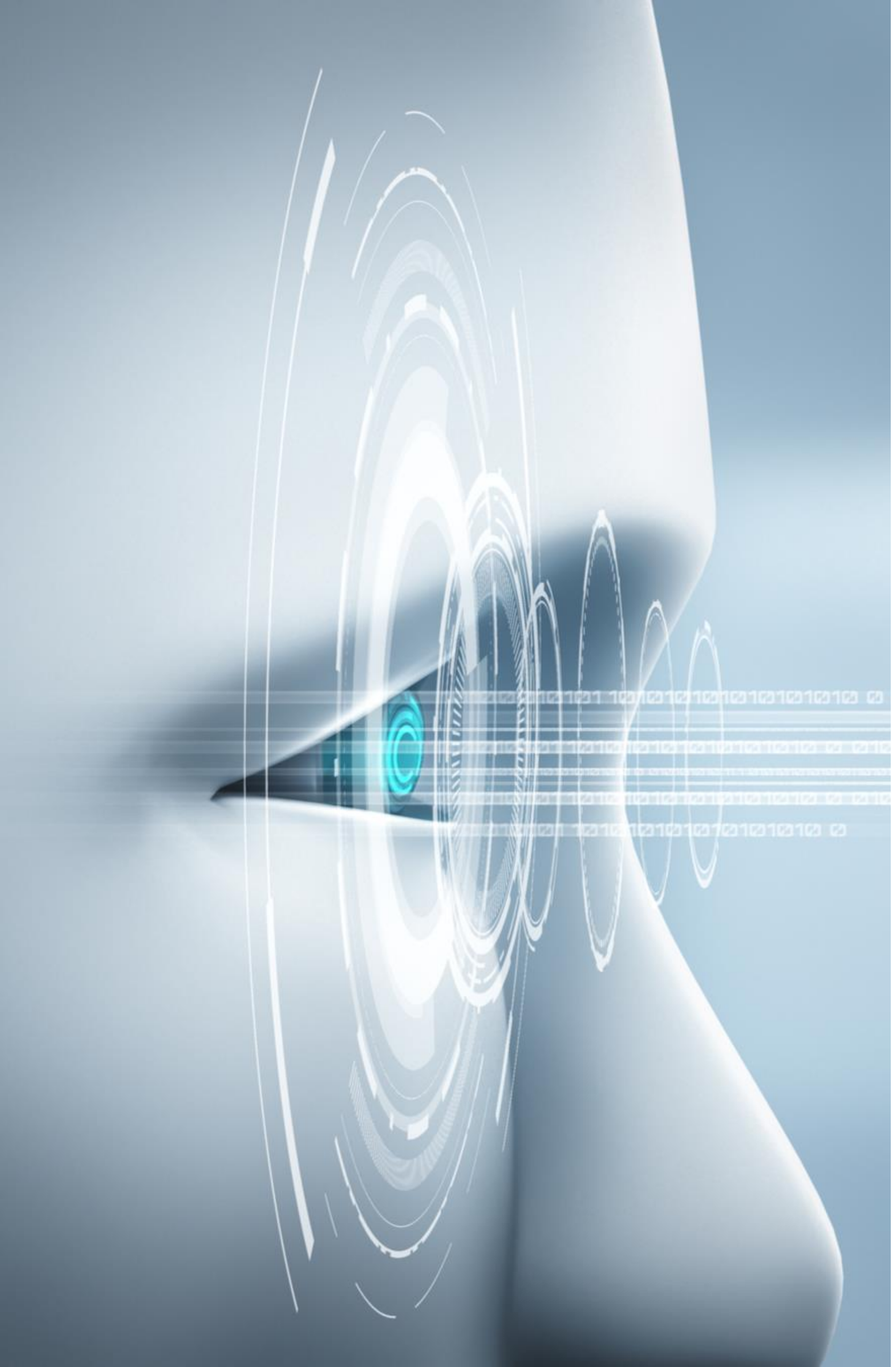


Computer Vision Applications

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)

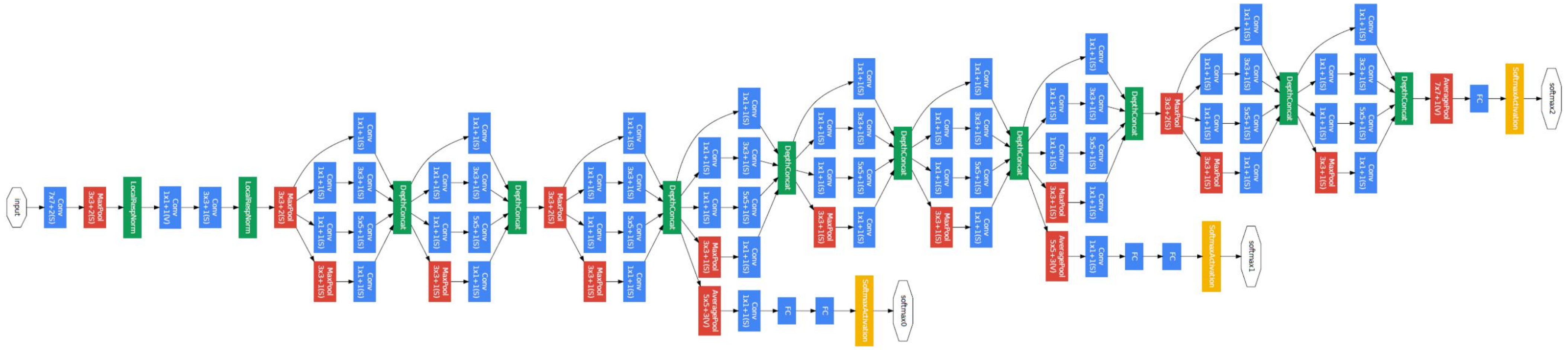
Education
- M.Tech – Artificial Intelligence
  - NMIMS (2021 - 2023, currently pursuing)
- B.E. – Computer Engineering
  - Mumbai University (2015 - 2019)

