PadhAI: CNN Architectures

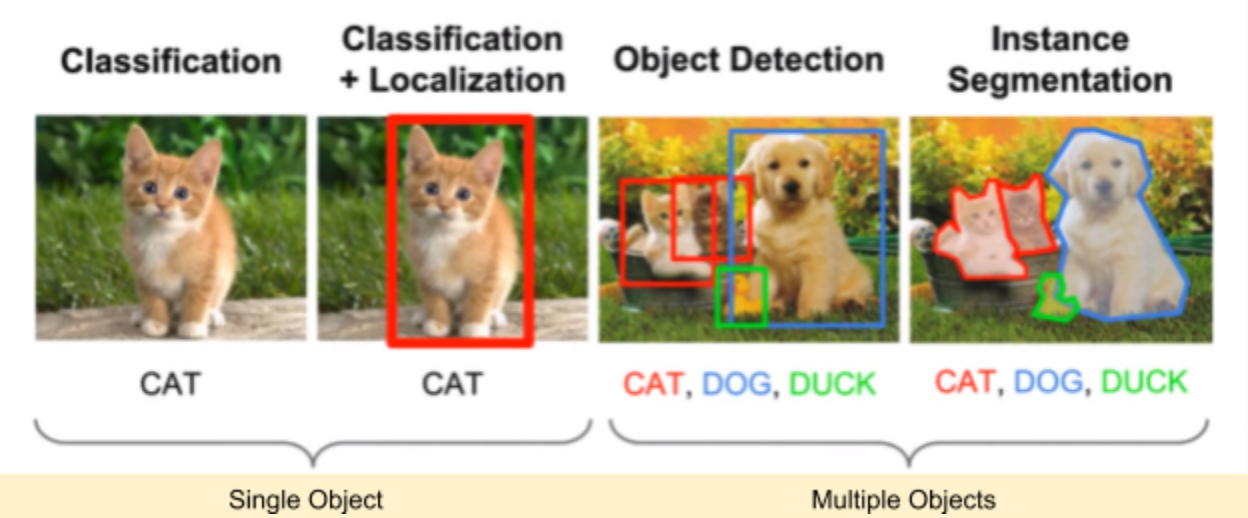
One Fourth Labs

CNN Architectures I

Setting the context

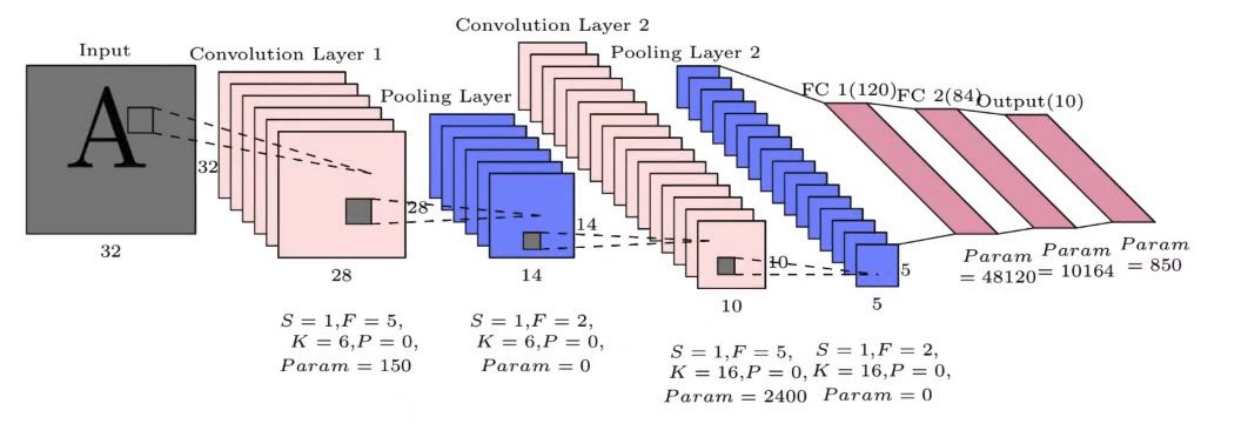
What are CNNs used for?

1. First, let's see **what kind of tasks** are CNNs used for
   a. Consider the following image for an overview of the tasks that CNNs are used for



   b. Using popular datasets like Imagenet, which contains 1000 classes of objects, ranging from vehicles and sceneries, up to the different dog & cat breeds.
   c. **Classification**: In the first image, the task at hand is to correctly predict the class to which the object belongs to. This example uses an "Iconic Photo", i.e. where the class-object of the photo occupies most of the photograph area.
   d. **Classification + Localization**: In the second image, we are predicting the class of the object and also precisely where it is located in the image. Given a starting point outside the object, we detect the width and height of the bounding box enclosing the object. This is both a classification problem and a regression problem.
   e. **Object Detection**: In the third image, we have multiple objects. Therefore, we must correctly detect each object and classify them respectively. It involves multiple Classification + Localization operations on the same image.
   f. **Instance Segmentation**: In the fourth image, we are moving a step further from object detection. Here, we are identifying the precise bounding area around each of the objects present in the image. This is commonly performed in autonomous driving etc, where we have to detect the presence of multiple objects at any given point in time.
   g. These are the four applications that commonly use CNNs. In fact, even our capstone-project of Character-detection and recognition also requires the above mentioned processes.
2. A typical recipe for solving image-related tasks is as follows
   a. First passing the input images through as series of convolutional layer
   b. We obtain a 3D tensor which we flatten to a single dimensional vector
   c. We pass the vector through a number of fully connected layers, which culminate in output prediction
   d. In these cases, we either perform Classification or Regression
3. Now, if we are going to use CNNs for these tasks, we have to make a few choices with regards to the design of the CNN layout.

4. What are some of the decisions that need to be taken?
   a. Let's look at a sample CNN architecture



   b. Some of the factors under our control are as follows:
   c. Number of layers
   d. Number of filters in each layer
   e. Filter Size
   f. Max pooling
5. With the amount of choice we have, designing a CNN architecture could become a very messy process.
6. So a **standard practice is to use tried and tested architectures**.
7. In this chapter, we will be looking at the most popular CNN architectures.

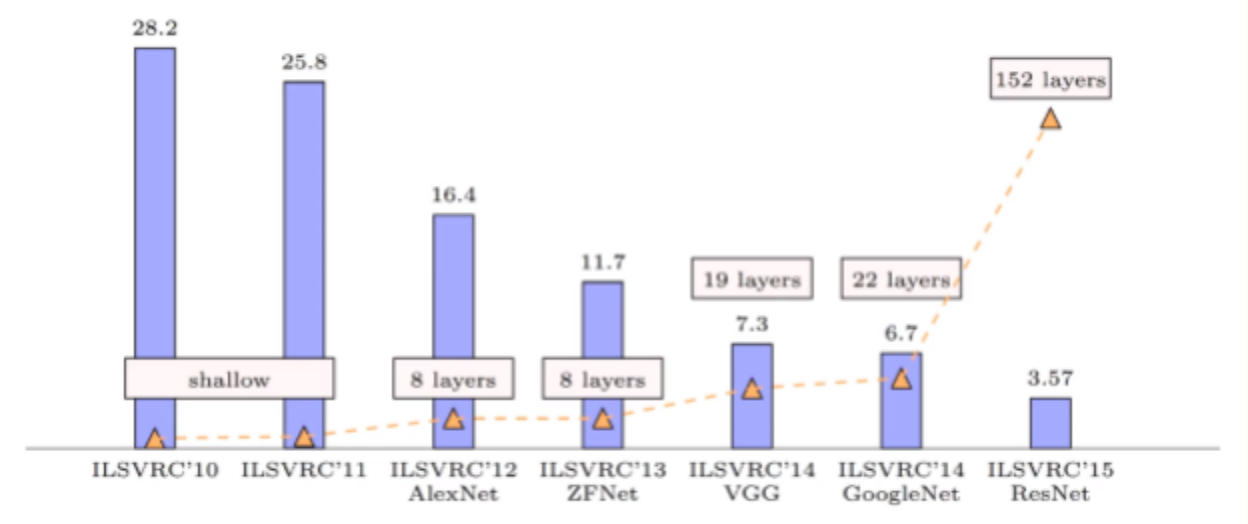
PadhAI: CNN Architectures

One Fourth Labs

The Imagenet Challenge

The Imagenet challenge over the years

1. The Imagenet dataset is a 1000-class, 1,000,000-image dataset (1000 images per class)
2. The **I**magenet **L**arge **S**cale **V**isual **R**ecognition **C**hallenge (**ILSVRC**) or the **Imagenet Challenge** is an annual contest for contender's models to correctly classify the images in the dataset
3. Let us look at the Challenge results between 2010-2015



4. Let us analyse the graph briefly
   a. The metric used to measure performance was Top-5 accuracy. I.e. If any of the top-five predicted class probabilities matched the true class, it was considered correct.
   b. **2010-2011**: This was the pre-DL era, and the Machine Learning models that were submitted had an error between 25-28 %
   c. **2012**: The AlexNet (CNN) architecture smashed existing records with 16.4% error. It kickstarted the Deep Learning Era.
   d. **2013-2014**: Significant improvements were made using models such as ZFNet, VGG and GoogLeNet. Error was brought down to ~6.7%
   e. **2015**: Microsoft's ResNet successfully brought the error down to 3.57% which is lower than the error scored by humans!
   f. One of the reasons for beating the human-error was because some of the classes in the Imagenet Dataset were very fine-grained, i.e. distinguishing between the different dog breeds.
   g. Another interesting point to note is the consistently increasing depth of the Networks used. From shallow networks in the ML era right up to 152 layers in ResNet

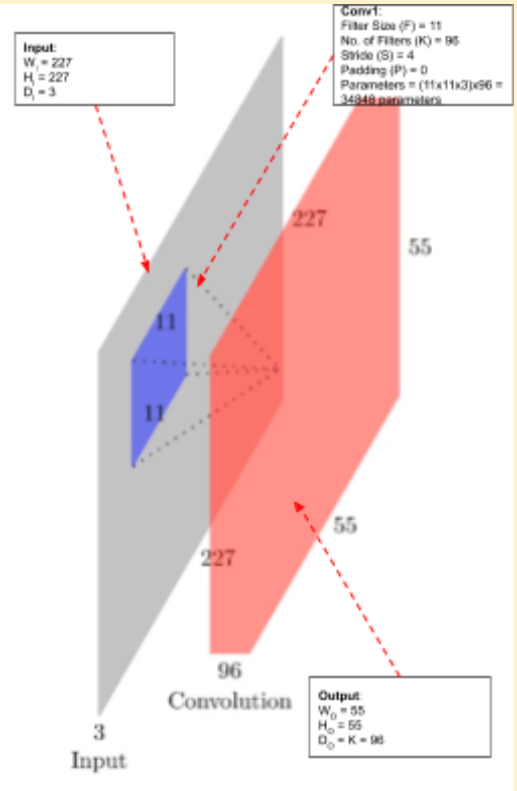
PadhAI: CNN Architectures

One Fourth Labs

Understanding the first layer of AlexNet

Let's break down the first layer of the AlexNet architecture

1. AlexNet was the winning architecture for the 2012 Imagenet Challenge
2. Let us look at the convolutional layer



3. The details are as follows
   a. **Input images**: 227x227x3 (colour images of 227x227 Width x Height)
      i. $W_I = 227$
      ii. $H_I = 227$
      iii. $D_I = 3$
   b. **Filter/Conv1** layer:
      i. Filter Size (F) = 11 (i.e. FxFxD_I or 11x11x3)
      ii. No. of Filters (K) = 96
      iii. Stride (S) = 4
      iv. Padding (P) = 0
      v. Parameters = (11x11x3) x 96 = 34,848
      vi. These values were determined through extensive experimentation
   c. **Output**:
      i. $W_O = \frac{W_I - F + 2P}{S} + 1 = 55$
      ii. $H_O = \frac{H_I - F + 2P}{S} + 1 = 55$
      iii. $D_O = K = 96$

