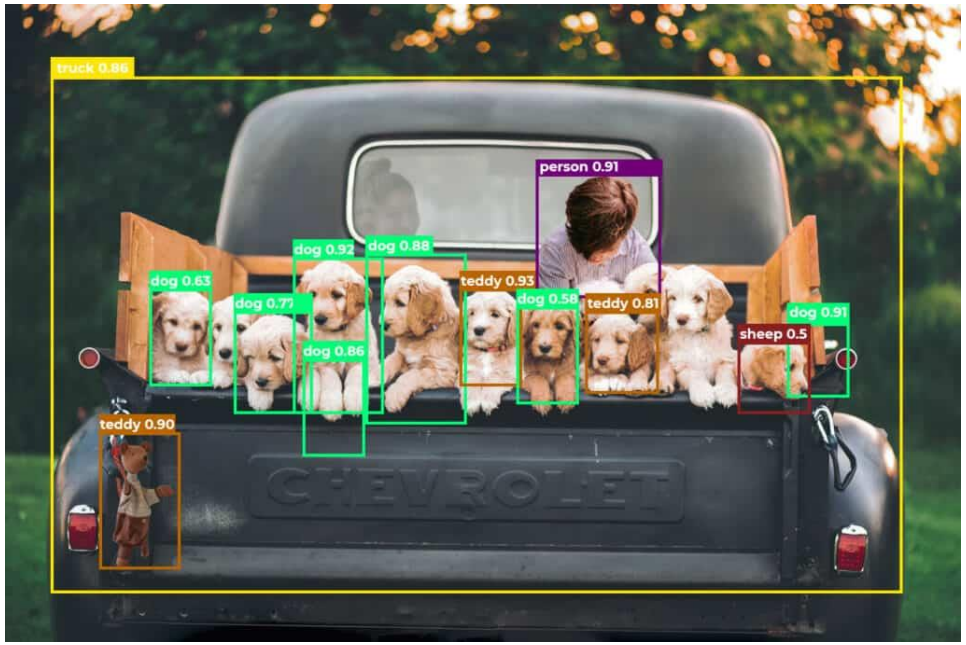
Similar to Precision, the value of **Recall** also ranges from 0 to 1.

mAP: mean Average Precision

mAP: mean Average Precision

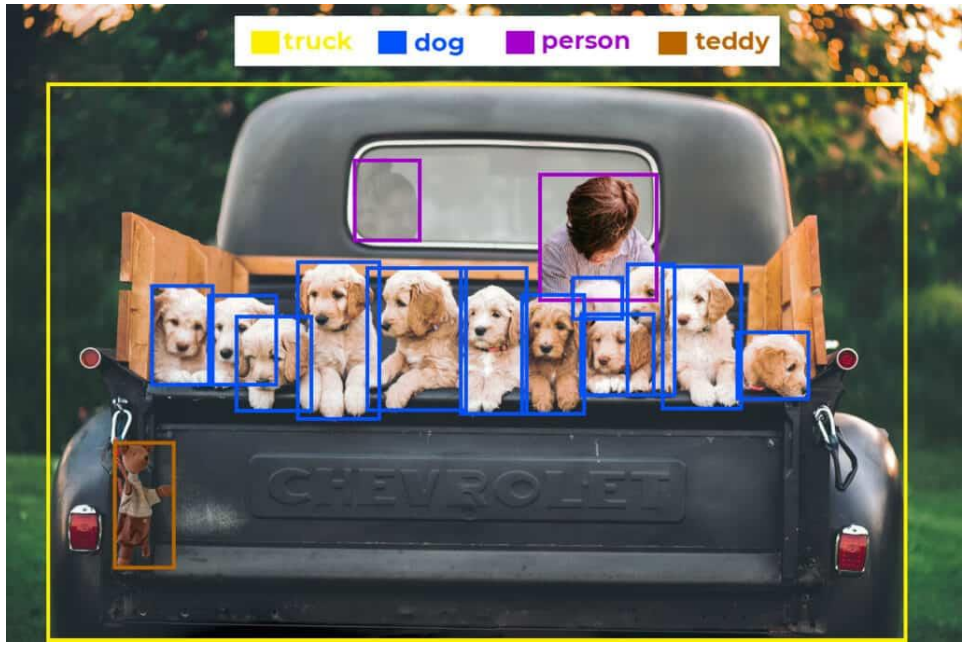
Average Precision (AP) is **actually not** the average of **Precision (P)**. The term AP has evolved with time. For simplicity, we can say that **it is the area under the precision-recall curve**.