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

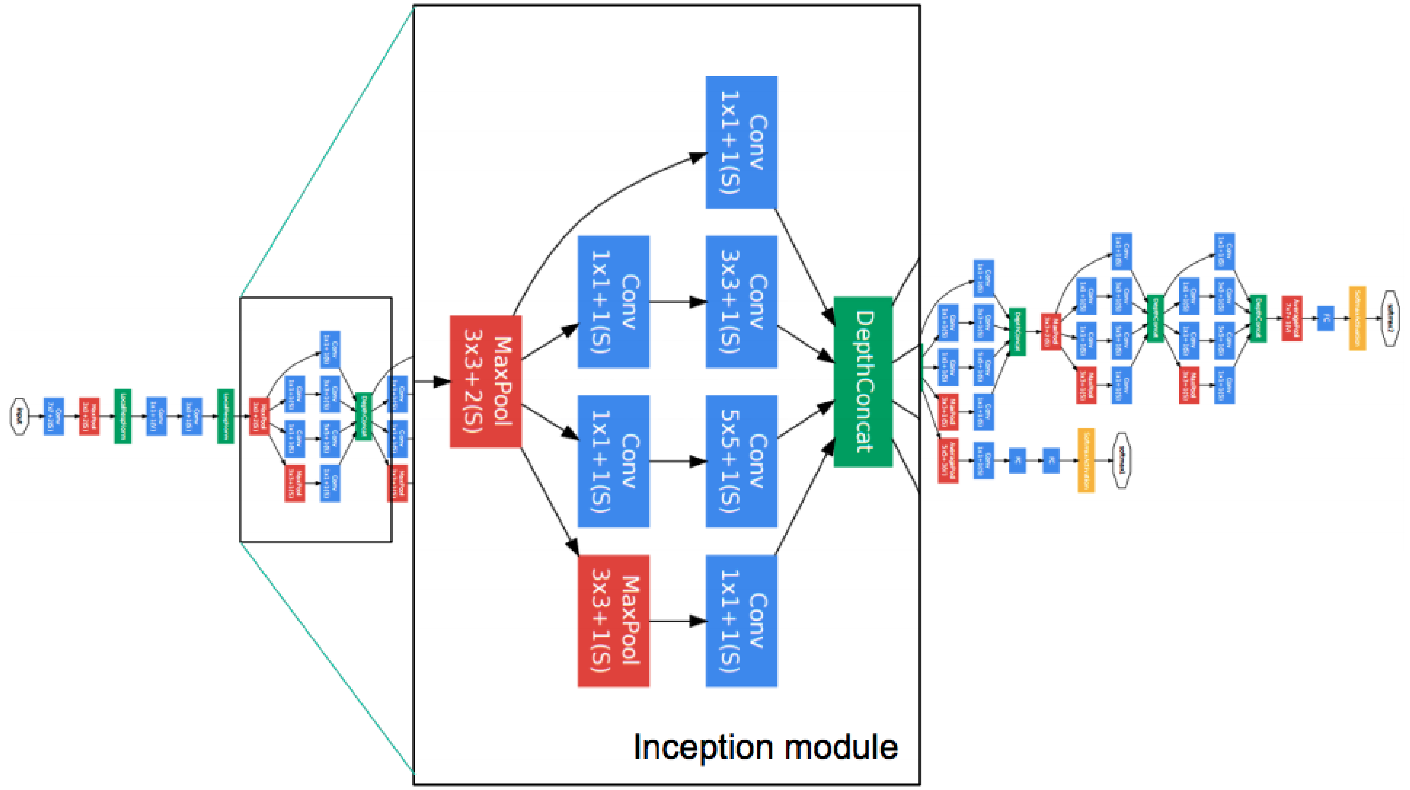
- GoogleNet/Inception V1 uses a network 22 layers deep (deeper than VGGNet) while reducing the number of parameters by 12 times (from ~138 million to ~13 million) and achieving significantly more accurate results.
- **The kernel size of the convolutional layer**—We’ve seen in previous architectures that 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 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.

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**

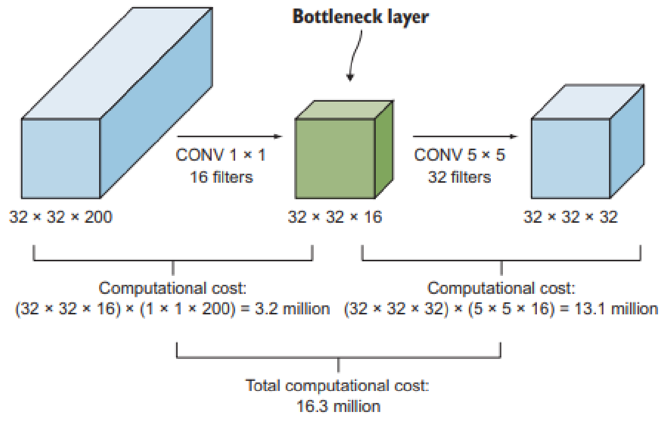
While Google thought that this computation can be performed with modern computers this computation is still expensive.

Dimensionality Reduction Layer (1x1 Conv Layer)

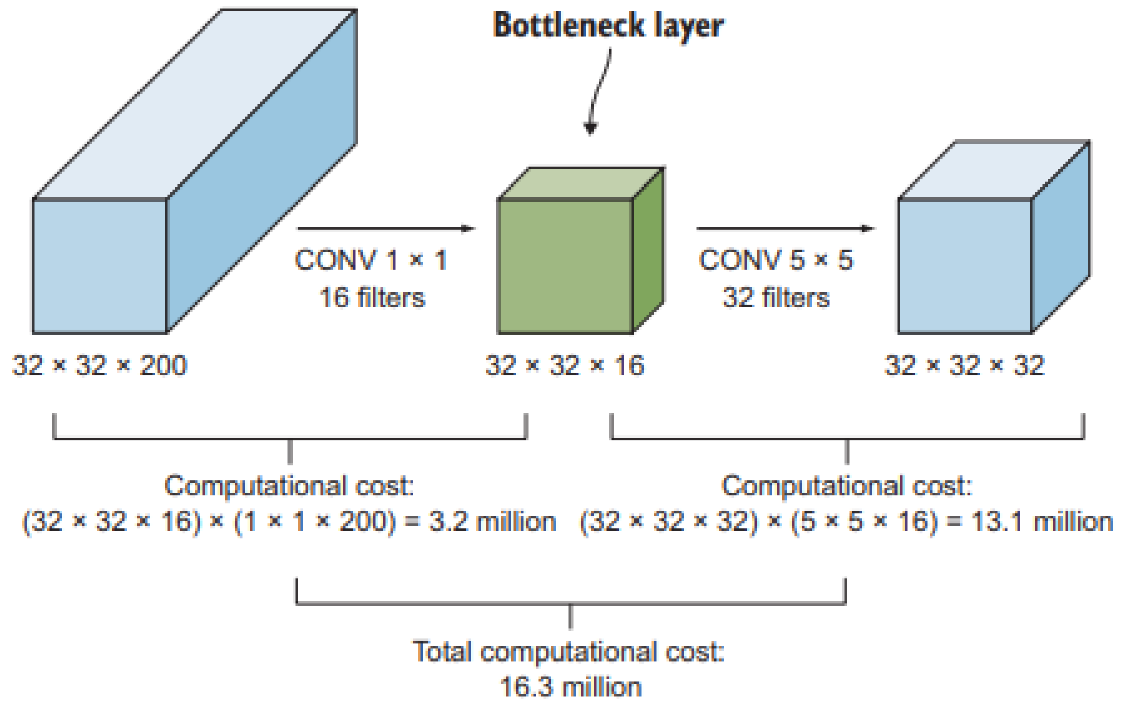
The 1 × 1 convolutional layer can reduce the operational cost of 163 million operations to about a tenth of that. That is why it is called a reduce layer/bottleneck layer.

