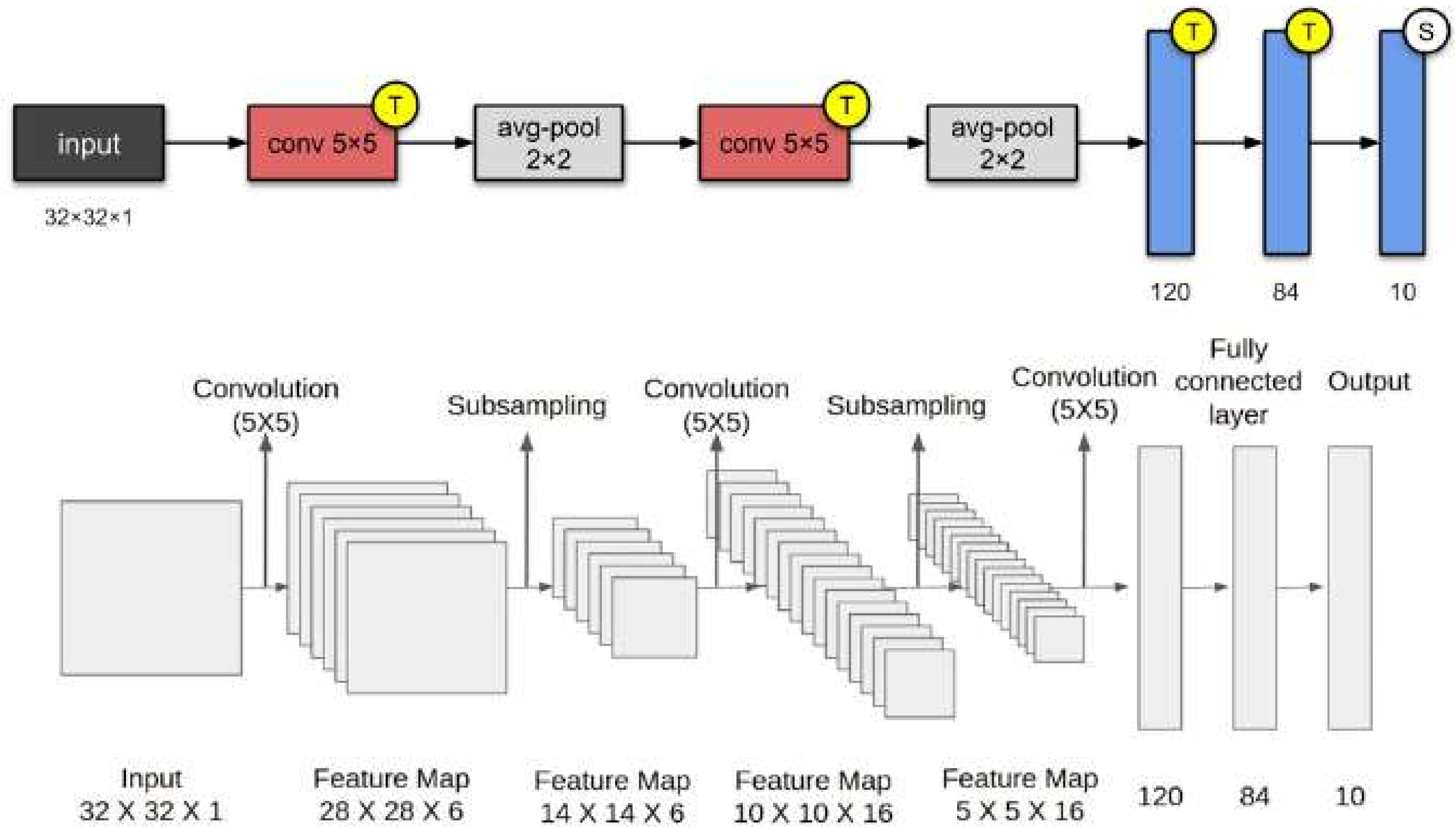
Different CNN Architectures

Sourav Karmakar