4. This was a standard architecture and can be used for a variety of tasks.

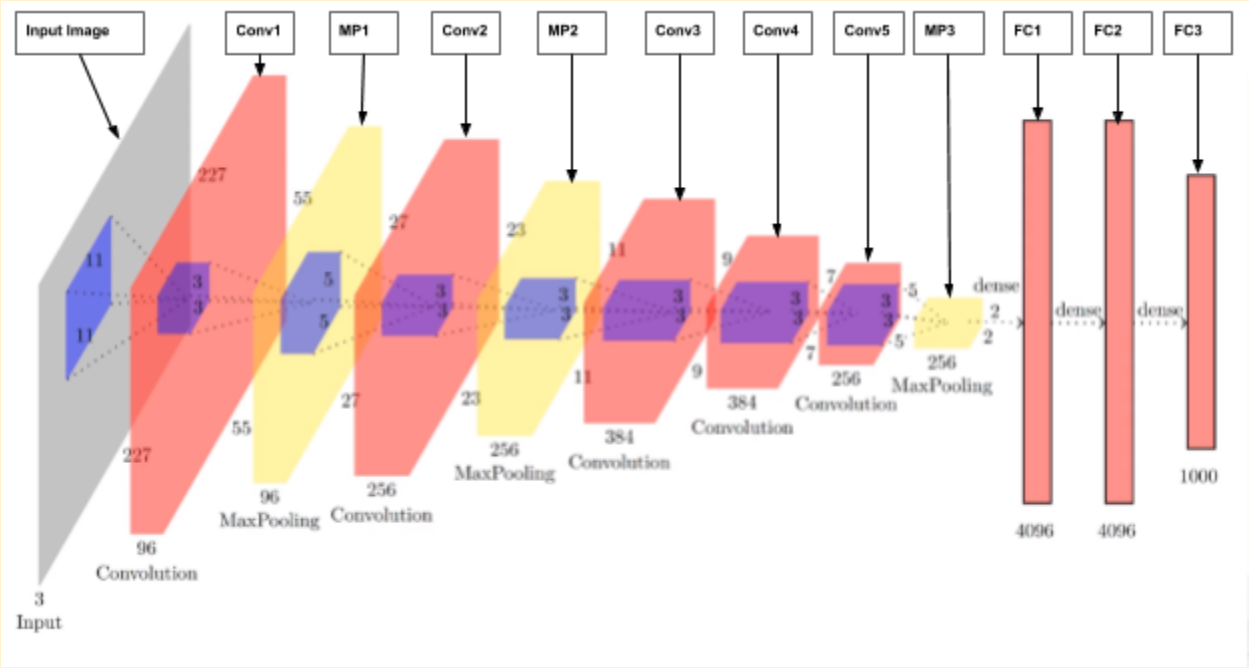
PadhAI: CNN Architectures

One Fourth Labs

Understanding all the layers of AlexNet

Let's now look at the entire AlexNet

1. Before moving into the AlexNet architecture, let us understand why we have decided to use a Convolutional layer instead of a Fully-connected layer
   a. In the convolutional layer, the number of parameters is (11x11x3) x 96 = **34,848**
   b. However, in a FC layer, the number of parameters would be
      i. Input:(227x227x3) x Output:(55x55x96) = **4.49×10^{10}**
   c. Thus, because of sparse-connectivity and weight sharing, we are able to achieve a similar degree of complexity with a significantly smaller number of parameters with a Convolutional layer.
2. Now, let us **break down the entire AlexNet architecture**



3. Let's look at each of the layers in depth
4. **Input Layer**: 227x227x3 (colour images of 227x227 Width x Height)
   a. W_{In} = 227
   b. H_{In} = 227
   c. D_{In} = 3
5. **Convolutional Layer 1**: Input is 227x227x3
   a. Filter Size (**F**) = 11 (11x11x3)
   b. No. of Filters (**K**) = 96
   c. Stride (**S**) = 4
   d. Padding (**P**) = 0
   e. **Parameters** = (11x11x3) x 96 = 34,848
   f. $\mathbf{W_1}$ = 55
   g. $\mathbf{H_1}$ = 55
   h. $\mathbf{D_1}$ = K = 96
   i. **ReLU** Non-linearity function is applied to every 2D area in the output volume.
6. **Max-Pooling Layer 1**: Input is 55x55x96
   a. Filter Size (**F**) = 3 (i.e. 3x3x96)
   b. Stride (**S**) = 4

c. **Parameters** = 0 (no parameters in max pooling)
d. $\mathbf{W_{1m}}$ = 27
e. $\mathbf{H_{1m}}$ = 27
f. $\mathbf{D_{1m}}$ = 96

7. **Convolutional Layer 2**: Input is 27x27x96
   a. Filter Size (**F**) = 5 (5x5x96)
   b. No. of Filters (**K**) = 256
   c. Stride (**S**) = 1
   d. Padding (**P**) = 0
   e. **Parameters** = (5x5x96) x 256 = 614,400
   f. $\mathbf{W_2}$ = 23
   g. $\mathbf{H_2}$ = 23
   h. $\mathbf{D_2}$ = K = 256
   i. **ReLU** Non-linearity function is applied.
8. **Max-Pooling Layer 2**: input is 23x23x256
   a. Filter Size (**F**) = 3 (3x3x256)
   b. Stride (**S**) = 3
   c. **Parameters** = 0
   d. $\mathbf{W_{2m}}$ = 11
   e. $\mathbf{H_{2m}}$ = 11
   f. $\mathbf{D_{2m}}$ = 256
9. **Convolutional Layer 3**: input is 11x11x256
   a. Filter Size (**F**) = 3 (3x3x256)
   b. No. of Filters (**K**) = 384
   c. Stride (**S**) = 1
   d. Padding (**P**) = 0
   e. **Parameters** = (3x3x256) x 384 = 884,736
   f. $\mathbf{W_3}$ = 9
   g. $\mathbf{H_3}$ = 9
   h. $\mathbf{D_3}$ = K = 384
   i. **ReLU** Non-linearity function is applied.
10. **Convolutional Layer 4**: input is 9x9x384
    a. Filter Size (**F**) = 3 (3x3x384)
    b. No. of Filters (**K**) = 384
    c. Stride (**S**) = 1
    d. Padding (**P**) = 0
    e. **Parameters** = (3x3x384) x 384 = 1,327,104
    f. $\mathbf{W_4}$ = 7
    g. $\mathbf{H_4}$ = 7
    h. $\mathbf{D_4}$ = K = 384
    i. **ReLU** Non-linearity function is applied.
11. **Convolutional Layer 5**: input is 7x7x384
    a. Filter Size (**F**) = 3 (3x3x384)
    b. No. of Filters (**K**) = 256
    c. Stride (**S**) = 1
    d. Padding (**P**) = 0
    e. **Parameters** = (3x3x384) x 256 = 884,736
    f. $\mathbf{W_5}$ = 5
    g. $\mathbf{H_5}$ = 5

h. $\mathbf{D_5}$ = K = 256
i. **ReLU** Non-linearity function is applied.

12. **Max-Pooling Layer 3**: input is 5x5x256
    a. Filter Size (**F**) = 3 (3x3x256)
    b. Stride (**S**) = 2
    c. **Parameters** = 0
    d. $\mathbf{W_{5m}}$ = 2
    e. $\mathbf{H_{5m}}$ = 2
    f. $\mathbf{D_{5m}}$ = 256
13. **Fully Connected Layer 1**: input is 2x2x256 = 1024
    a. Number of Neurons = 4096
    b. Parameters = (2x2x256) x 4096 = 4,194,304
14. **Fully Connected Layer 2**: input = 4096
    a. Number of Neurons = 4096
    b. Parameters = 4096 x 4096 = 16,777,216
15. **Fully Connected Layer 3**: input = 4096
    a. Number of Neurons/output-classes = 1000
    b. Parameters = 4096 x 1000 = 4,096,000

16. Totally, there are around 27.55 Million parameters, out of which roughly 25 Million parameters were in the last 3 Fully-connected layers.
17. When counting the total number of layers, we do not include the max-pooling layers as they do not carry weights. Thus, we say that **AlexNet has 8 layers**

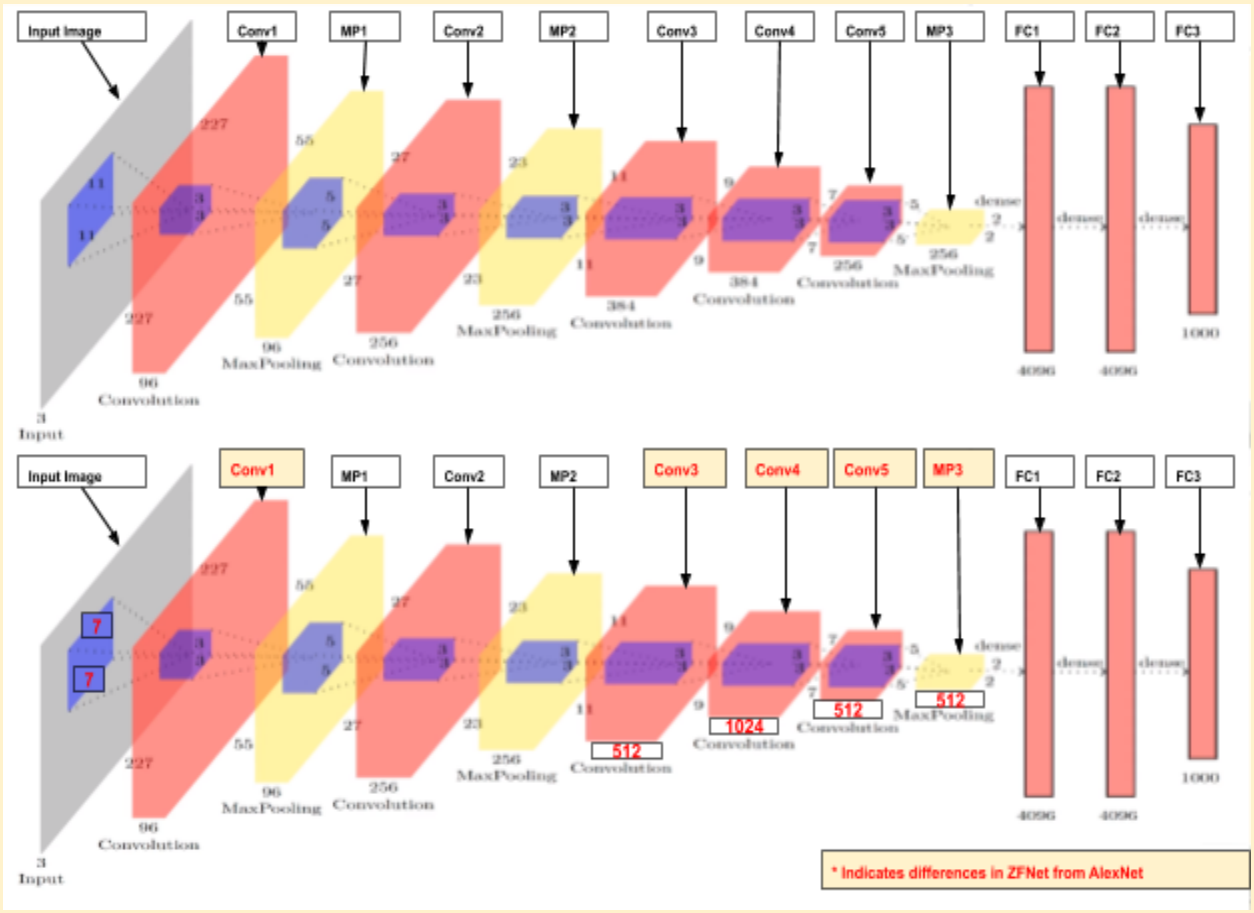
PadhAI: CNN Architectures

One Fourth Labs

ZFNet

Let's now look at the entire AlexNet

1. ZFNet is another 8-layer CNN architecture. Let's understand it better with a side-by-side comparison with AlexNet.



2. ZFNet is largely similar to AlexNet, with the exception of a few of the layers. Let us highlight those differences.
3. **Convolutional Layer 1**: Input is 227x227x3
   a. Filter Size (**F**) = 7 (7x7x3)
   b. No. of Filters (**K**) = 96
   c. Stride (**S**) = 4
   d. Padding (**P**) = 0
   e. **Parameters** = (7x7x3) x 96 = 14,112
   f. $\mathbf{W_1}$ = 55
   g. $\mathbf{H_1}$ = 55
   h. $\mathbf{D_1}$ = K = 96
   i. **ReLU** Non-linearity function is applied to every 2D area in the output volume.