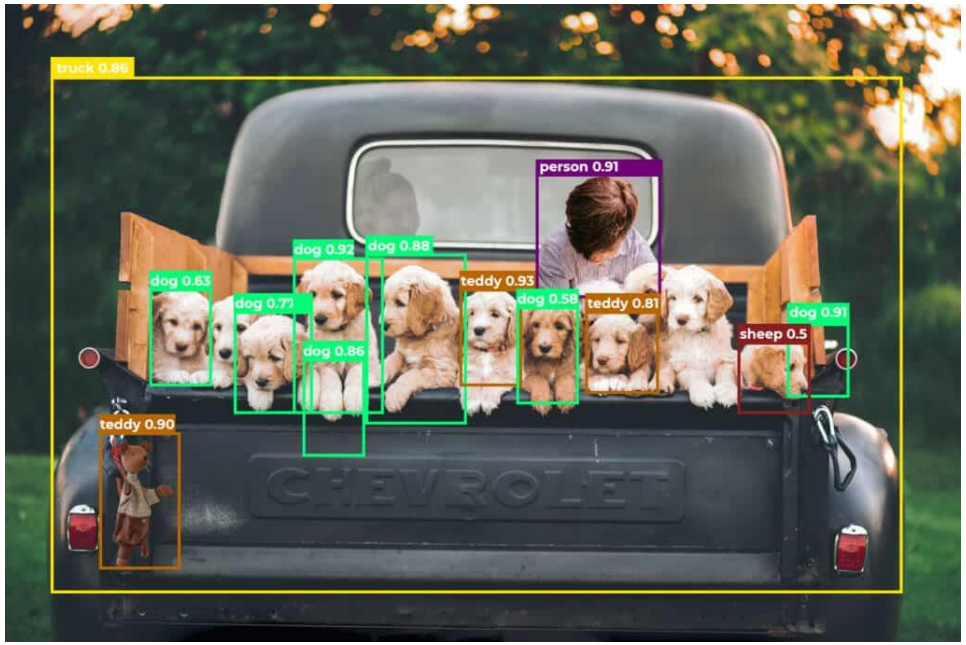




mAP: mean Average Precision

There are 12 dogs, 2 persons, 1 teddy and 1 truck. The predictions are 7 dogs, 1 person, 3 teddies and 1 truck.





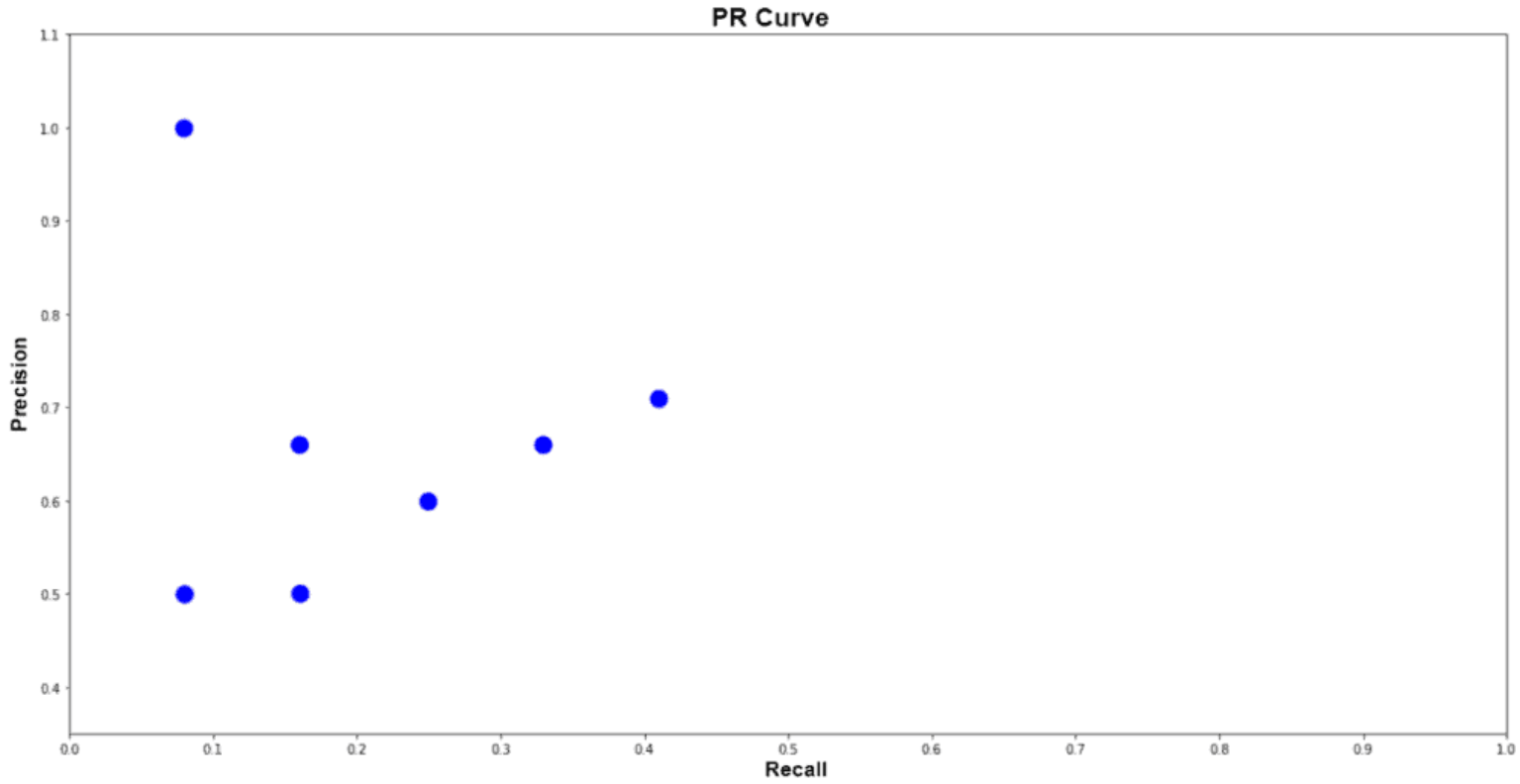
AP: for class Dog

Detections							
Conf.	0.63	0.77	0.92	0.86	0.88	0.58	0.91
Matches GT by IoU?	TP	TP	TP	FP	TP	TP	FP