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)

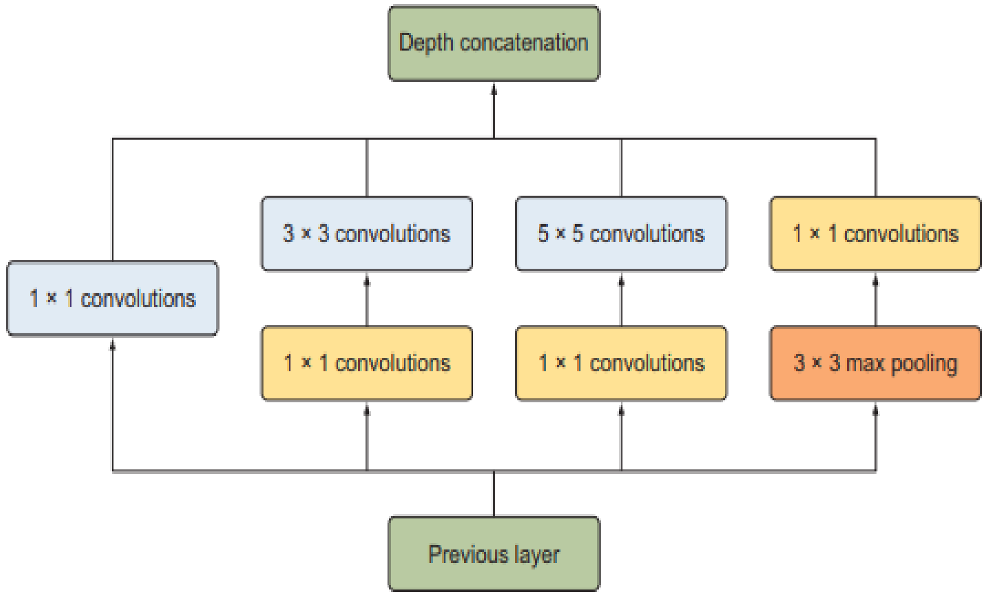


Inception Module with Dimensionality Reduction

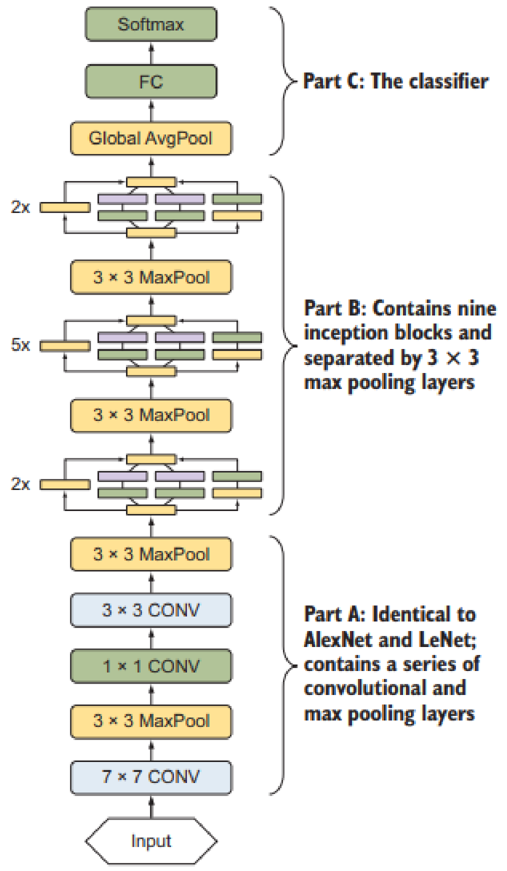
Does shrinking the depth representation size so dramatically hurt the performance of neural network ?

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

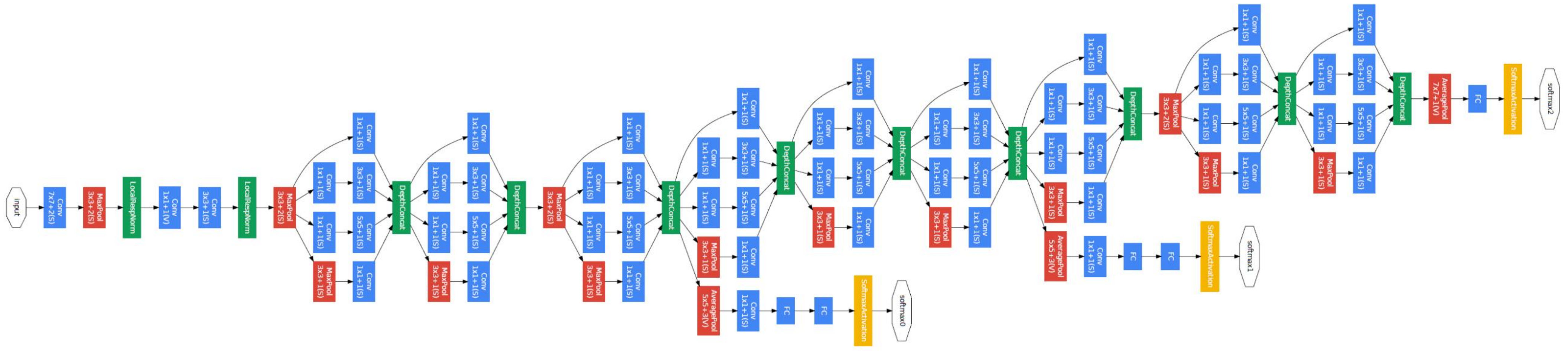


GoogleNet/Inception V1


Softmax
FC
Global AvgPool
Part C: The classifier
2x
3 × 3 MaxPool
5x
3 × 3 MaxPool
2x
Part B: Contains nine inception blocks and separated by 3 × 3 max pooling layers
3 × 3 MaxPool
3 × 3 CONV
1 × 1 CONV
3 × 3 MaxPool
7 × 7 CONV
Input
Part A: Identical to AlexNet and LeNet; contains a series of convolutional and max pooling layers

GoogleNet/Inception V1

“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.”