4. **Convolutional Layer 3**: input is 11x11x256
   a. Filter Size (**F**) = 3 (3x3x256)
   b. No. of Filters (**K**) = 512
   c. Stride (**S**) = 1
   d. Padding (**P**) = 0
   e. **Parameters** = (3x3x256) x 512 = 1,179,648
   f. $\mathbf{W_3}$ = 9
   g. $\mathbf{H_3}$ = 9
   h. $\mathbf{D_3}$ = K = 512
   i. **ReLU** Non-linearity function is applied.
5. **Convolutional Layer 4**: input is 9x9x512
   a. Filter Size (**F**) = 3 (3x3x512)
   b. No. of Filters (**K**) = 1024
   c. Stride (**S**) = 1
   d. Padding (**P**) = 0
   e. **Parameters** = (3x3x512) x 1024 = 4,718,592
   f. $\mathbf{W_4}$ = 7
   g. $\mathbf{H_4}$ = 7
   h. $\mathbf{D_4}$ = K = 1024
   i. **ReLU** Non-linearity function is applied.
6. **Convolutional Layer 5**: input is 7x7x1024
   a. Filter Size (**F**) = 3 (3x3x1024)
   b. No. of Filters (**K**) = 512
   c. Stride (**S**) = 1
   d. Padding (**P**) = 0
   e. **Parameters** = (3x3x1024) x 512 = 4,718,592
   f. $\mathbf{W_4}$ = 5
   g. $\mathbf{H_4}$ = 5
   h. $\mathbf{D_4}$ = K = 512
   i. **ReLU** Non-linearity function is applied.
7. **Max-Pooling Layer 3**: input is 5x5x512
   a. Filter Size (**F**) = 3 (3x3x512)
   b. Stride (**S**) = 2
   c. **Parameters** = 0
   d. $\mathbf{W_{2m}}$ = 2
   e. $\mathbf{H_{2m}}$ = 2
   f. $\mathbf{D_{1m}}$ = 512
8. **Fully Connected Layer 1**: input is 2x2x512 = 2048
   a. Number of Neurons = 4096
   b. Parameters = (2x2x512) x 4096 = 8,388,608
9. The **total difference in the number of parameters** ZFNet - AlexNet = 1.45 Million

10. There are other variants of ZFNet where we use a stride of 2 in the first convolutional layer, thereby changing the subsequent layer dimensions.

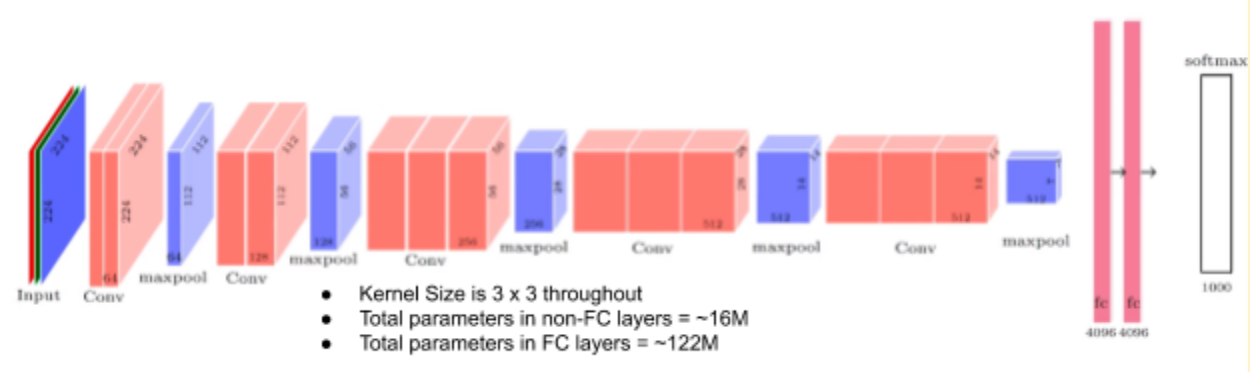
PadhAI: CNN Architectures

One Fourth Labs

VGGNet

Let's look at the architecture of VGGNet

1. During the design of the VGGNet, it was found that alternating convolution & pooling layers were not required. So VGGnet uses multiple of Convolutional layers in sequence with pooling layers in between.
2. Let us break down the VGGNet architecture



3. A few points to note
4. The kernel size 3x3 is maintained throughout the network, only the depth is changed between layers
5. Appropriate padding is provided to maintain the dimensions across the layers
6. **Convolutional Bundle 1**: There are **2 convolutional layers** of size **224x224x64**
7. **Max Pool Layer 1**: The size is **112x112x64**
8. **Convolutional Bundle 2**: There are **2 convolutional layers** of size **112x112x128**
9. **Max Pool Layer 2**: The size is **56x56x128**
10. **Convolutional Bundle 3**: There are **3 convolutional layers** of size **56x56x256**
11. **Max Pool Layer 3**: The size is **28x28x256**
12. **Convolutional Bundle 4**: There are **3 convolutional layers** of size **28x28x512**
13. **Max Pool Layer 4**: The size is **14x14x512**
14. **Convolutional Bundle 5**: There are **3 convolutional layers** of size **14x14x512**
15. **Max Pool Layer 5**: The size is **7x7x512**
16. The number of **parameters in the Non-FC layers is ~16 Million**
17. **FC Layer 1** has **4096** Neurons
18. **FC Layer 2** has **4096** Neurons
19. **FC Layer 3** is a softmax with **1000** Neurons/Output-classes
20. The number of **parameters in the FC layers is ~122 Million** (Most in FC Layer 1: ~102 Million)
21. Though the number of parameters in this network seems very large, it would have been exponentially larger if we had chosen an entirely Fully-Connected network.
22. The above shown VGGNet is a 16 layer network called VGG16. There are also other versions like the 19 layered VGG19

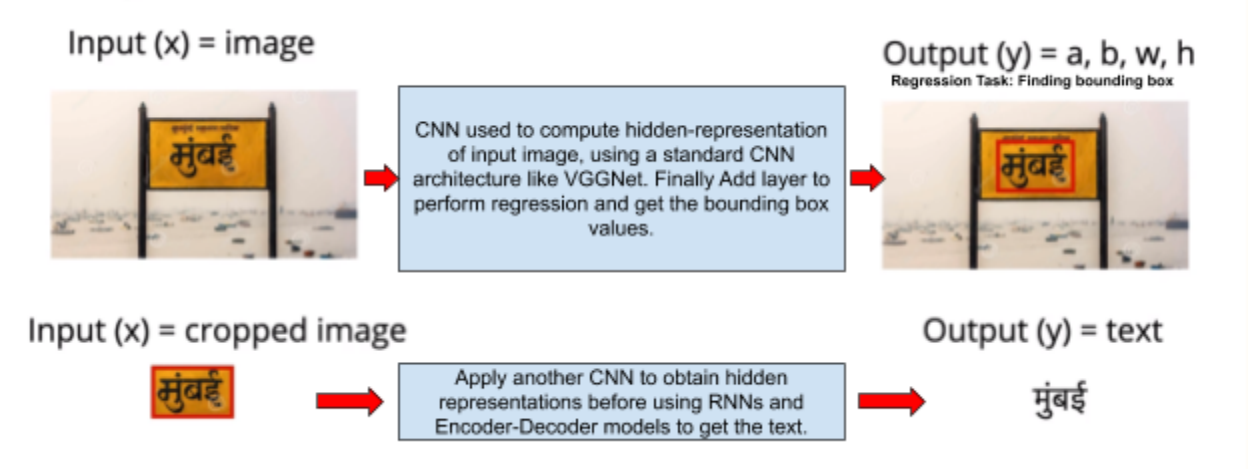
PadhAI: CNN Architectures

One Fourth Labs

Summary

Connecting this to the capstone project

1. Let's see how CNNs come into play for our Signboard translation capstone project



2. The above diagram shows how we use CNNs in our capstone project.

PadhAI: CNN Architectures

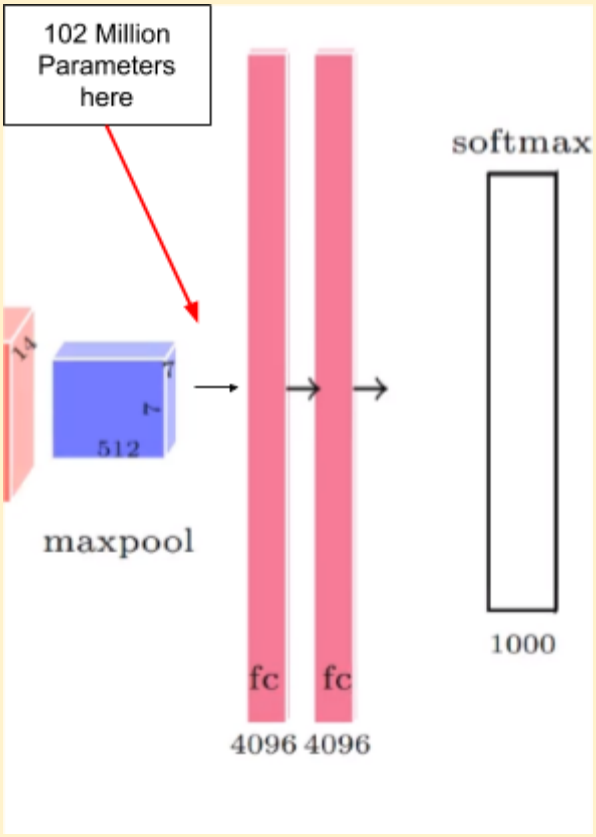
One Fourth Labs

CNN Architectures II

Setting the context

How do we address what's been bothering us so far with CNNs

1. To pick up from VGGNet, can we do something to reduce the huge number of parameters incurred between the non-FC Layer and the FC Layer.



2. Another question we can ask is why we must stop at 16 layers, can we not go deeper.
3. So by combining the above two points, we can make the network deeper to reduce the number of parameters at the interface between the non-FC and the FC layers.
4. We must also make sure that the number of computations (Sliding a filter across the input) is not very high.
5. Our problem points can be summarised as follows
   a. Increase choice of filters
   b. Reduce number of parameters
   c. Reduce number of computations
   d. Make a deeper network

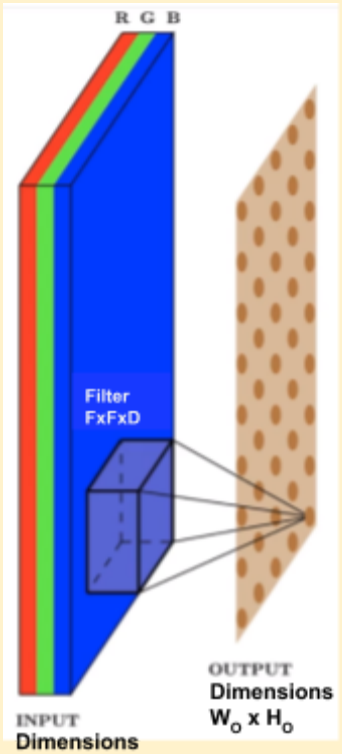
PadhAI: CNN Architectures

One Fourth Labs

Number of computations in a convolution layer

Let's see how many computations are needed in a CNN.

1. We will be looking at the GoogLeNet architecture as an improvement to the VGGNet based on the points discussed in the previous section.
2. However, before that, we must look at two key concepts in the GoogLeNet layout: **1x1 convolution** and an **interesting way to perform max-pooling**.
3. To approach these two, we first need to see how many computations are needed in one convolutional layer



Assume S = 1 and we have used appropriate padding so that W_O = W_I = W and H_O = H_I = H

   a. **Input dimensions**: W_I x H_I x D_I
   b. **Filter size**: F x F x D_I
   c. **Output dimensions**: W_O x H_O
   d. Stride = 1 and appropriate padding so that **W_O = W_I = W and H_O = H_I = H**
4. To calculate the number of computations:
   a. For every pixel of interest, for D layers, we perform FxFxD computations
   b. So for an output area of WxH, we perform (WxH) x (FxFxD) computations
   c. From the previous point, we can observe that the Depth of the output layer will be very large if there is a large number of filters applied on the input layer, as each filter generates a 2D area of unit depth.
   d. So if we use a **large number of filters**, the **output volume will be very deep**, subsequently **increasing the number of computations in the next layer's calculation** (Due to high D value).
   e. We can also try controlling W and H, but they can be more easily regulated using max-pooling. However, depth is directly related to the number of filters used.