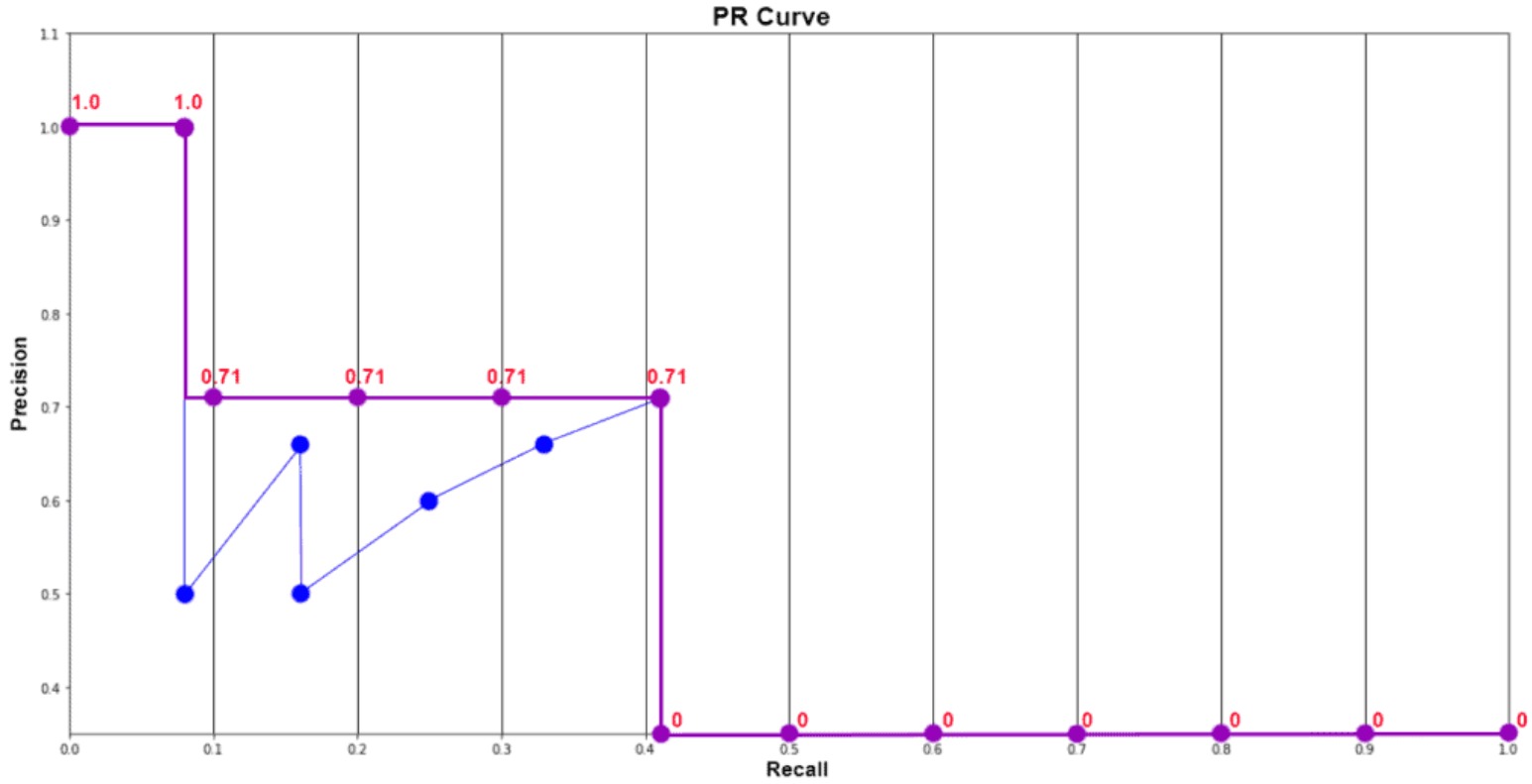
mAP: mean Average Precision

** space for lethal math **

mAP: PR curve


PR Curve
Precision
Recall
1.1
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0

mAP: PR curve


PR Curve
Precision
Recall
1.1
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0
1.0
1.0
0.71
0.71
0.71
0.71
0
0
0
0
0
0
0

mAP: mean Average Precision

If we calculated just AP for dog ..... does that mean mAP is nothing but the mean of average precision calculated for each class ? Yes

As the name suggests, **Mean Average Precision or mAP** is the average of **AP** over all detected classes.

mAP = 1/n * sum(AP), where n is the number of classes.