souravkarmakar29@gmail.com

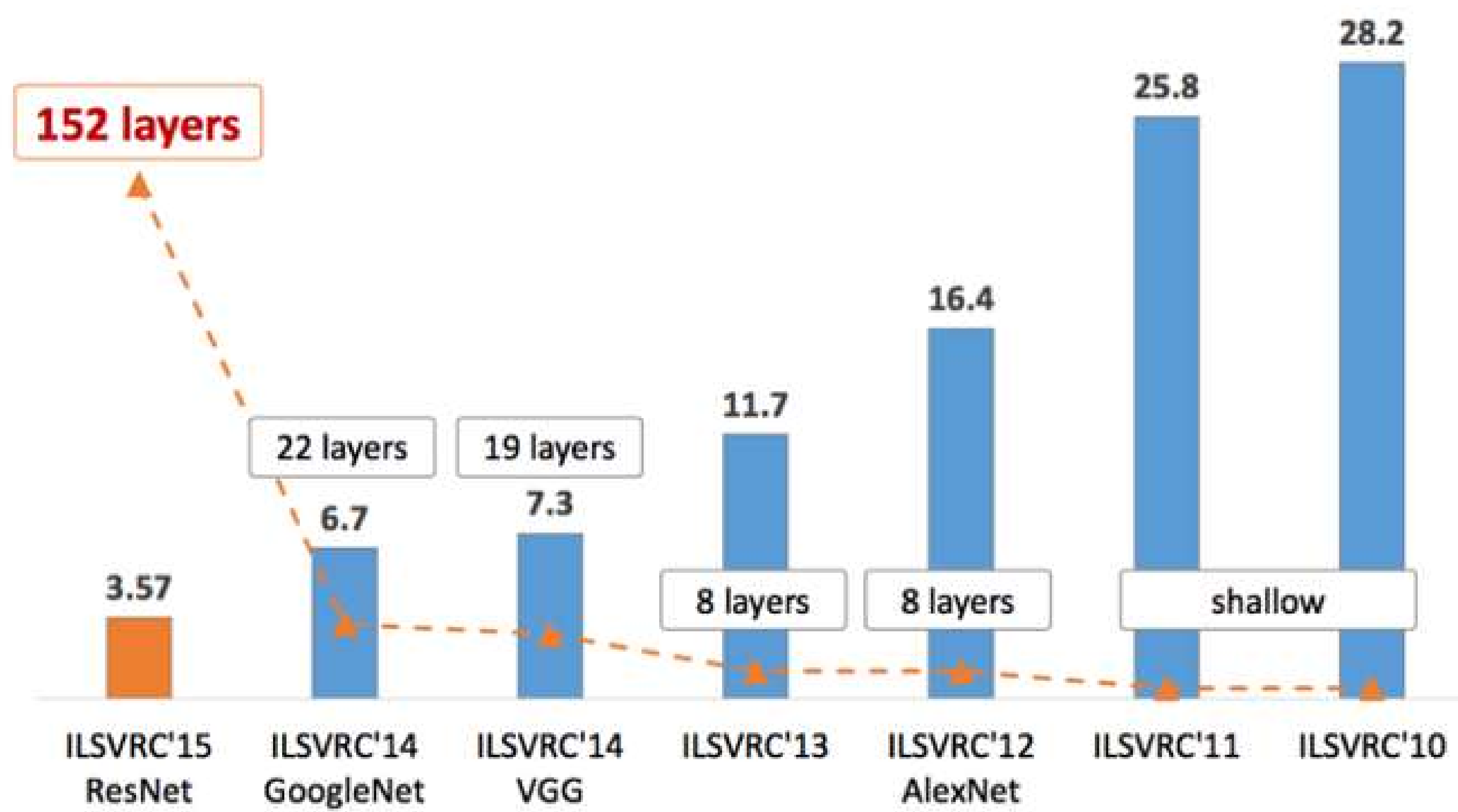
LeNet-5 (1998)