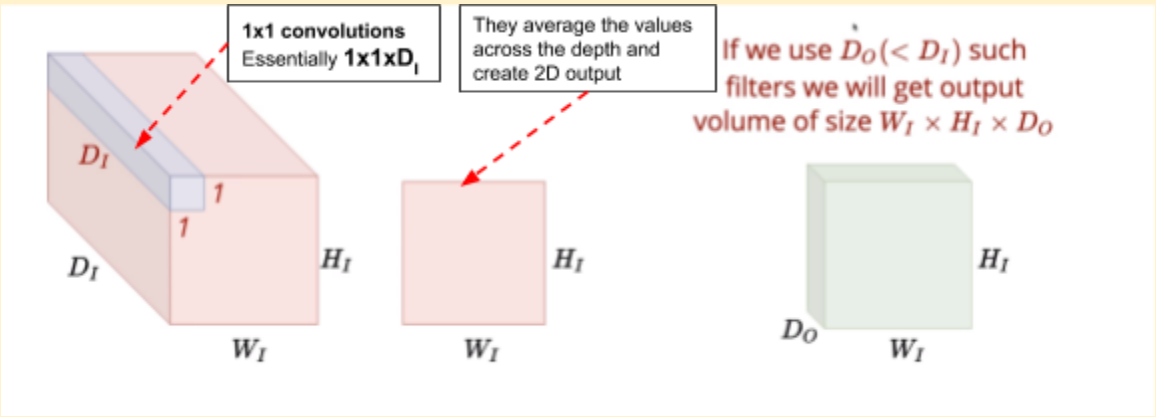
PadhAI: CNN Architectures

One Fourth Labs

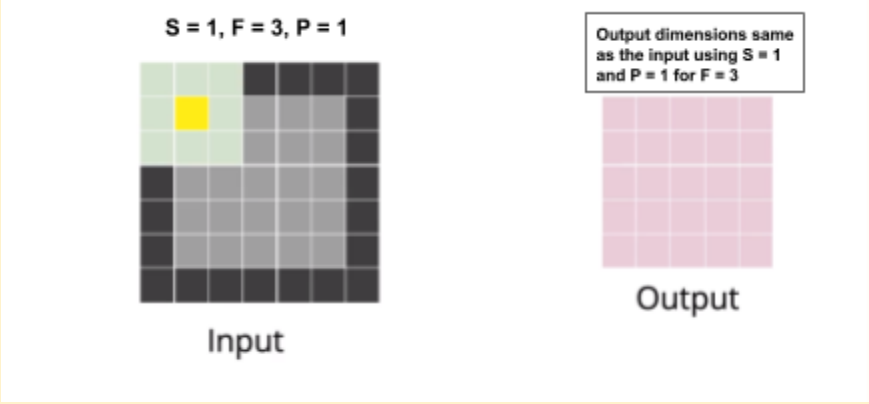
1x1 Convolutions

What is a 1x1 convolution used for?

1. We've mostly worked with 3x3 convolutions so far, i.e. a grid containing 3 rows, 3 columns with 9 cells in total.
2. The result of a single 3x3 operation is the weighted average of all the points in the grid, applied to our selected pixel.
3. Now, let us look at a 1x1 convolution



4. A 1x1 kernel takes a neighborhood of 1 row and 1 column, which is essentially the pixel itself. Since the kernel is of size $1x1xD_I$, it computes the weighted average of all the pixels across the input depth D_I
5. Here, the Input is 3D, the filter is 3D but the operation is 2D, as we are only moving horizontally and vertically. The 3D volume is compressed to a 2D area.
6. If we were to use D_O number of filters, where $(D_O < D_I)$, we will get an output of $W_I \times H_I \times D_O$. Each of the 1x1 kernels will give one 2D output and D_O such kernels gives us an output volume of the dimensions $W_I \times H_I \times D_O$.
7. If D_O is much smaller than D_I, we effectively shrink the input volume while still effectively retaining the depth information (Due to averaging across depth).
8. Now, this output behaves as an input to the next layer, resulting in a much smaller number of computations due to smaller depth.
9. In a nutshell, 1x1 filters are used to compress input volumes across their depth to get a smaller output volume of same Width and Height.
10. Another operation we need to look at is Max Pooling. We usually perform Max-pooling with a Stride=2, resulting in halving the input dimensions. However, we can also perform it with a stride of 1. With S = 1 and appropriate padding, we can preserve input dimensions.



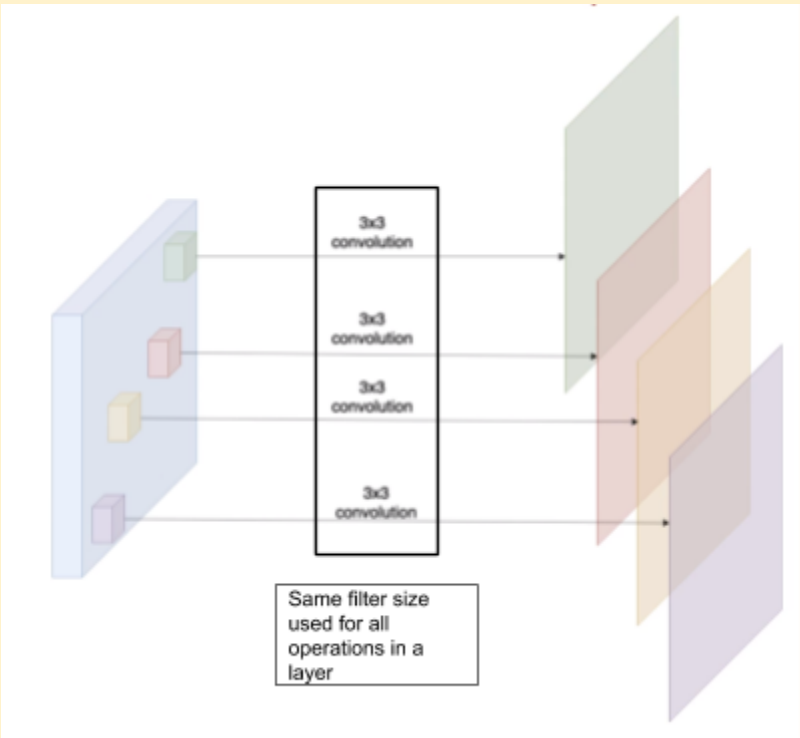
PadhAI: CNN Architectures

One Fourth Labs

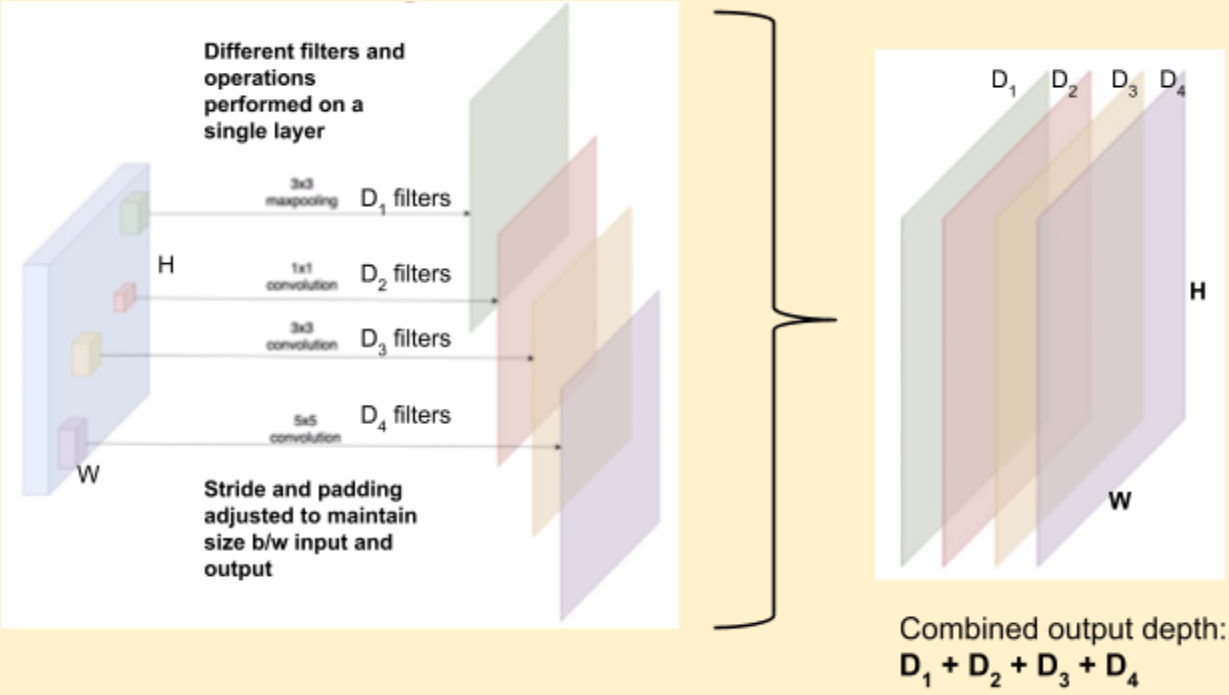
The Intuition behind GoogLeNet

What is the intuition behind GoogLeNet?

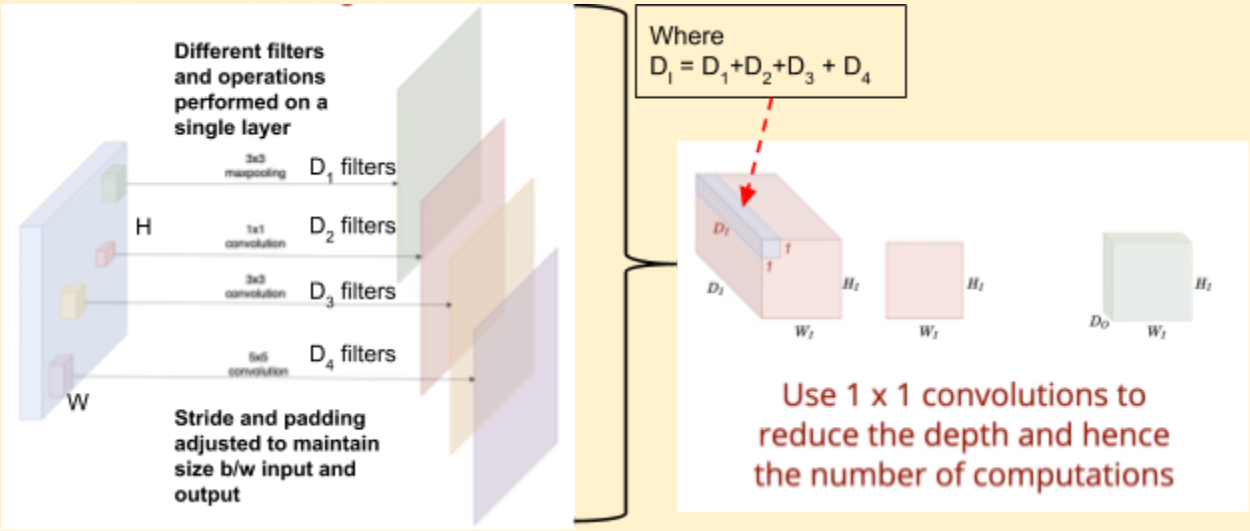
1. One point to note in the architectures used thus far, is that we must always make a choice of a particular filter size for any given layer. For eg, in VGGNet, all filters used were 3x3
2. Another point is the interspersing of Max-pooling layers between convolutional layers. How do we decide the arrangement to follow?



3. In GoogLeNet, the choice was eliminated, instead we are able to apply all our operations whatever combination of filter size and max-pooling/non that we'd like.



4. In GoogLeNet, we can apply multiple filters of varying size to perform either convolutional or max-pooling operations
5. Through experimentation with the older architectures, we have found that when there are multiple convolutional operations in a layer, we needn't use larger filter sizes.
6. Therefore, 5x5 filters are usually the upper limit of size.
7. One constraint is that for each operation in a layer, appropriate padding and stride must be taken so as to preserve the width and height between the input and the output
8. The Number of filter for each operation can vary, (D_0 to D_3 etc)
9. In the output volume, the total depth is the combined depth of the individual volumes from each of the operations. $D = D_0 + D_1 + D_2 + D_3$
10. The problem with combining the depths is that it has the potential to become very large, thus drastically increasing the number of computations.
11. We can mitigate this problem by performing 1x1 convolutions



12. By performing 1x1 convolutions, we can reduce the depth of the output volume, thereby reducing the number of computations to be performed in the subsequent layers.

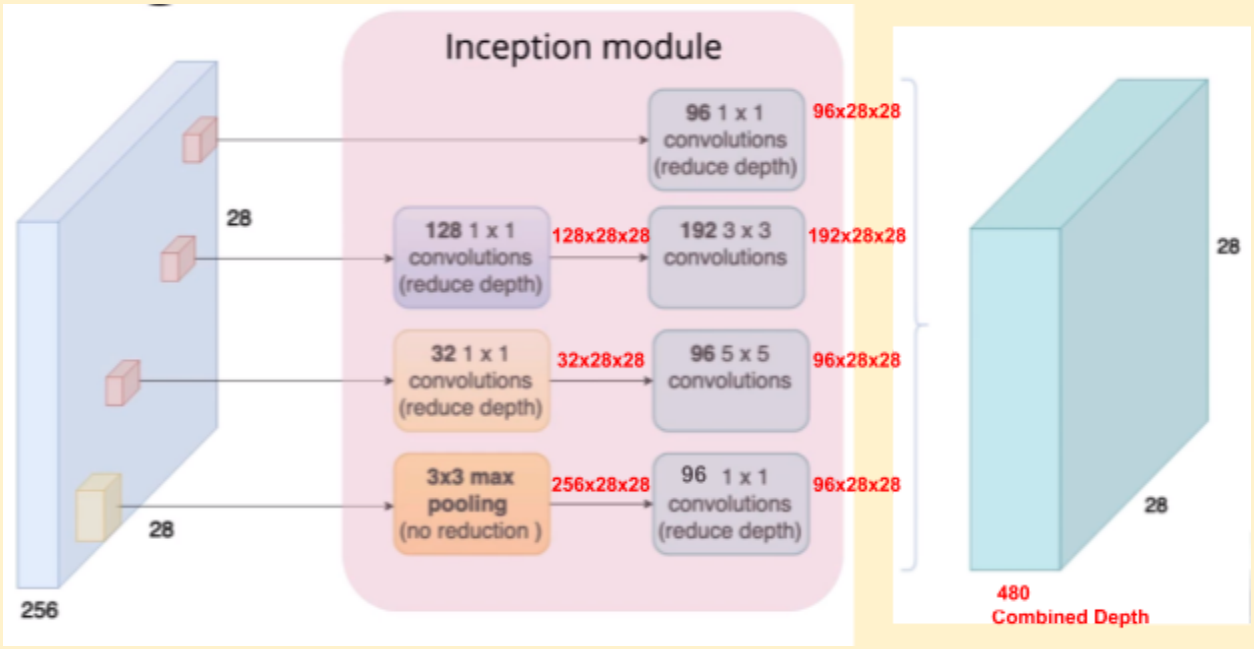
PadhAI: CNN Architectures

One Fourth Labs

The Inception Module

What is the Inception Module?