In the example above, we have 5 classes. Therefore, the calculated mAP is;

mAP = 1/5 * (0.349 + 0.545 + 0 + 1 + 0.5) (example APs)

= 0.4788

= 47.88 %

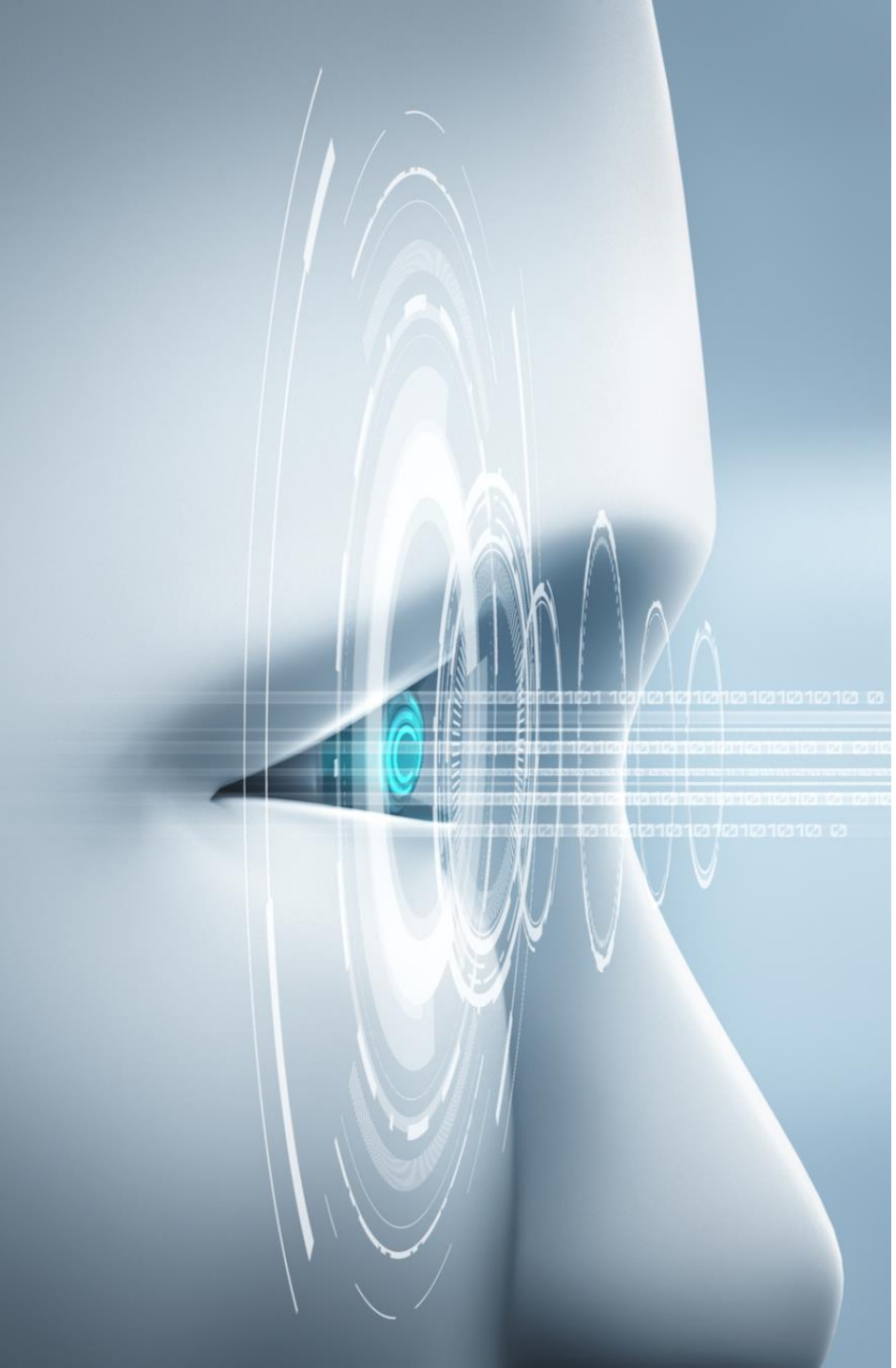
What is mAP@0.50 or mAP@0.50:0.95 ?

We decided the TPs and FPs wrt the IoU values threshold at 0.5. i.e. if IoU > 0.5 then it's a TP else it's a FP. Therefor the mAP calculation done for IoU thresh = 0.5 is known as mAP@0.50

Similarly MS COCO defines Average Precision (AP) as mAP@[0.5:.05:.95] i.e. an average of mAPs at 0.50, 0.55,0.60…..0.95