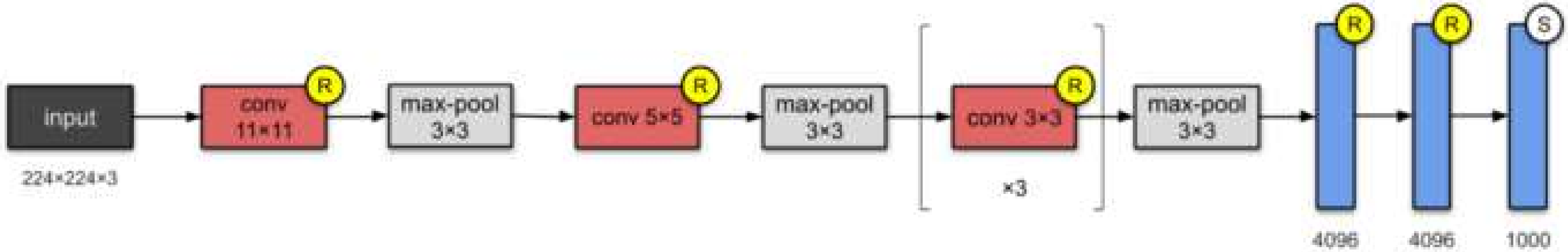
- LeNet-5 is one of the simplest architectures. It has 2 convolutional and 3 fully-connected layers (hence “5” — it is very common for the names of neural networks to be derived from the number of *convolutional* and *fully connected* layers that they have). The average-pooling layer as we know it now was called a *sub-sampling layer* and it had trainable weights (which isn’t the current practice of designing CNNs nowadays).
- Very efficient in recognizing handwritten digits.

IMAGENET Challenge

- ImageNet Large Scale Visual Recognition Challenge (ILSVRC) was an annual computer vision competition developed upon a subset of a publicly available computer vision dataset called **ImageNet**. As such, the tasks and even the challenge itself is often referred to as the **ImageNet Competition**.
- The ImageNet competitions were hosted from 2010 to 2016 and very interesting and useful CNN architectures were developed by industry and academia for this competition. Those architectures are still widely used.

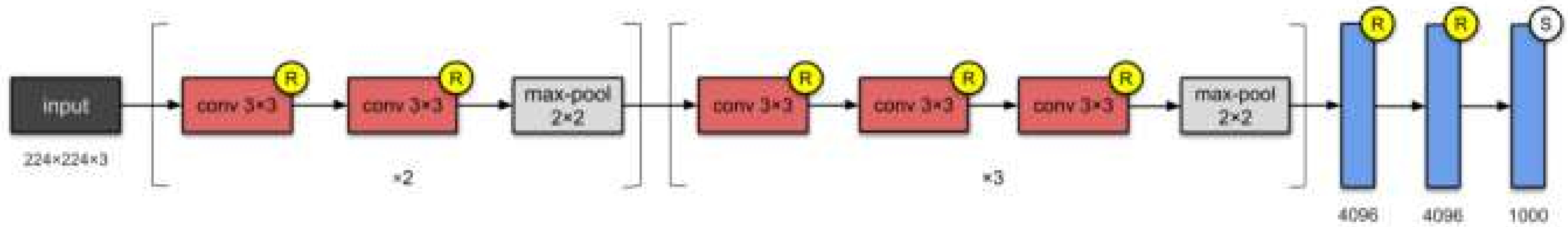


AlexNet (2012)