1. GoogLeNet is also called Inception net. It is made up of multiple modular operation-blocks known as inception modules.
2. Let us break down a single inception module



3. Here, we can see the sequence of operations performed in this inception module.
4. There are a few operations that are a blueprint of inception modules
5. Direct 1x1 convolutions
6. 1x1 convolutions followed by 3x3 and 5x5 convolutions
7. 3x3 convolutions followed by 1x1 convolutions
8. Here, The number of filters can change but the same blueprint repeats itself throughout the network.
9. Summing the output of all of these, we get the output volume as shown in the figure.

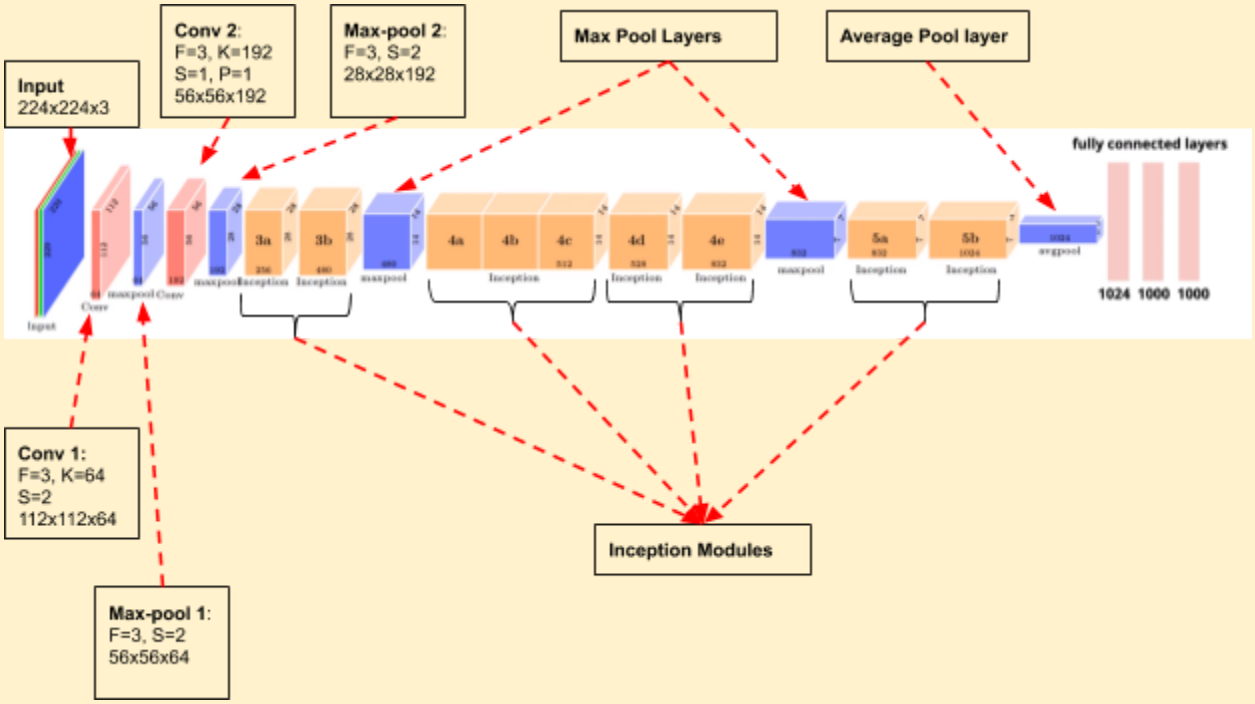
PadhAI: CNN Architectures

One Fourth Labs

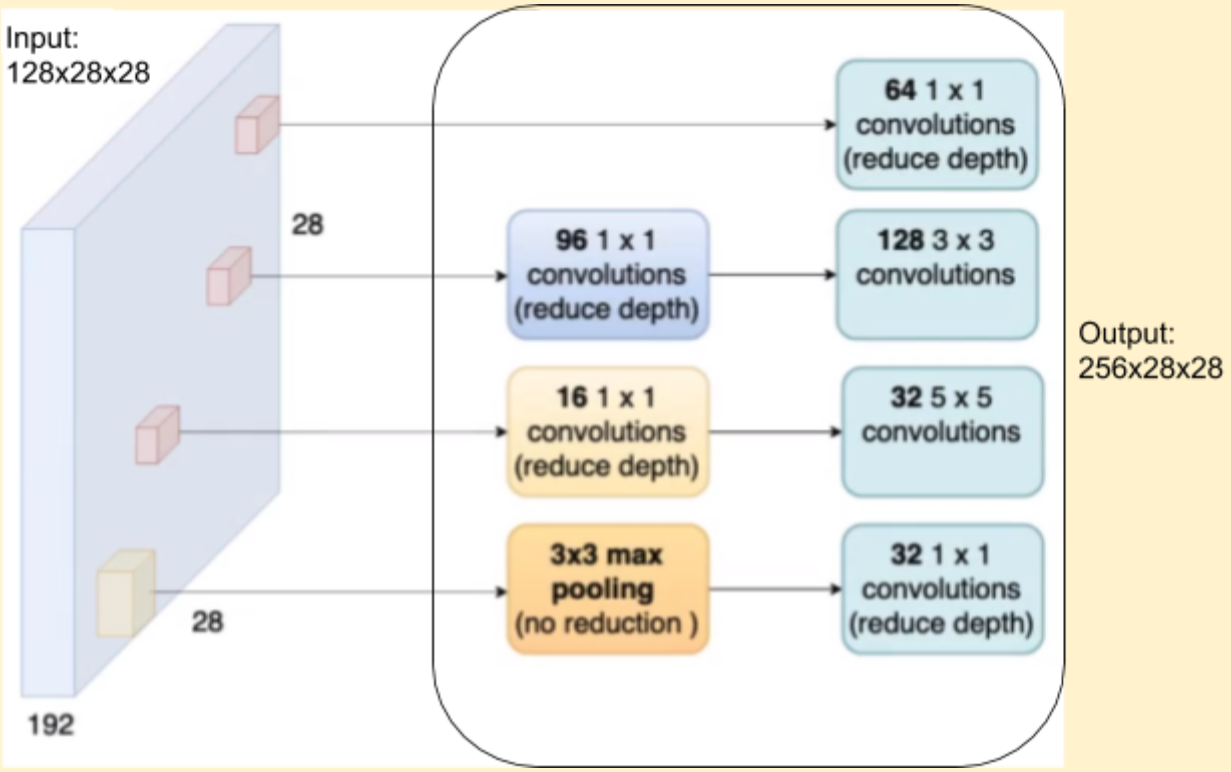
The GoogLeNet Architecture

What does the full network look like?

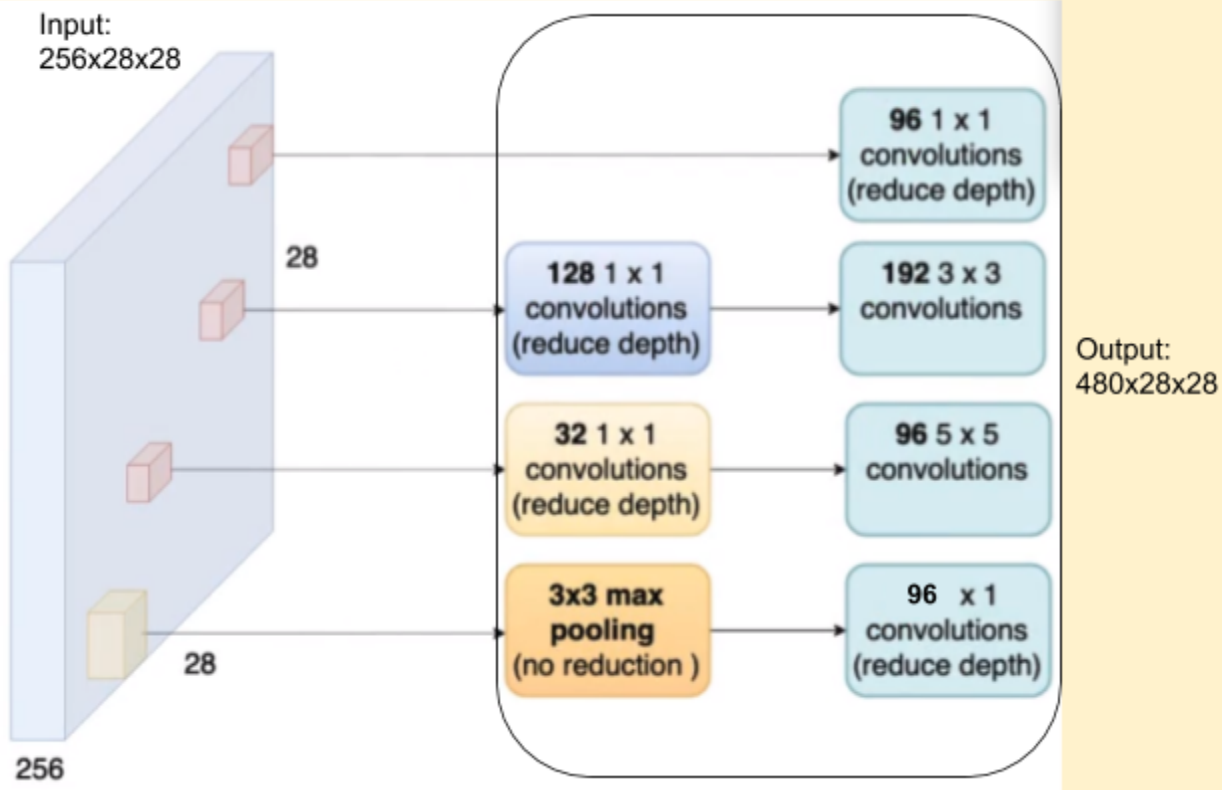
1. Let's take a look at the entire GoogLeNet architecture



2. Up till the second max-pooling layer, the architecture is similar to what we've seen in earlier configurations. Post that, we begin moving into the Inception modules.
3. **Inception Module 1:**



4. **Inception Module 2**:

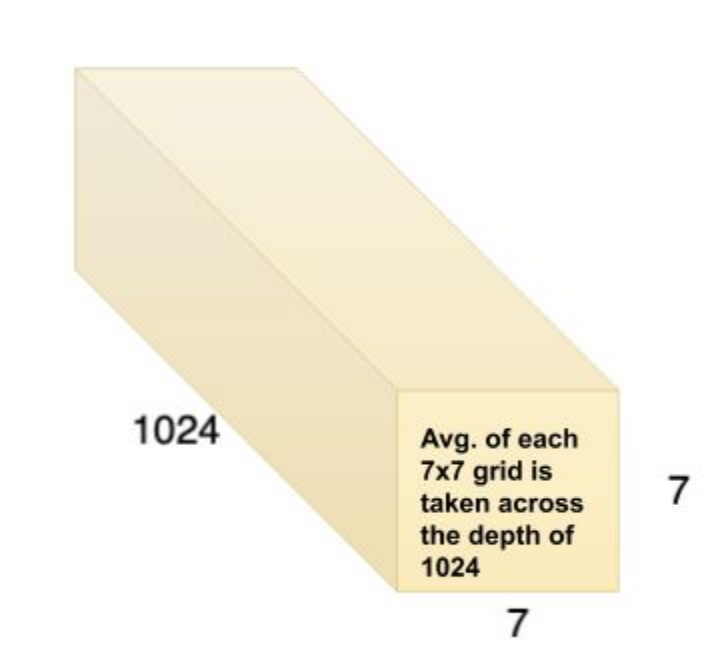


5. Then there is a **Max-pooling layer which reduces the Dimensions by Half (480x14x14)**. This Max-pooling layer is used because the Max-pooling layers in the Inception modules do not reduce the dimensions.
6. **Inception Module 3,4 & 5**:
   a. **Input at Inception Module 3** is: 480x14x14
   b. **Output at Inception Module 5** is: 512x14x14
7. **Inception Module 6**:
   a. **Input**: 512x14x14
   b. **Output**: 528x14x14
8. **Inception Module 7**:
   a. **Input**: 528x14x14
   b. **Output**: 832x14x14
9. Then there is a **Max-pooling layer which reduces the Dimensions by Half (832x7x7)**.
10. **Inception Module 8**:
    a. **Input**: 832x7x7
    b. **Output**: 832x7x7
11. **Inception Module 9**:
    a. **Input**: 832x7x7
    b. **Output**: 1024x7x7
12. **Average Pool layer is used to reduce the output from Inception Module 9**, thereby reducing the number of parameters between the non-FC and FC layer interface.
13. Each Inception Module counts as 2 layers, therefore we have more than 20 layers in GoogLeNet.