Thank you

For any queries drop an email at: quadeershaikh15.8@gmail.com



Computer Vision Applications

BY QUADEER SHAIKH

About me

Work Experience

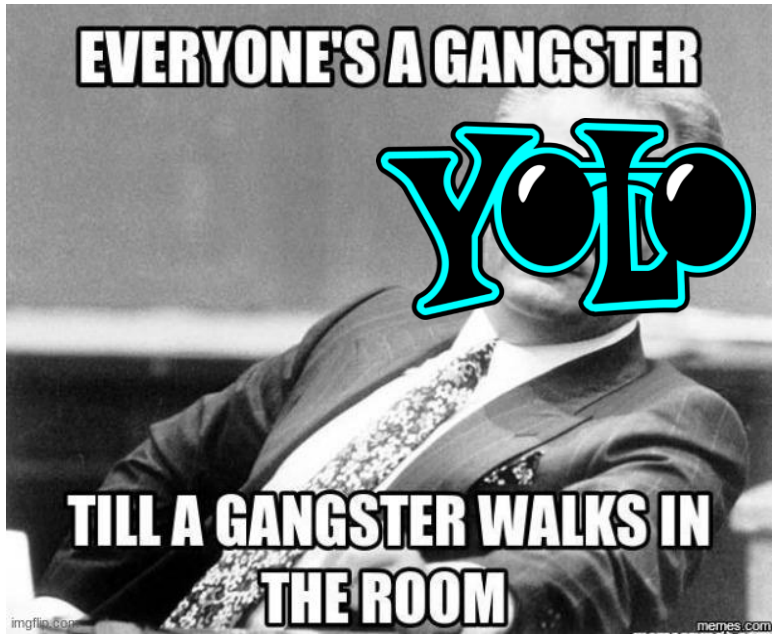
- Risk Analyst
  - Morgan Stanley (Jan 2023 – Present)
- Data Science Intern
  - AkzoNobel Coatings International B.V. Netherlands (Feb 2022 – Dec 2022)
- Data Science Intern
  - EzeRx Health Tech Pvt. Ltd. (Jan 2022 – July 2022)
- Associate Engineer
  - Tata Communications Ltd. (July 2019 – Aug 2020)
- Network Automation and Analysis Engineer Intern
  - Cisco (June 2018 – July 2018)

Education

- M.Tech – Artificial Intelligence
  - NMIMS (2021 - 2023, currently pursuing)
- B.E. – Computer Engineering
  - Mumbai University (2015 - 2019)

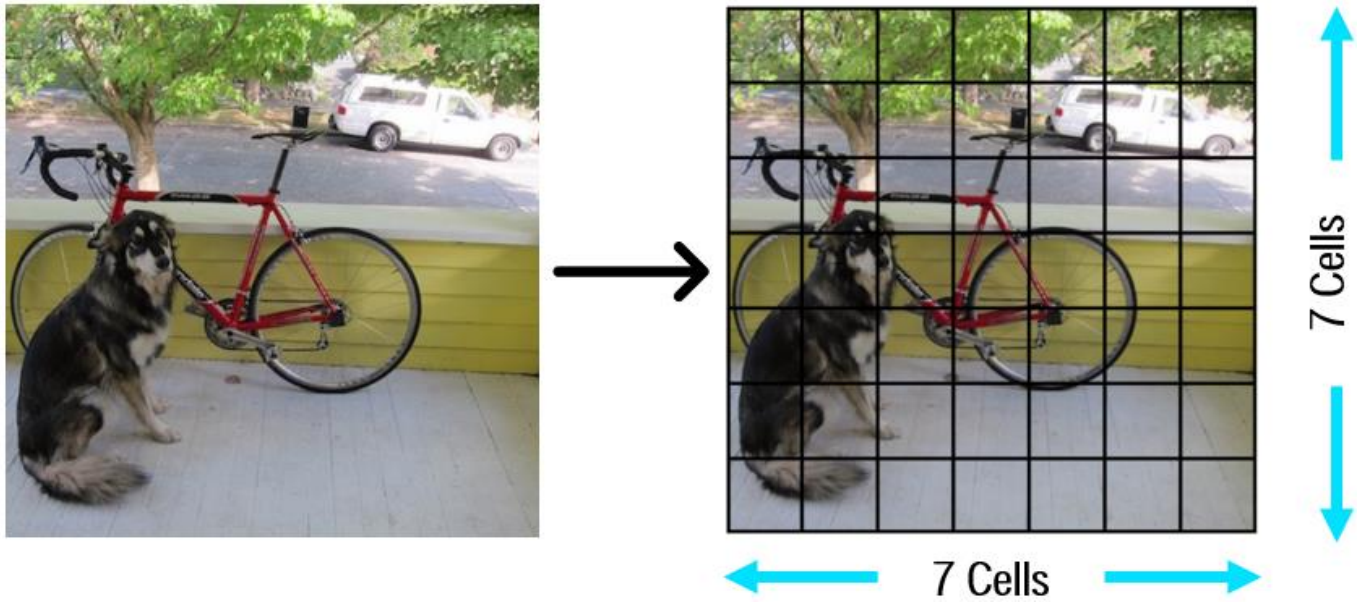
YOLOv1: You Only Look Once

- The network only looks at the image once to detect multiple objects. Thus, it is called YOLO, You Only Look Once.
- Unlike the two stage detectors like the RCNN family YOLO is a on stage detector and hence the name YOLO