GoogleNet/Inception V1 Summary

Following were the developments

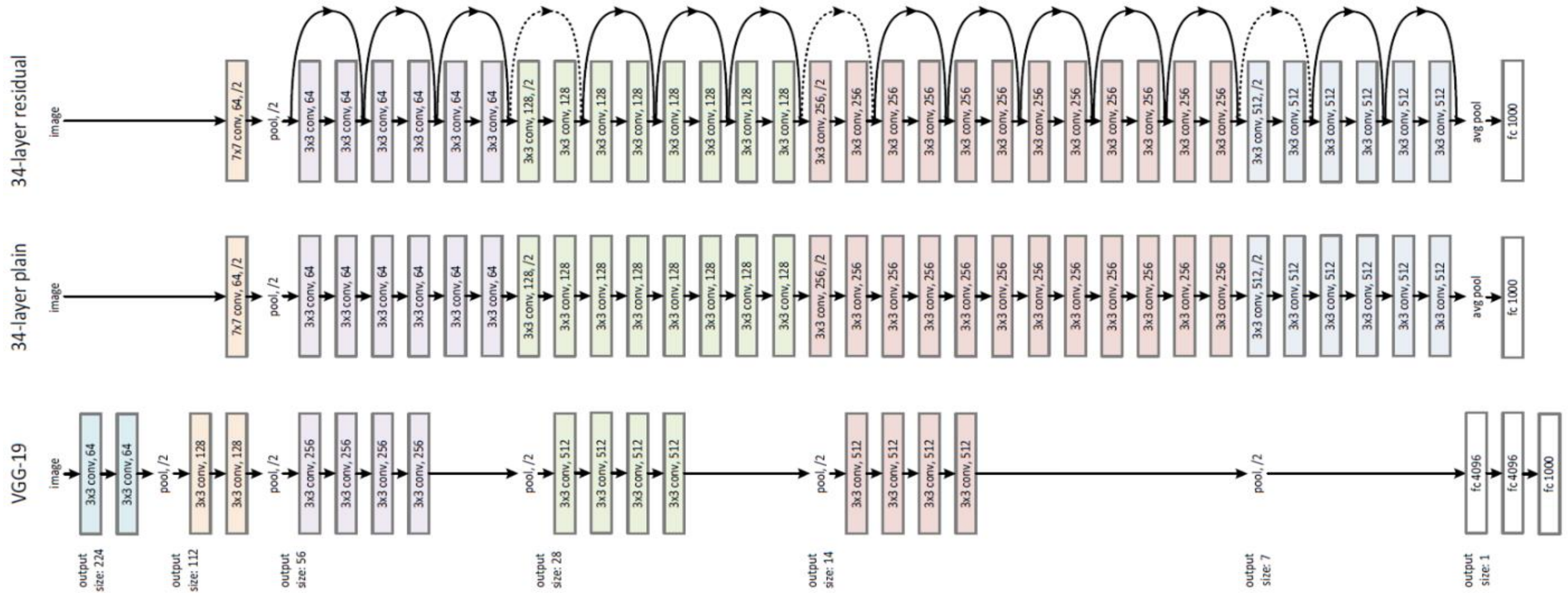
1. Inception Block contains
   1. 1 × 1 convolutional layer
   2. 1 × 1 convolutional layer + 3 × 3 convolutional layer
   3. 1 × 1 convolutional layer + 5 × 5 convolutional layer
   4. 3 × 3 pooling layer + 1 × 1 convolutional layer
2. 1x1 conv layers are used for depth dimensionality reduction thus 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

Microsoft's thoughts on building CNNs

Google

Microsoft

ResNet


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
34-layer plain
VGG-19
3x3 conv, 256
3x3 conv, 512
fc 4096
output size: 224
output size: 112
output size: 56
output size: 28
output size: 14
output size: 7
output size: 1

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

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)

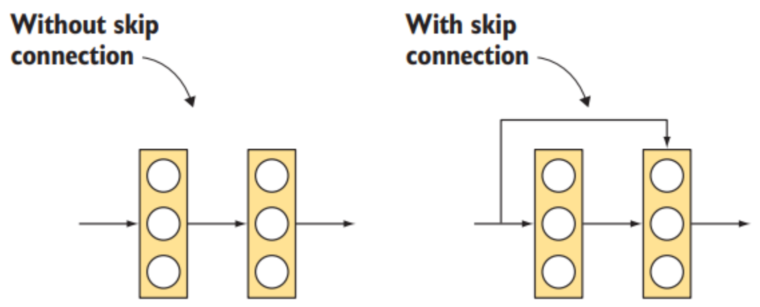
Microsoft's approach to Vanishing Gradient

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.

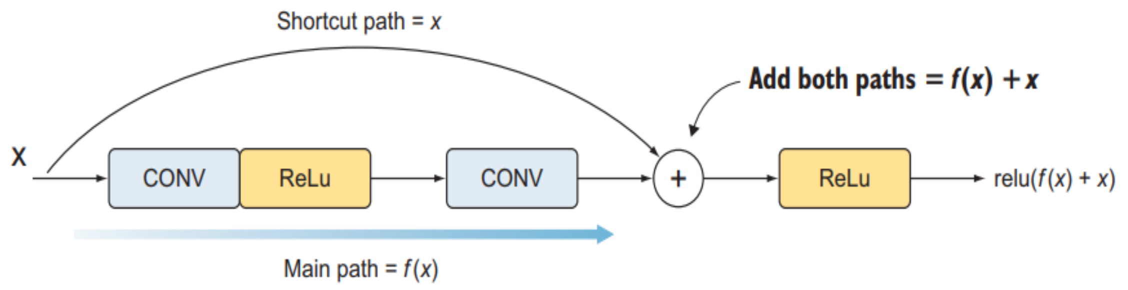
They are used to flow information from earlier layers in the network to later layers, creating an alternate shortcut path for the gradient to flow through.