Average Pooling

How does average pooling reduce the output size?

1. At the final Inception Module, we have an output dimension of 1024x7x7. If this was to directly interface with a Fully-Connected layer with a 1000 Neurons, we would get ~50 Million parameters
2. To reduce this number, Google added another layer which performs Average-pooling



3. By taking the average value of each 7x7 slice to get a 1x1 value across the depth, we are left with a 1024x1x1 vector.
4. This vector of 1024 values interfaces with the Fully connected layer with a 1000 Neurons.
5. This gives us ~1 Million parameters, as opposed to the earlier seen 50 Million parameters.

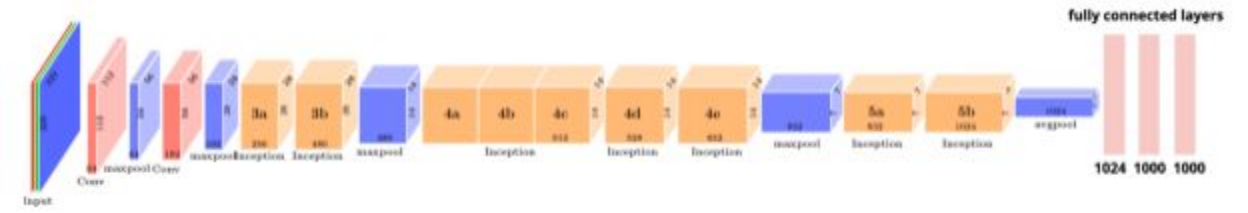
PadhAI: CNN Architectures

One Fourth Labs

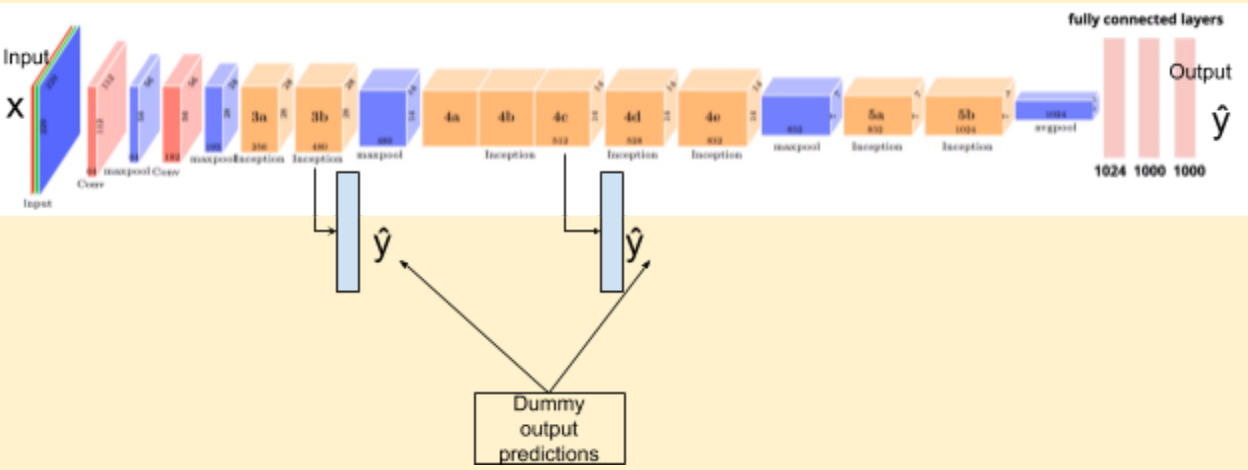
Auxiliary Loss for training a deep network

Can auxiliary loss help to train the network better?

1. Let's look at how GoogLeNet responds to the 4 problem points from the previous CNN architectures



2. **Increase choice of filters**: Parallel convolutions/max-pooling
3. **Reduce number of parameters**: Average Pooling
4. **Reduce number of computations**: 1x1 convolutions
5. **Make a deeper network**: Has 22 layers as opposed to VGG19's 19 layers.
6. Now, since it is a very deep network, there is a possibility for vanishing gradients to occur when backpropagating the Loss. This is mitigated using a technique called Auxiliary Loss



7. In addition to the final output prediction, we are also trying to make partial predictions from the above specified regions in the network.
8. We compute the loss at the final prediction and at both the dummy predictions.
9. Now we can backpropagate from the final loss or from the dummy-losses obtained, thereby shortening the effective depth of the network and lowering the chance of vanishing gradients occurring.
10. Some interesting points to note about GoogLeNet
    a. 12x less parameters than AlexNet
    b. 2x more computations than AlexNet
    c. Improved performance on ImageNet

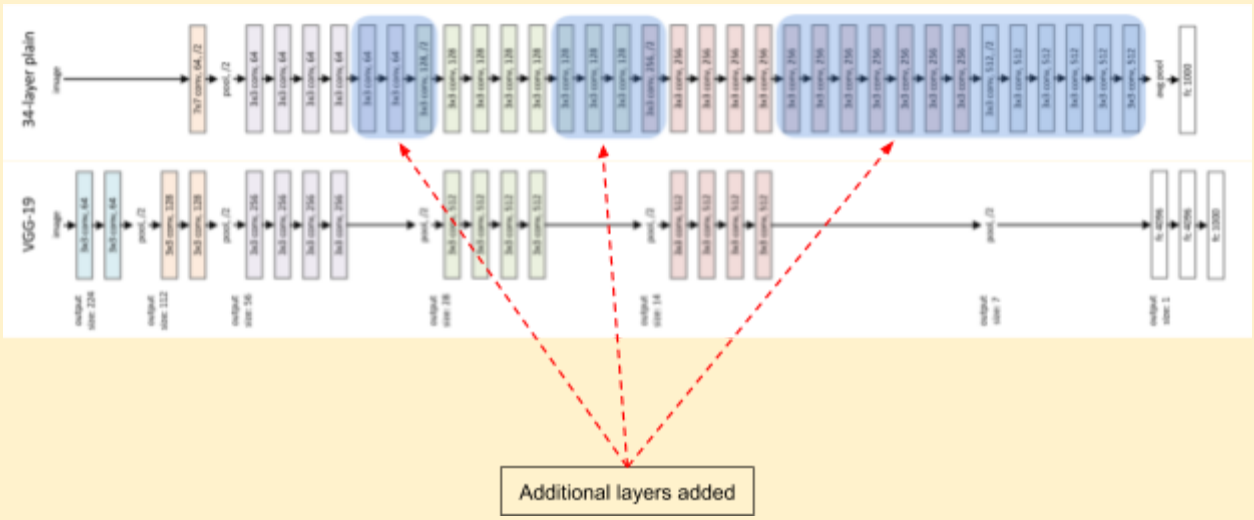
PadhAI: CNN Architectures

One Fourth Labs

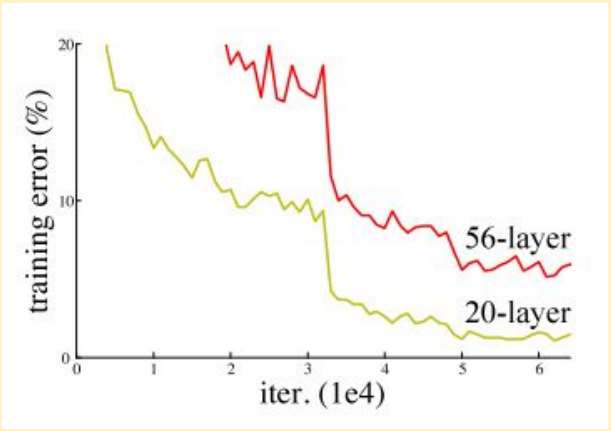
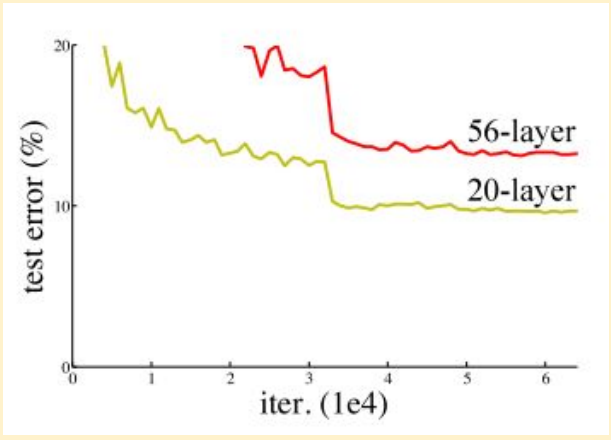
ResNet

What happens if you increase the depth of the network?

1. Let us consider the basic architecture of the VGG-19 network when compared to another 34 layer network architecture

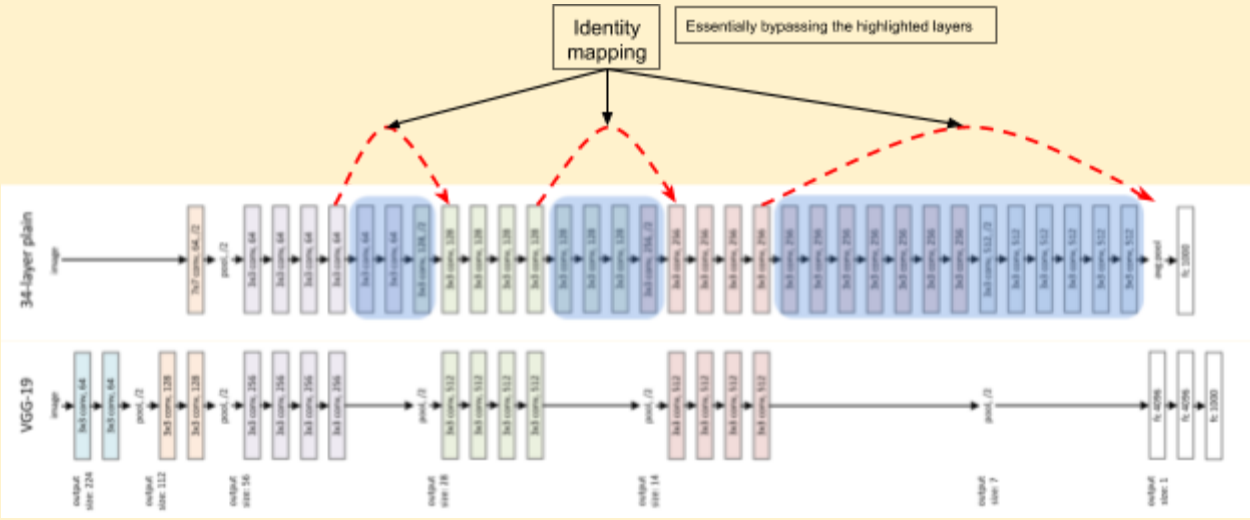


2. The train/test error curves were plotted for some other 20-layer and 56-layer networks

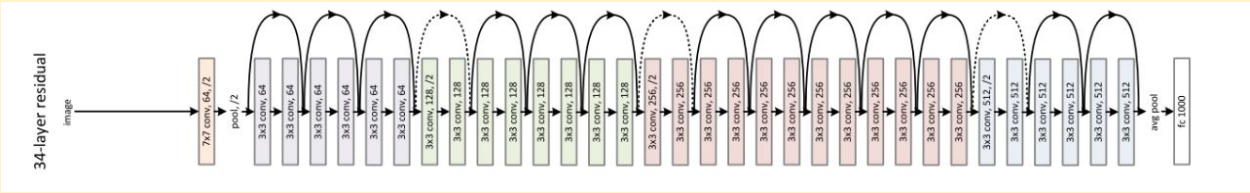
Training curves	Test Curves
	
Unexpectedly, the 56-layer had a higher train error than the 20-layer. It was expected to have overfit the training data, thereby having a lower training error. It was hypothesised that the gradients were not able to flow well through this deeper network.	Here, as predicted, the 56-layer performed worse than the 20-layer due to overfitting.