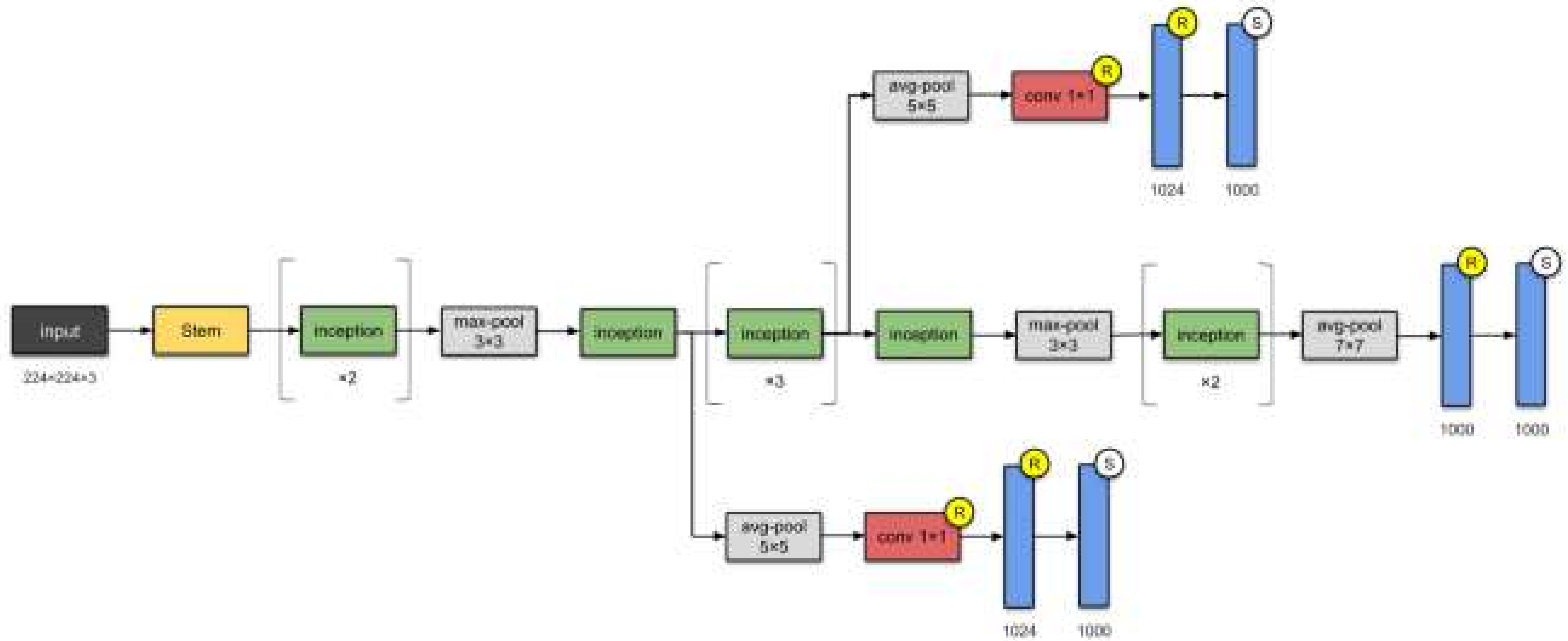
- With **60M parameters**, AlexNet has 8 layers — 5 convolutional and 3 fully connected. AlexNet just stacked a few more layers onto LeNet-5. At the point of publication, the authors pointed out that their architecture was "one of the largest convolutional neural networks to date on the subsets of ImageNet."
- They were the first to implement Rectified Linear Units (ReLUs) as activation functions.
- They also implemented Dropout for handling overfitting in large Neural Networks.
- Winner of ILSVRC – 2012.