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

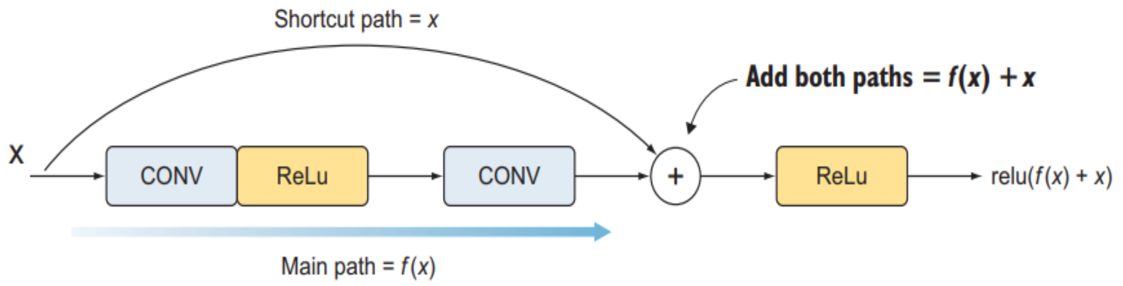
The shortcut arrow/skip connection is added at the end of the convolution layer

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

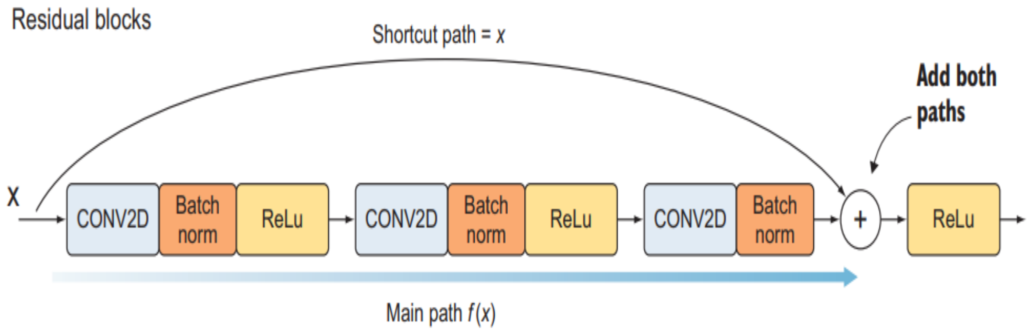


Residual Blocks

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.

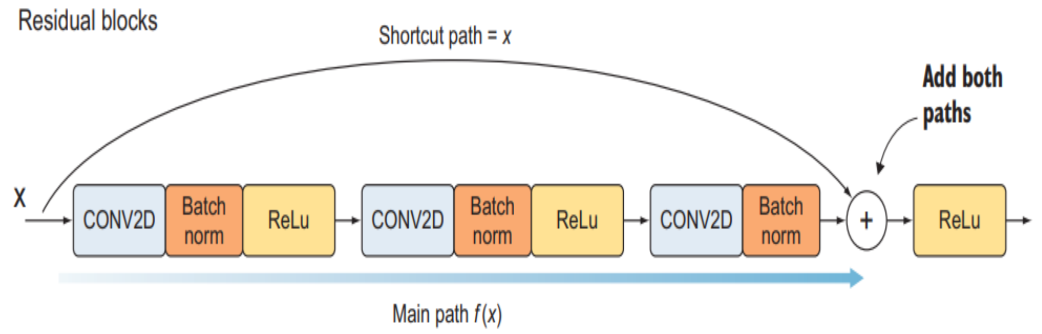
As mentioned earlier the shortcut path is added to the main path right before the activation function of the last convolution layer. Then we add ReLU function after adding the two paths.

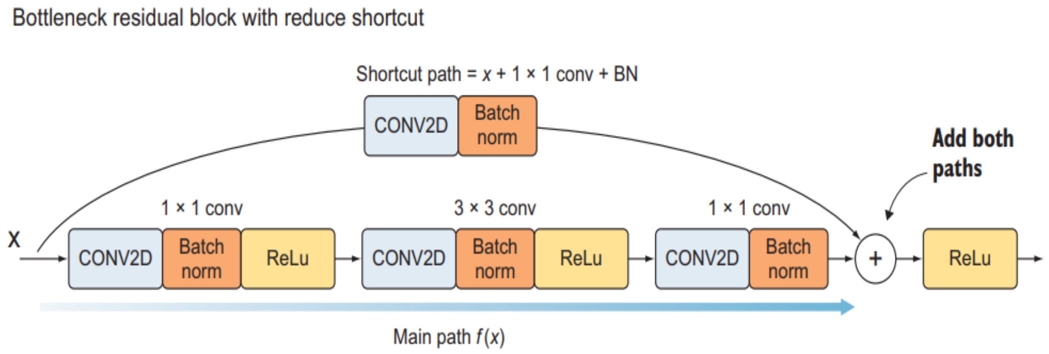


Residual Blocks

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
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.



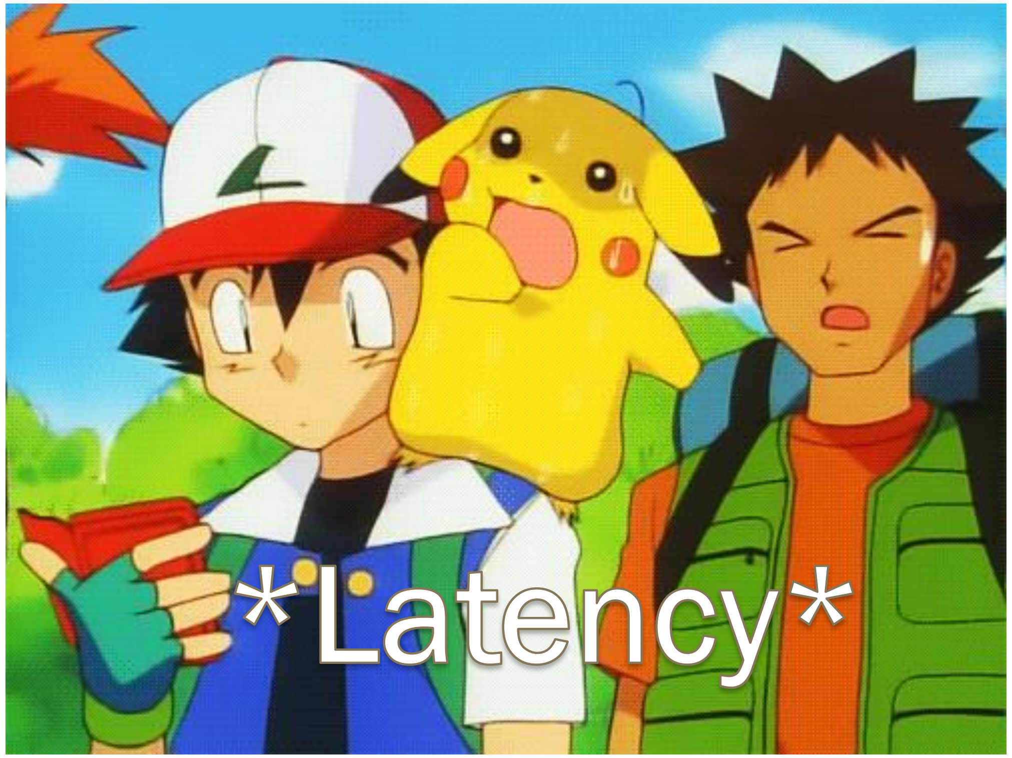


ResNet Summary

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.