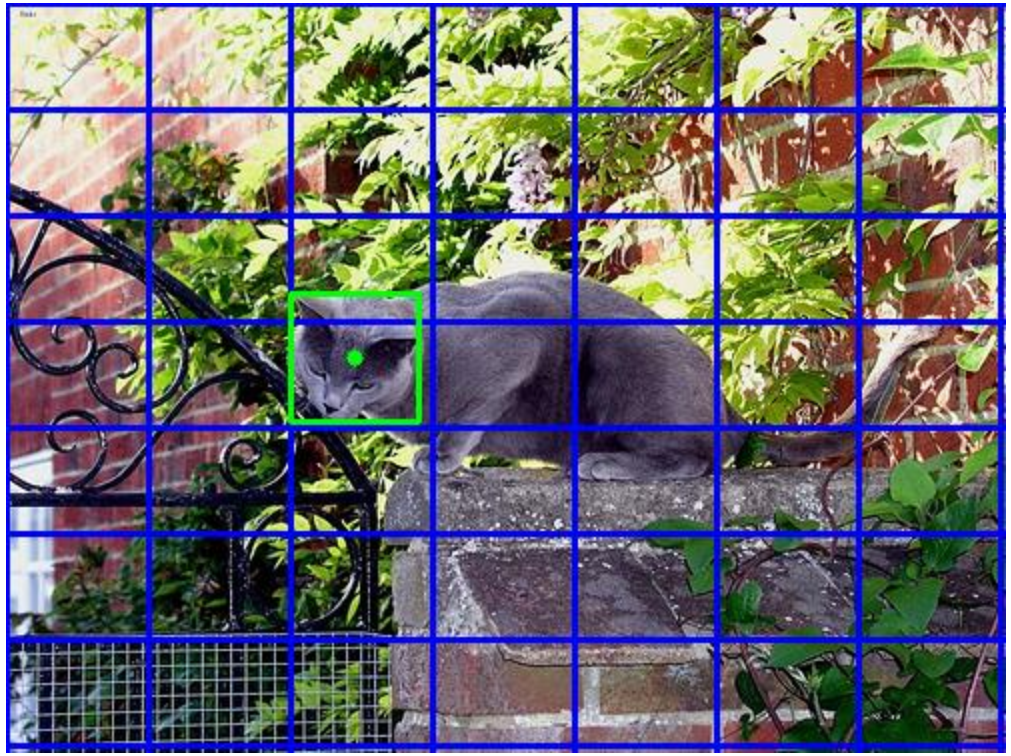


YOLOv1: Unified Detection/Detection at one go

- **The input image is divided into an S×S grid (S=7)**. If the center of an object falls into a grid cell, that grid cell is responsible for detecting that object.

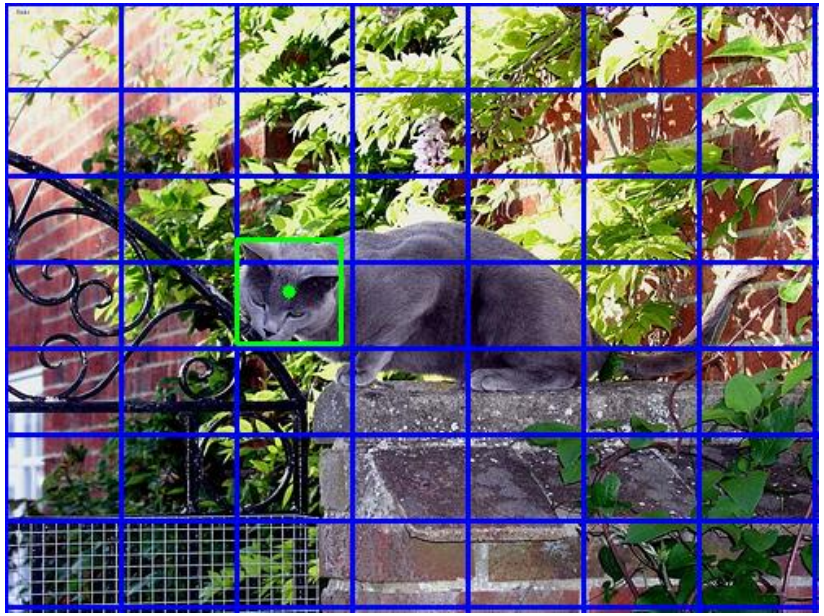


YOLOv1: Unified Detection/Detection at one go



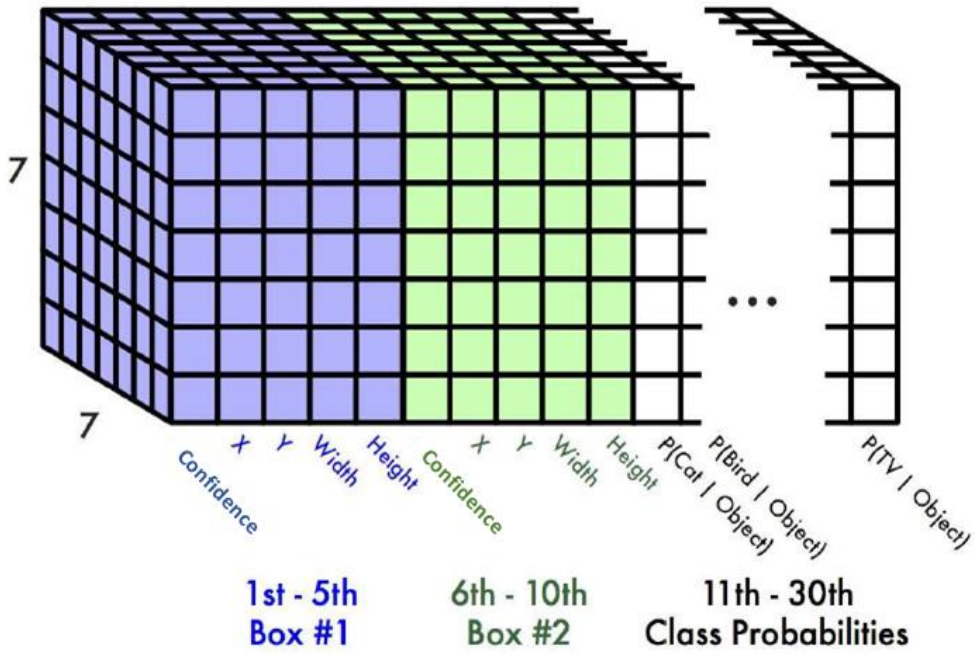
YOLOv1: Unified Detection/Detection at one go