3. Now, when comparing the 19 and 34-layer networks, we see that at the very least, the 34-layer network should be able to match the performance of the 19-layer network.

4. Matching the performance could be done by bypassing the additional layers in the 34-layer network by using Identity Mapping.
5. Identity mapping refers to learning filter values such that the output is preserved identical across a finite number of layers, till it reaches the target layer. Basically cloning the layer output till required.



6. However, it wasn't able to match the 19-layer's error. This implies that the information from the input is getting highly morphed and by the time we reach the output, it is highly transformed.
7. A simple solution would be to keep passing the input information repeatedly in stages.
8. To attempt this, they tried the Residual Network or the ResNet



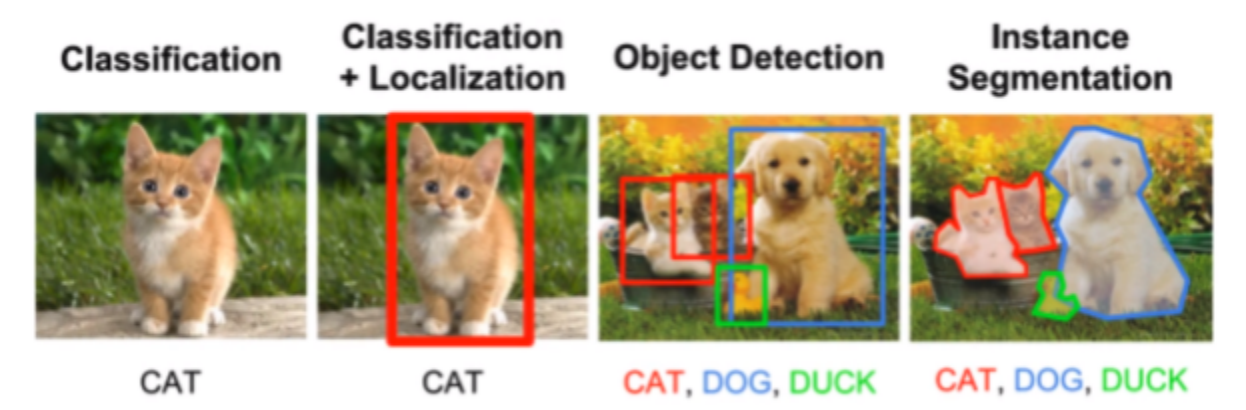
9. In the ResNet, **every two layers, we pass the input given to the first layer along with the output obtained at the second layer.**
   a. Input: x_1
   b. Output: $x_2 = f(x_1) + x_1$
   c. Output after two layers: $x_3 = f(x_2) + x_2$
10. This helped the gradients to flow back better and the training to improve
11. It is called a Residual Network because at every stage, there is a residue of the input which is passed once again with the output.
12. Using this technology, they were able to train very deep Neural networks of up to 151 layers.

PadhAI: CNN Architectures

One Fourth Labs

---

13. The ResNet showed remarkable performance among the various tasks



14. It was the winner among the 4 main tasks across the following datasets
    a. ImageNet Classification **(ResNet-151)**
    b. ImageNet Localization **(ResNet-101)**
    c. ImageNet Detection **(ResNet-101)**
    d. Coco Detection **(ResNet-101)**
    e. Coco Segmentation **(ResNet-101)**
15. Some of the popular ResNets are **(ResNet-51, ResNet-101 and ResNet-151)**

PadhAI: Visualising CNNs

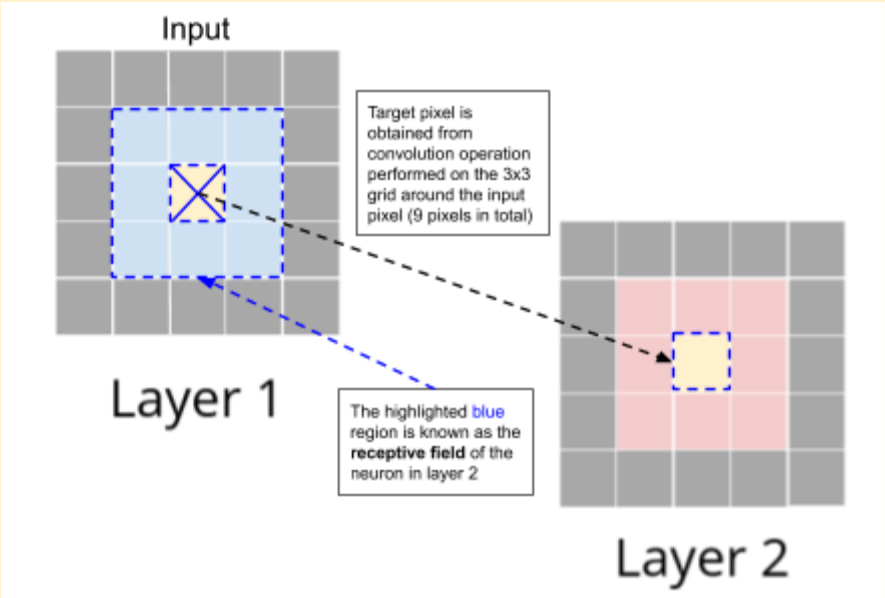
One Fourth Labs

Visualising CNNs

Receptive field of a neuron

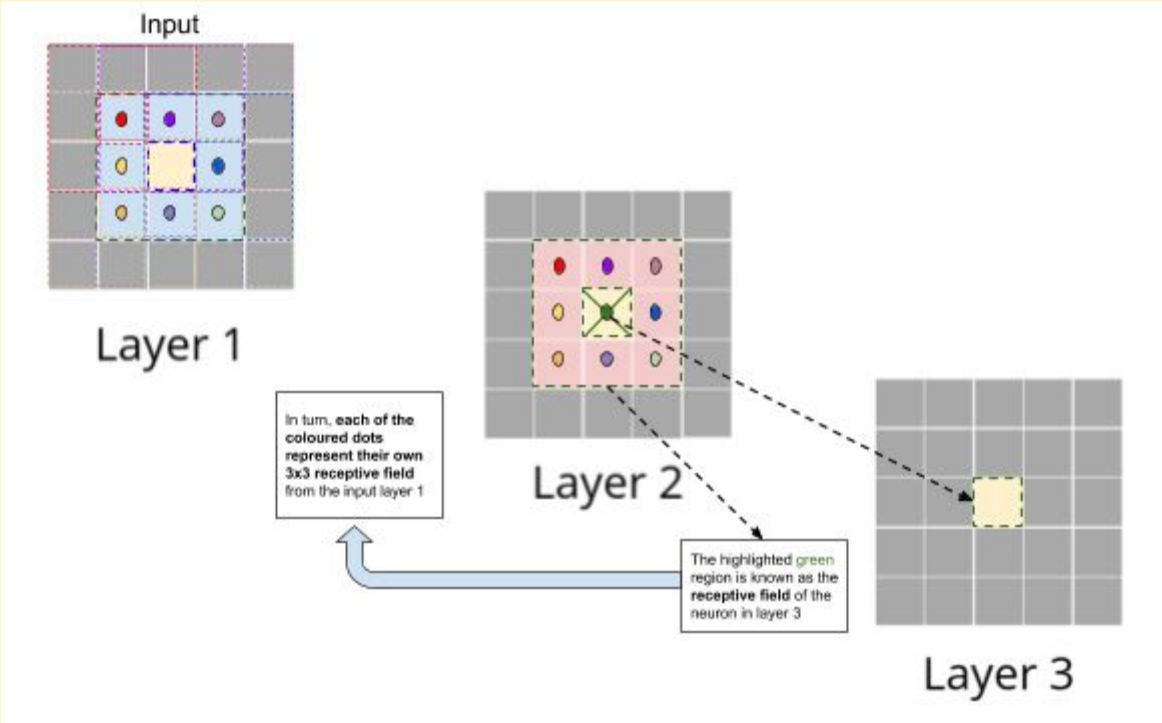
How does the receptive field increase across layers?

1. What does a filter learn? What kind of images cause certain neurons to fire? How good are the hidden representations?
2. To answer these questions, we need to talk about the receptive field.
3. Consider a 3 layered CNN, where in each of the layers, we apply a 3x3 filter

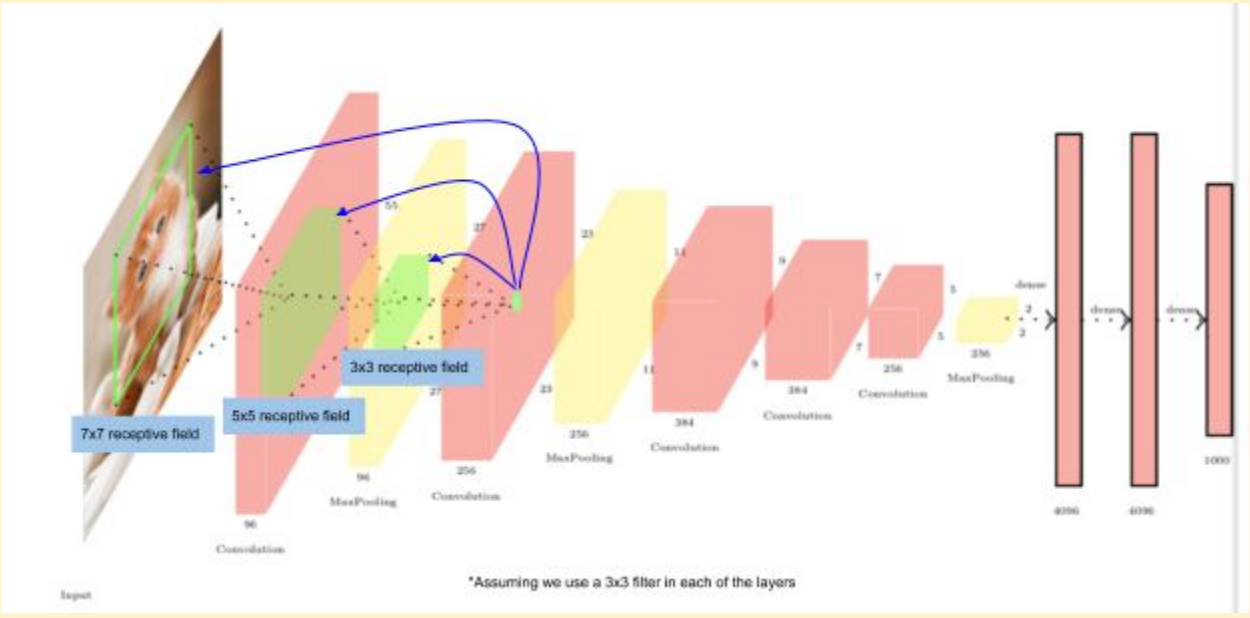


4. The **Receptive Field or Region of influence** of the neuron/pixel in a subsequent layer refers to all the neurons/pixels from the previous that were involved in the convolution operation to produce said output neuron/pixel.

5. Let's look at layer 3, and express the receptive field in terms of layer 1.



6. Thus, we can see that the 3x3 grid in the receptive field of the layer 3 pixel translates to a 5x5 grid receptive field in the input image.
7. We can see a sort of pattern, whereby if we have more layers, the receptive field becomes larger and larger in the input image.
8. If there was another Layer 4 with the same 3x3 filter, it would correspond to a 7x7 receptive field in the input layer.
9. The following diagram illustrates how the receptive field in the input layer is larger and larger as we move down the layers.