Google's thoughts on Models running on mobile devices

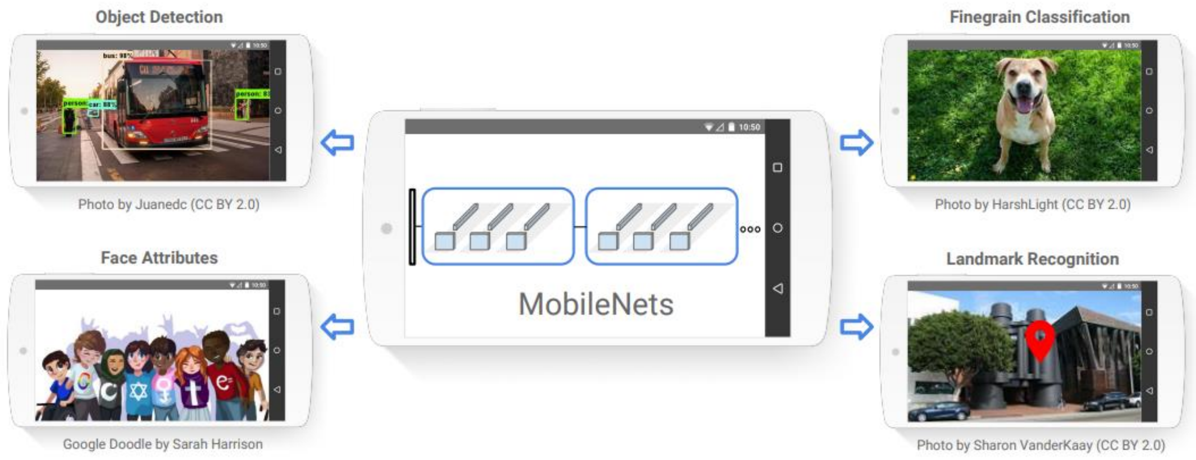


MobileNet: Google's approach at light weight models

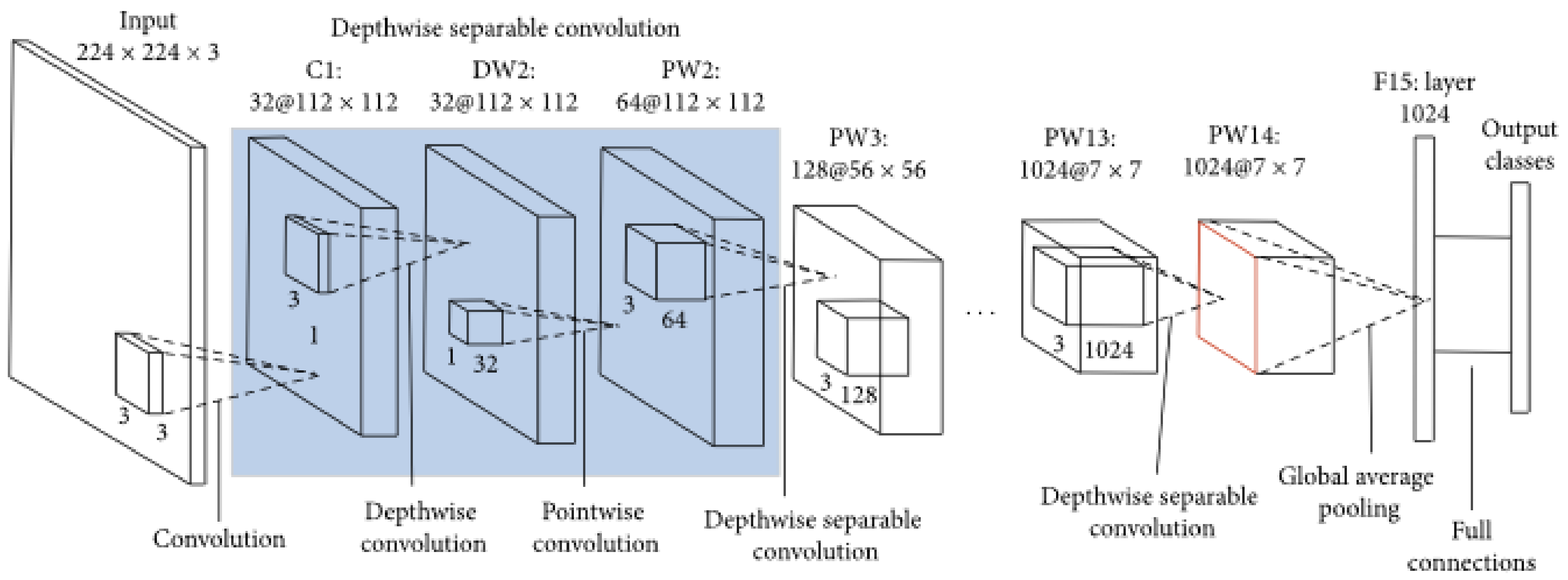
A class of efficient models for mobile and embedded vision applications

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


Object Detection
Photo by Juanedc (CC BY 2.0)
Face Attributes
Google Doodle by Sarah Harrison
MobileNets
Finegrain Classification
Photo by HarshLight (CC BY 2.0)
Landmark Recognition
Photo by Sharon VanderKaay (CC BY 2.0)

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

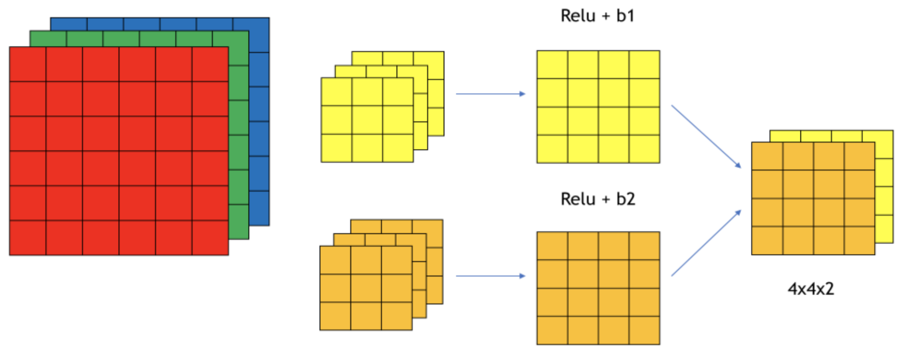
MobileNet Approach: Depthwise Separable Convolutions

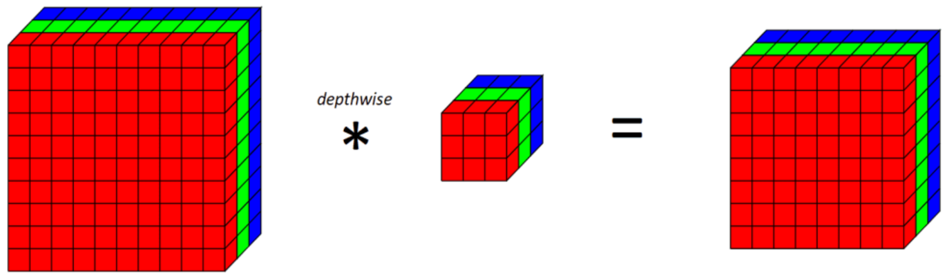
In regular convolution a kernel is applied across all channels of the image

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.