1. Each grid cell predicts B bounding boxes (B=2 in the paper) and confidence scores for those boxes.
2. These confidence scores reflect how confident the model is that the box contains an object. i.e. the probability of prediction of the box
3. Each bounding box consists of 5 predictions: x, y, w, h, and confidence.
   - The (x, y) coordinates represent the center of the box relative to the bounds of the grid cell.
   - The width w and height h are predicted relative to the whole image.
   - The confidence represents the Intersection Over Union (IOU) between the predicted box and any ground truth box.



YOLOv1: Unified Detection/Detection at one go

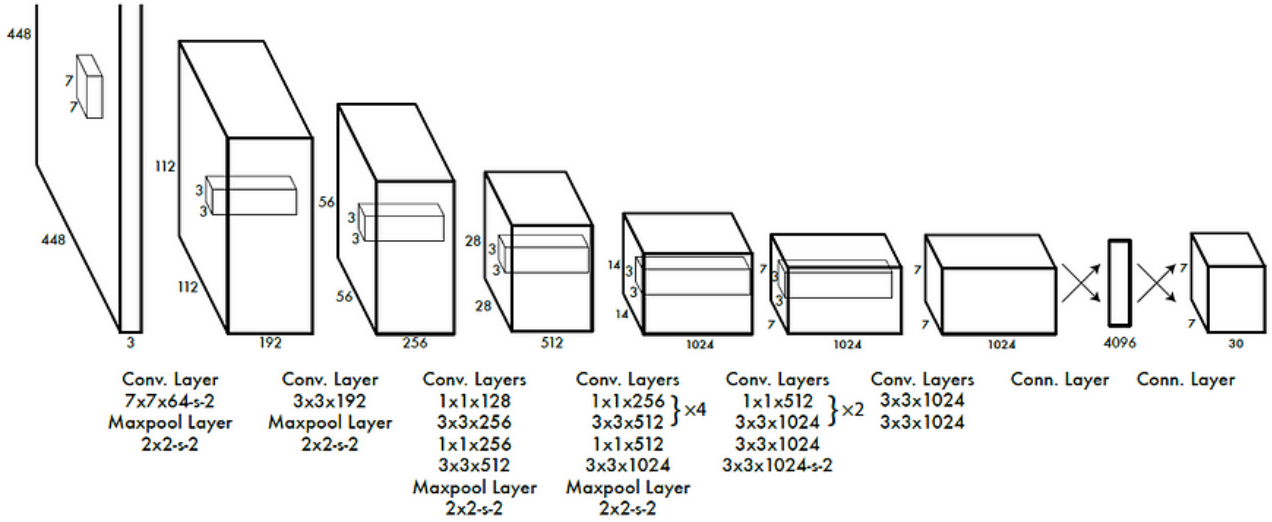
1. Out of the B bounding boxes for each grid cell only one is selected based on the highest confidence.
2. Each grid cell also predicts C conditional class probabilities, where C is the number of classes. YOLOv1 was originally trained on PascalVOC2007 dataset which had 20 classes so in that case C=20
3. S = 7, B = 2, C = 20
4. S x S x (B x 5 + C) = 7x7x(2x5+20) = 7x7x30 is the output shape of the network
5. The ground truth data for object detection consisting of bounding box coordinates and class predictions should be structured in the shape of the above output



YOLOv1 Architecture

The model consists of 24 convolutional layers followed by 2 fully connected layers. Alternating 1×1 convolutional layers reduce the features space from preceding layers.

Input Size: 448x448x3, **Output size:** 7x7x30 (PascalVOC2007)



YOLOv1 Loss Function

Surprisingly YOLO's loss function is the most odd one to have ever been defined as it comprises of squared errors for both box coordinates and class predictions.