10. The increase in receptive field size through the layers increases with a larger filter size.

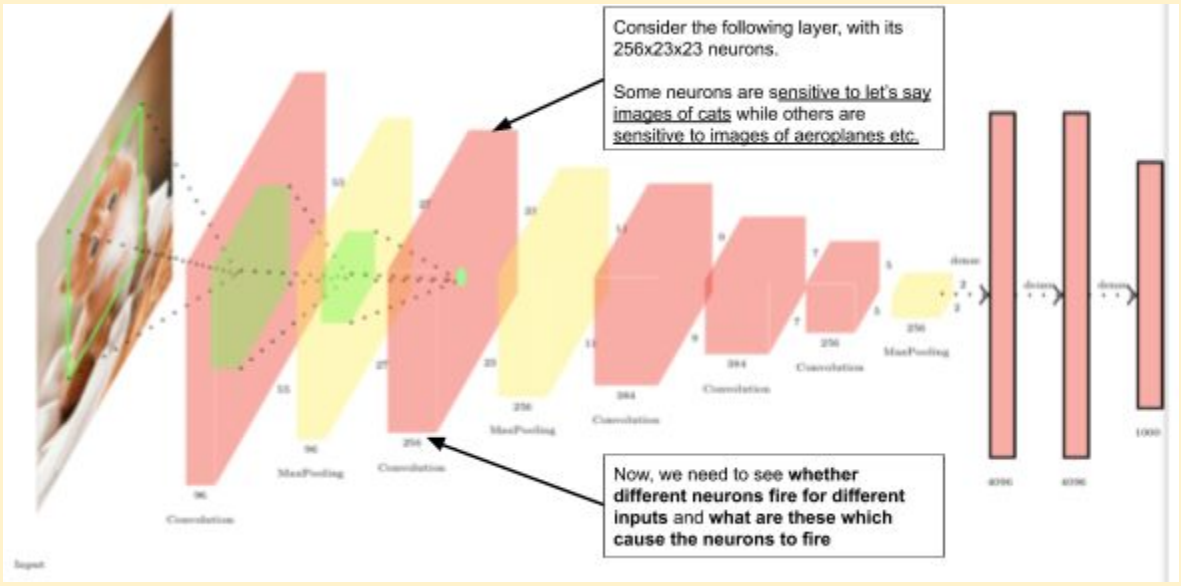
PadhAI: Visualising CNNs

One Fourth Labs

Identifying images which cause certain neurons to fire

How do you check what is causing a neuron to fire?

1. From the previous section, why is the concept of an increasing receptive field important?
2. Consider the same diagram from the before



3. Now, what does Firing mean? In the case of sigmoid/tanh neurons, it refers to when the output approaches 1. In the case of ReLU and leaky ReLU, it refers to any large positive value, greater than some threshold.
4. Here are the results of an experiment to understand the firing of different neurons

5. In the above image, the firing of 6 different neurons was observed across the training set of 1000 images. Each neuron is shown to be sensitive to particular inputs as shown in the figure above.
6. This analysis helps us to understand whether neurons in different layers are learning some meaningful patterns, and having some discriminatory power, instead of all of them firing for all types of inputs..

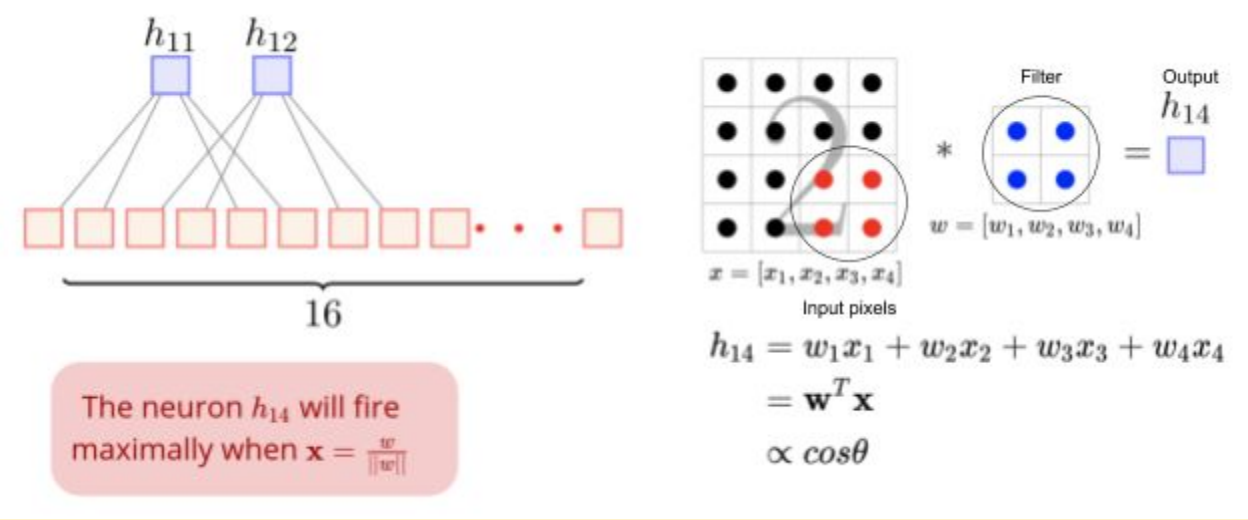
PadhAI: Visualising CNNs

One Fourth Labs

Visualising filters

What does a filter capture?

1. We have dealt with filters in all our CNN models so far. Now the question is, what exactly does a filter capture?
2. Let's look at the working of a 2x2 filter on a 4x4 input image



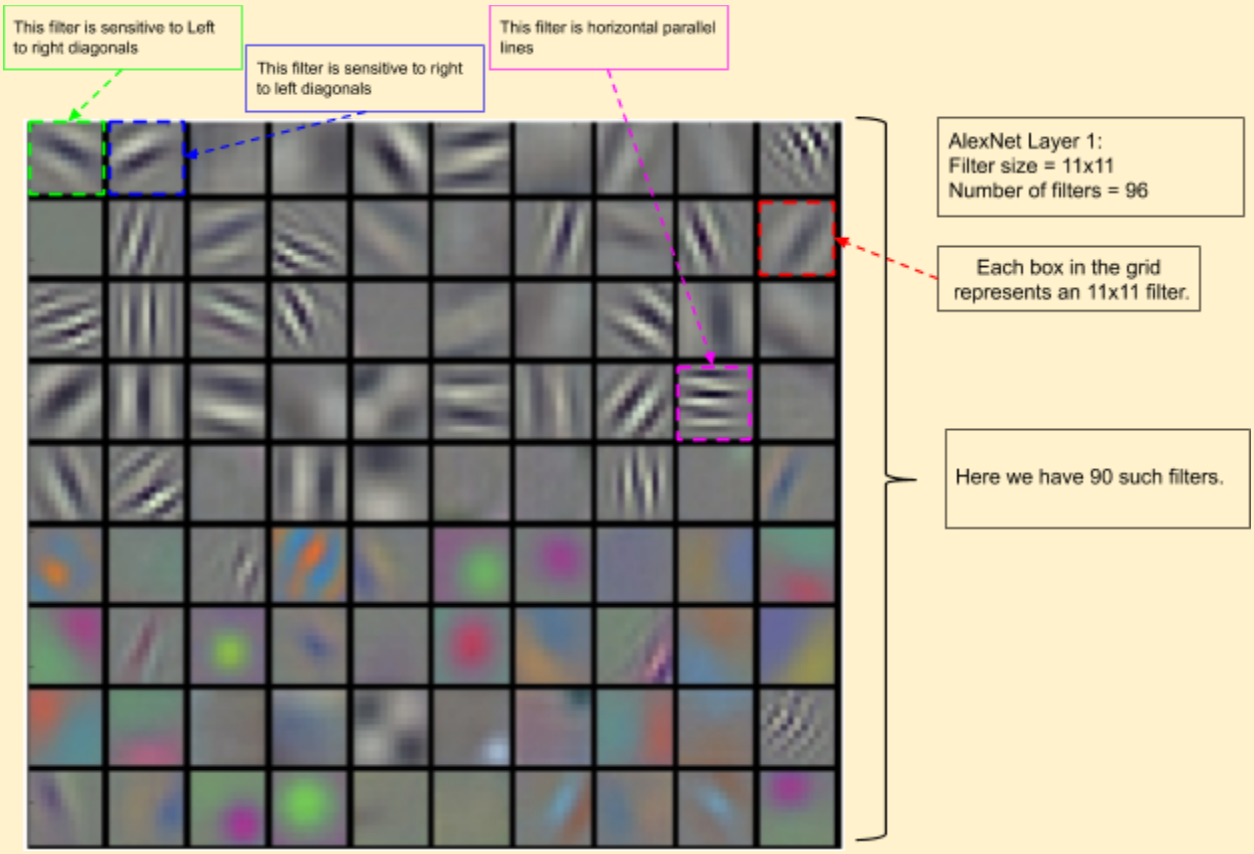
a. Here, the input image is 4x4 while the fiter is 2x2
b. The red input pixel vector $x = [x_1, x_2, x_3, x_4]$
c. The weight vector $w = [h_1, h_2, h_3, h_4]$
d. By convolving the input pixels with the filter, we get the output
e. Output $h_{14} = w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4$
f. $h_{14} = w^Tx$ (This is the same as the dot product between the two vectors)
g. $h_{14} \propto cos(\theta)$ $[where\ \theta\ is\ the\ angle\ between\ the\ two\ vectors]$ $cos(\theta) = \frac{w^Tx}{||w||\ ||x||}$
h. Now for certain inputs, we want the filter to fire (give a high value).
i. Now, h_{14} will be high when $cos(\theta)$ is high (cos(θ) = 1), i.e. when θ is 0. This implies the two vectors w and x are in the same direction.
j. So, we can say that an input vector which aligns with a filter vector yields maximum output.
k. The neuron h_{14} will fire maximally when $x = \frac{w}{||w||}$ (x is a unit vector in the direction of w)
l. Thus, when we **slide the 2x2 filter w** across the 4x4 input region, whenever we **reach a 2x2 region x** that looks exactly like the filter, we get a high output. For all other regions which do not align with the filter, the output is low.

PadhAI: Visualising CNNs

One Fourth Labs

---

3. Now, let us visualize the filters in AlexNet



   a. The above image shows us how different patterns are identified by different filters.

PadhAI: Visualising CNNs

One Fourth Labs

Occlusion experiments

Which patches in the image contribute maximally to the output?

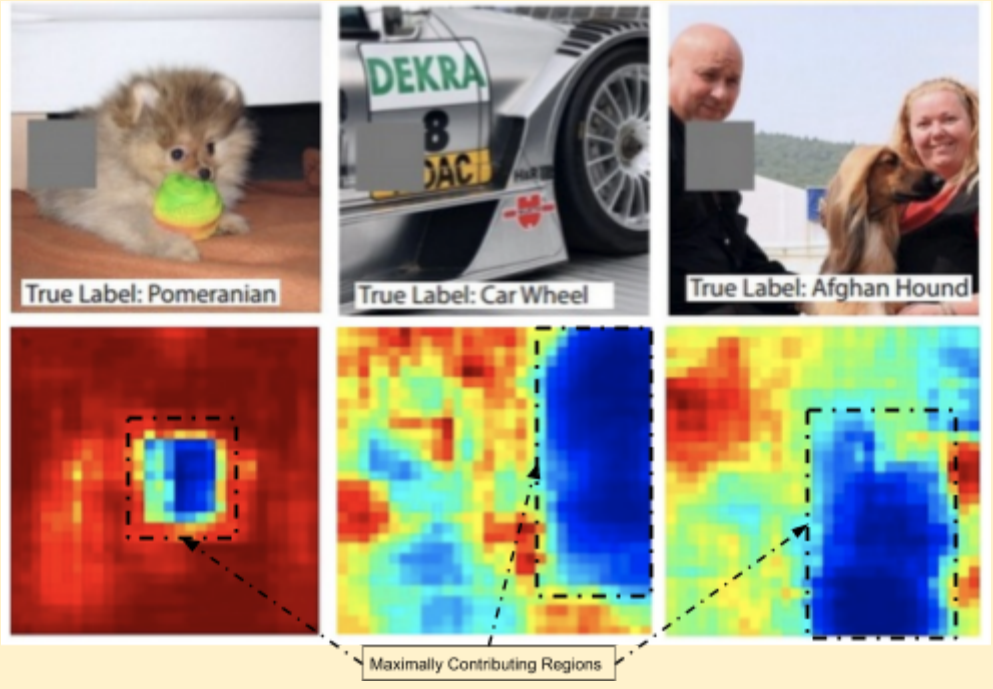
1. We can determine the maximally contributing regions in an image using occlusion experiments
2. Occlusion experiments refer to applying occlusions to the input image and seeing how they alter the predicted output distribution.
3. Then we take the difference between the un-occluded distribution and occluded distributions for occlusions placed in all patches of the images.
4. These differences allow us to generate a heatmap of maximally contributing patches.
5. Consider the following images with occlusions applied to them

   a. The true distribution for any of these images would be say $y = [... 1 ...]$
   b. The un-occluded predicted distribution $\hat{y}_1 = [.... 0.89 ....]$
   c. Occlusion placed at position 1 $\hat{y}_2 = [.... 0.84 ....]$

   ...

   d. Occlusion placed at position L $\hat{y}_n = [.... 0.3 ....]$

6. Now, calculating the difference between $(\hat{y}_1 and \sum_{n=2}^{L} y_n)$, we get differences in probilites which can be plotted as a heatmap over the input image with colder regions being more highly contributing.



---

7. Thus, the **higher the difference between un-occluded and occluded distributions**, the **more significant the contribution of that particular occluded region** to the output.