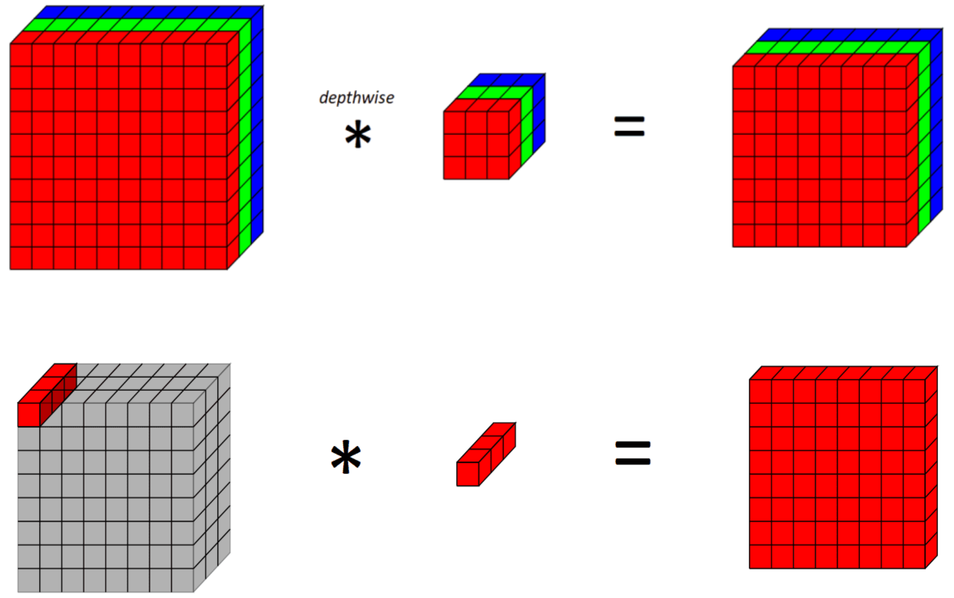


MobileNet Approach: Depthwise Separable Convolution + Pointwise Convolution

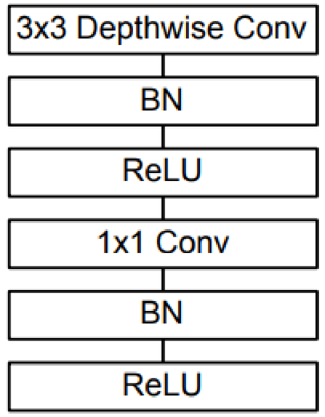
Suppose we have a RGB image of size 224x224x3. You want to apply the convolutional layer with 3x3 size kernel with 64 total kernels

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 !

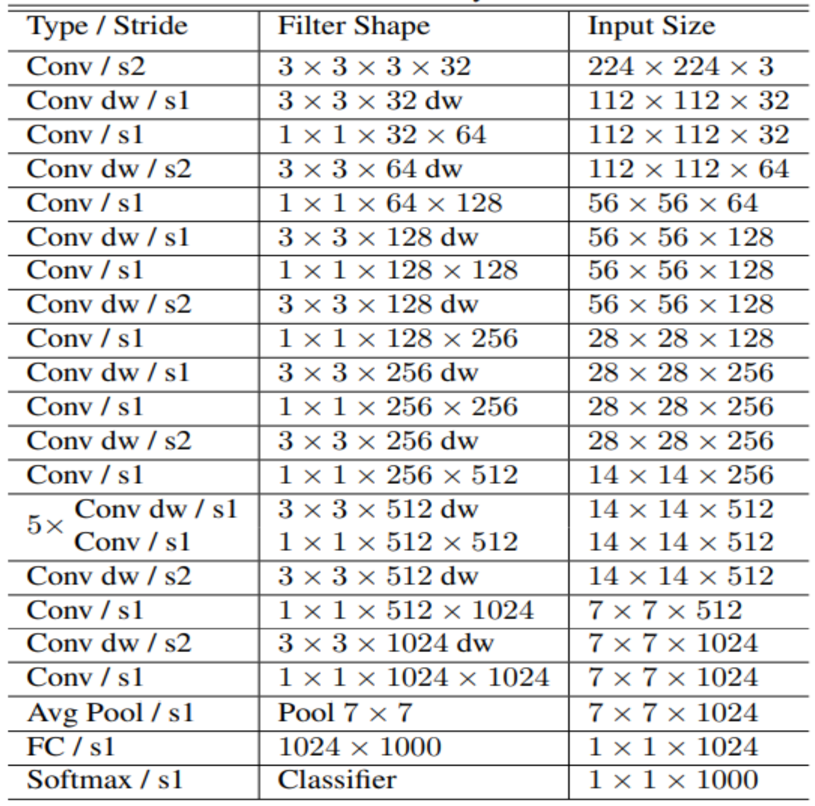


MobileNet Architecture



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]
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 ?

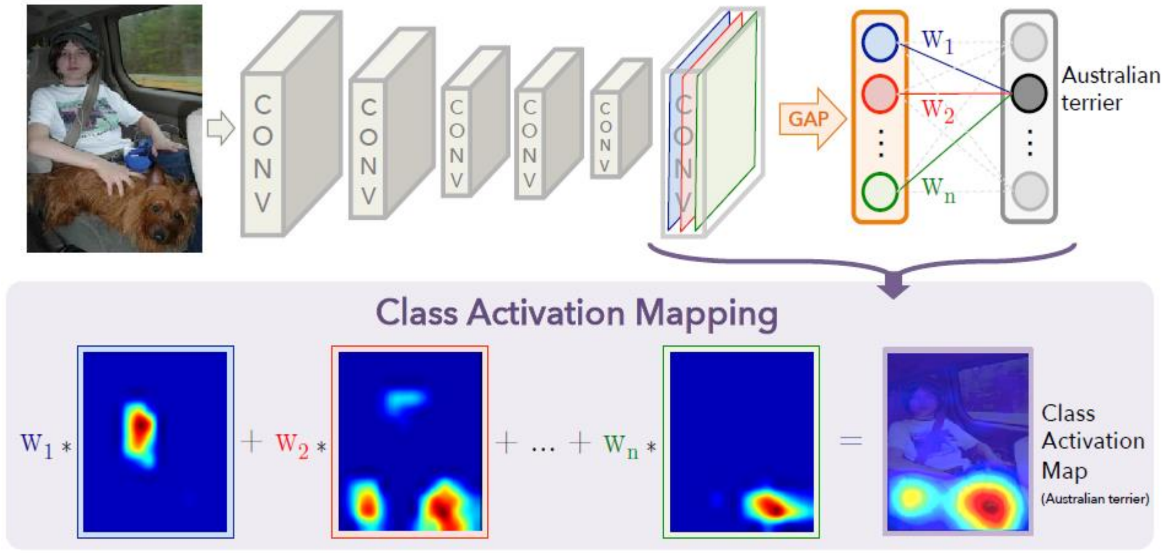
1. Class Activation Maps – Naïve and Restricted Approach
2. Gradient Class Activation Maps (Grad-CAM) – Appropriate for all kinds of tasks
   - Image Classification
   - Image Captioning
   - Visual QnA
   - Image Segmentation