VGG-16 (2014)

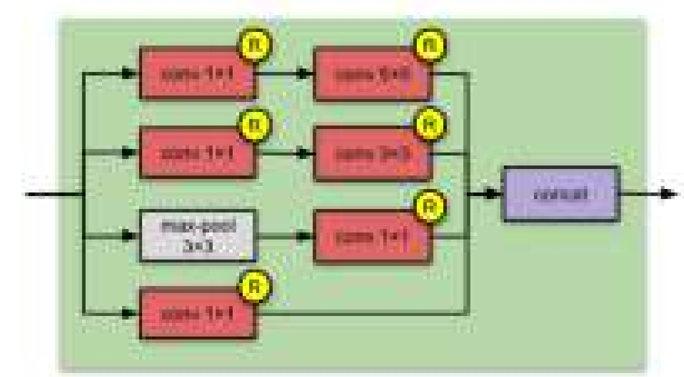


- This was developed at Visual Geometry Group (VGG) of Oxford University. VGG-16 has 13 convolutional and 3 fully-connected layers, carrying with them the ReLU tradition from AlexNet.
- It consists of 138M parameters and takes about 500MB storage Space.
- It is used as backbone architecture for feature extraction for various Computer Vision related tasks.

Inception-v1 (2014)



Stem



Inception module

Inception-v1 (2014)

- This 22-layer architecture with **5M** parameters is called the Inception-v1. Here, the **Network In Network** approach is heavily used, as mentioned in the paper. The Network In Network is implemented via ***Inception modules***. Each module presents 3 ideas:

  1. Having **parallel towers** of convolutions with different filters, followed by concatenation, captures different features at 1×1, 3×3 and 5×5, thereby 'concatinating' them.

  2. 1×1 convolutions are used for dimensionality reduction to remove computational bottlenecks.

  3. Due to the activation function from 1×1 convolution, its addition also adds nonlinearity.

Challenges of training Large CNN

- → As there are large number of parameters, the CNN tends to overfit for a relatively small datasets.
- → It will take longer time to train.

Remedy :-

→ To tackle overfitting we can use -

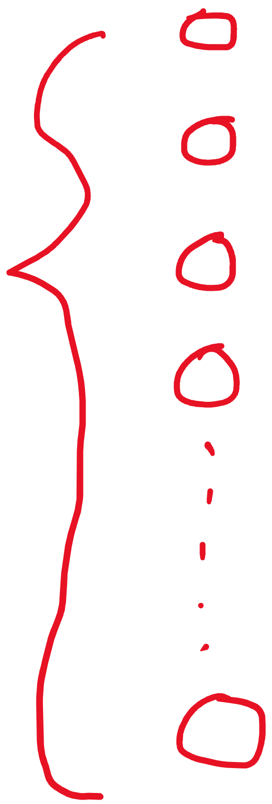
(1) Dropout layers.

(2) Batch-Normalisation

→ We can use transfer learning where we shall use some pretrained network for feature extraction & use custom classification layer on top of it.

Dropout:-

Large no of nodes.