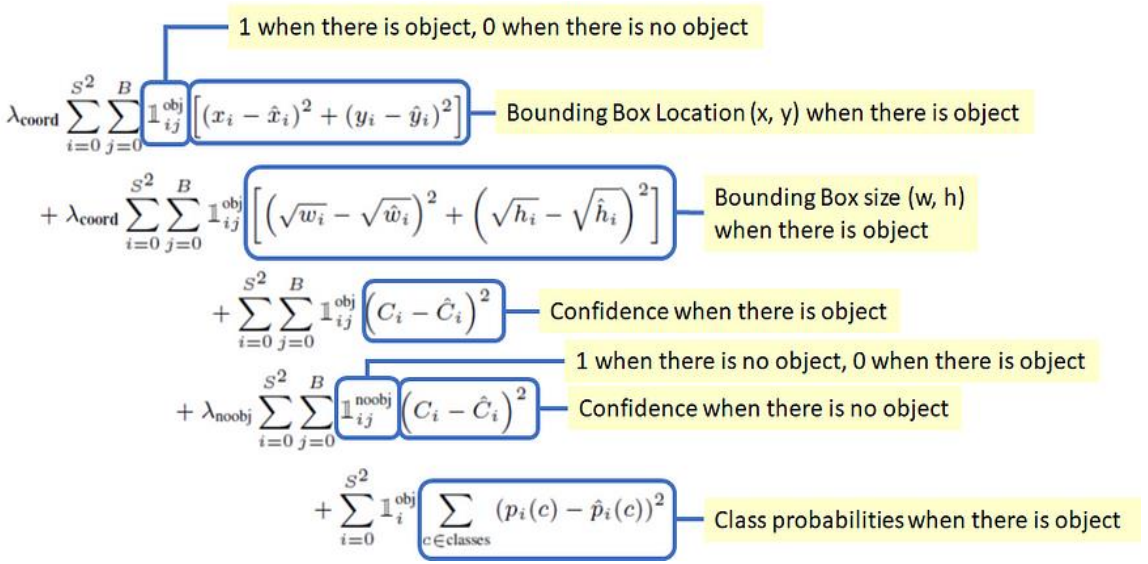
1. The bounding box x and y coordinates is parametrized to be offsets of a particular grid cell location so they are also bounded between 0 and 1. And the sum of square error (SSE) is estimated only when there is object.

2. The bounding box width and height are normalized by the image width and height so that they fall between 0 and 1. SSE is estimated only when there is object. Since small deviations in large boxes matter less than in small boxes. square root of the bounding box width w and height h instead of the width and height directly to partially address this problem.

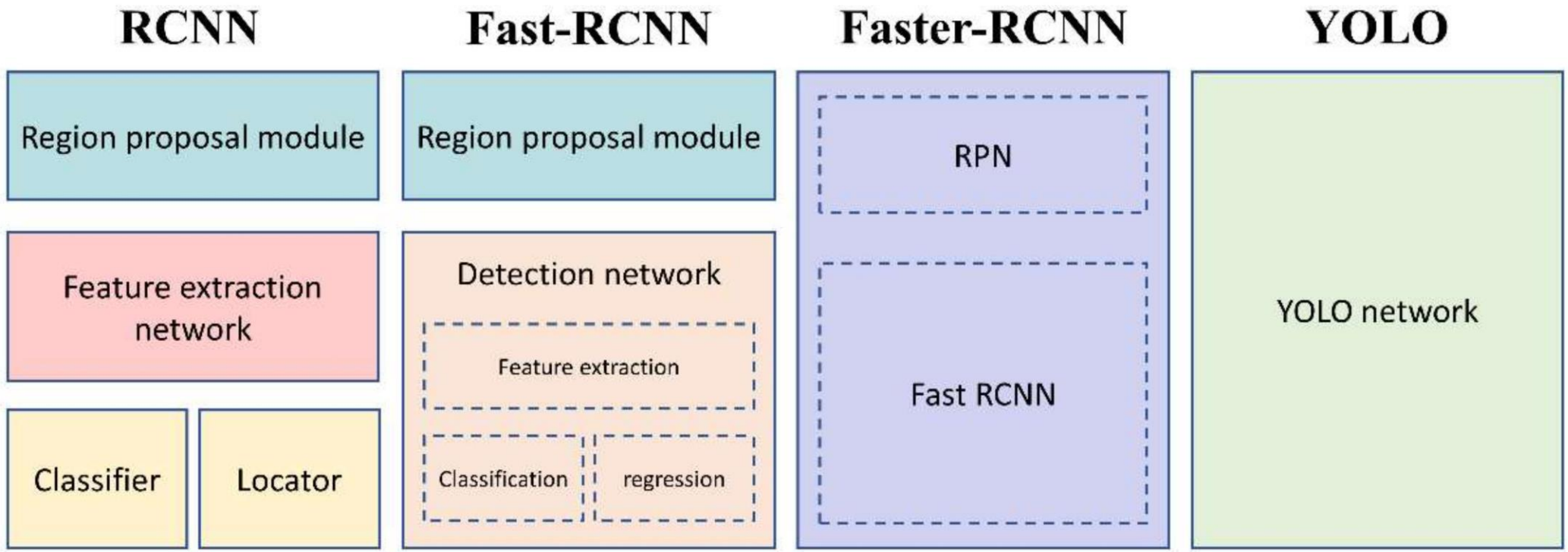
3. In every image many grid cells do not contain any object. This pushes the "confidence" scores of those cells towards zero, often overpowering the gradient from cells that do contain objects, and makes the model unstable. Thus, the loss from confidence predictions for boxes that don't contain objects, is decreased, i.e. **λnoobj**=0.5.

4. SSE of class probabilities when there is objects.

5. Due to the same reason mentioned in 3 and 4, λcoord = 5 to increase the loss from bounding box coordinate predictions.



RCNN Family vs YOLOv1



Thank you

For any queries drop an email at: quadeershaikh15.8@gmail.com