Class Activation Maps

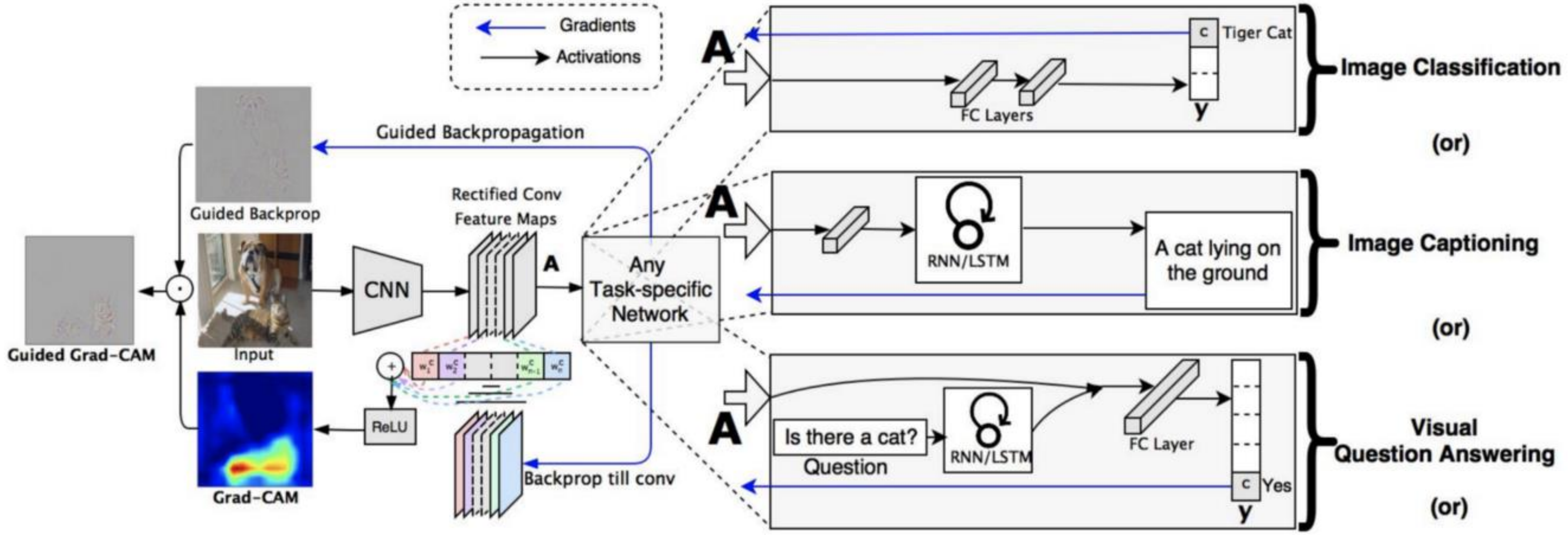
Similar to how you find importance of input features by looking at weights

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

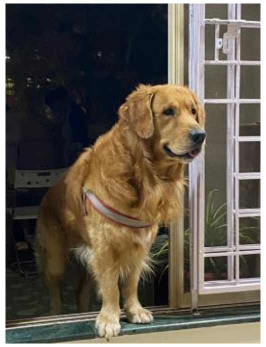


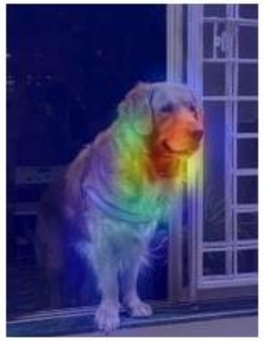
Grad-CAMs


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

Grad-CAMs: Binary Image Classification

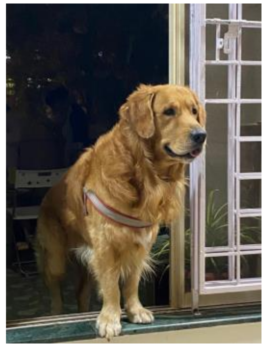
1. Create a custom model in and deep learning framework tf.keras where
   1. Input: image
   2. Output: Gives last Convolution Output and Classification Probability score
2. Get the Classification probability score: Sigmoid activation value
3. Calculate the gradient of this classification activation 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

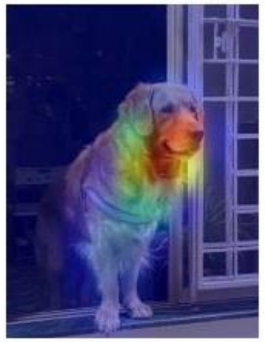




Grad-CAMs: Multiclass Image Classification

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





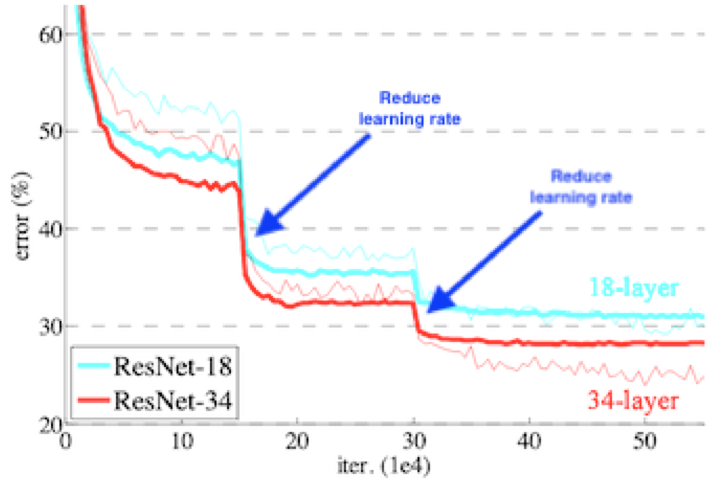
Miscellaneous

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