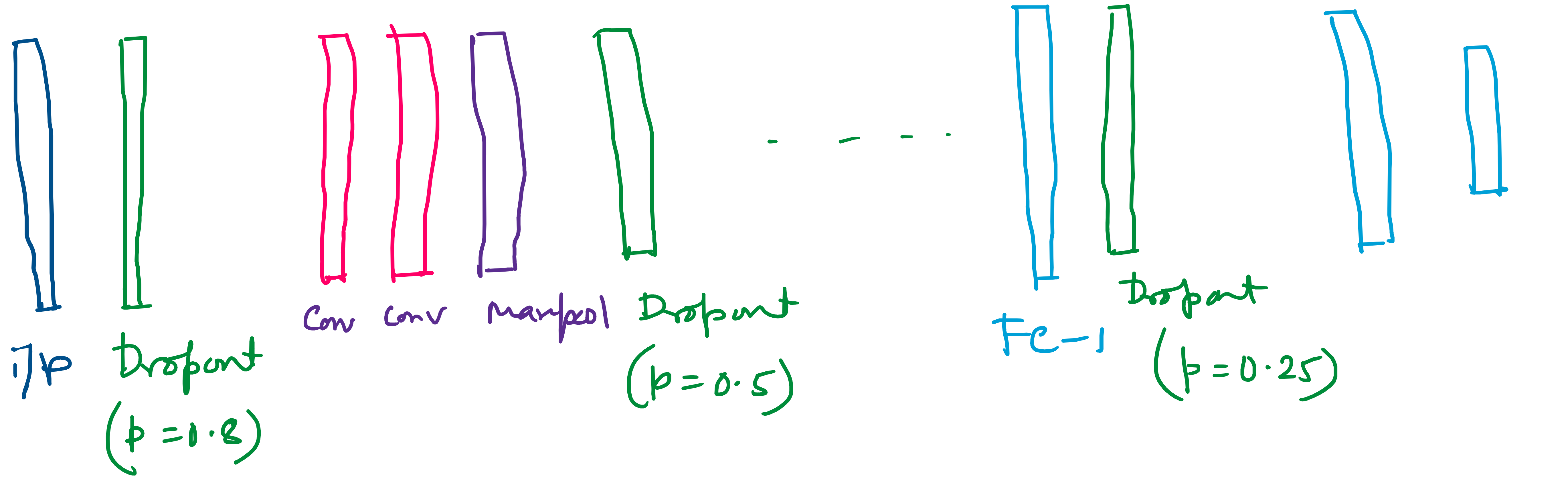


In dropout we shall train a 'p' fraction (p is known as dropout rate) of nodes while training the neural net.

This fraction of nodes are choosen randomly.

During inference we will use all the nodes.

Layer (1000 nodes) $p = 0.7 \rightarrow$ 700 nodes (randomly selected)

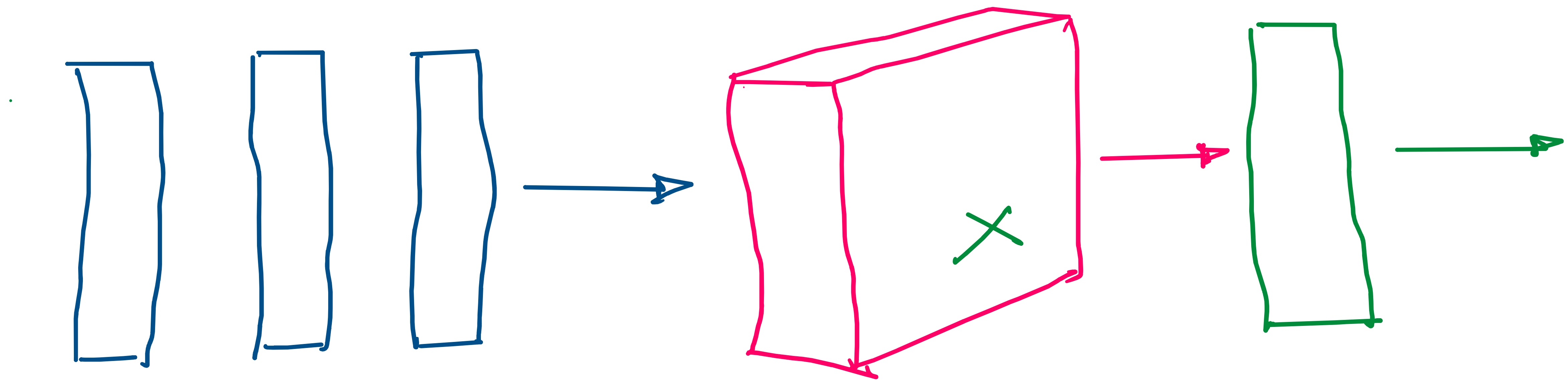


Dropout rate is high in i/p layer.

That is how we can modify neural network architecture to add dropout layers in between so that the NN generalizes well & prevents overfitting.

Batch Normalization :-

Another way of preventing overfitting in NNs. (Don't use Batchnorm & Dropout together).



Conv. Conv. Maxpool. o/p volume. Batchnorm

o/p is normalized

(notscaled & are skewed).

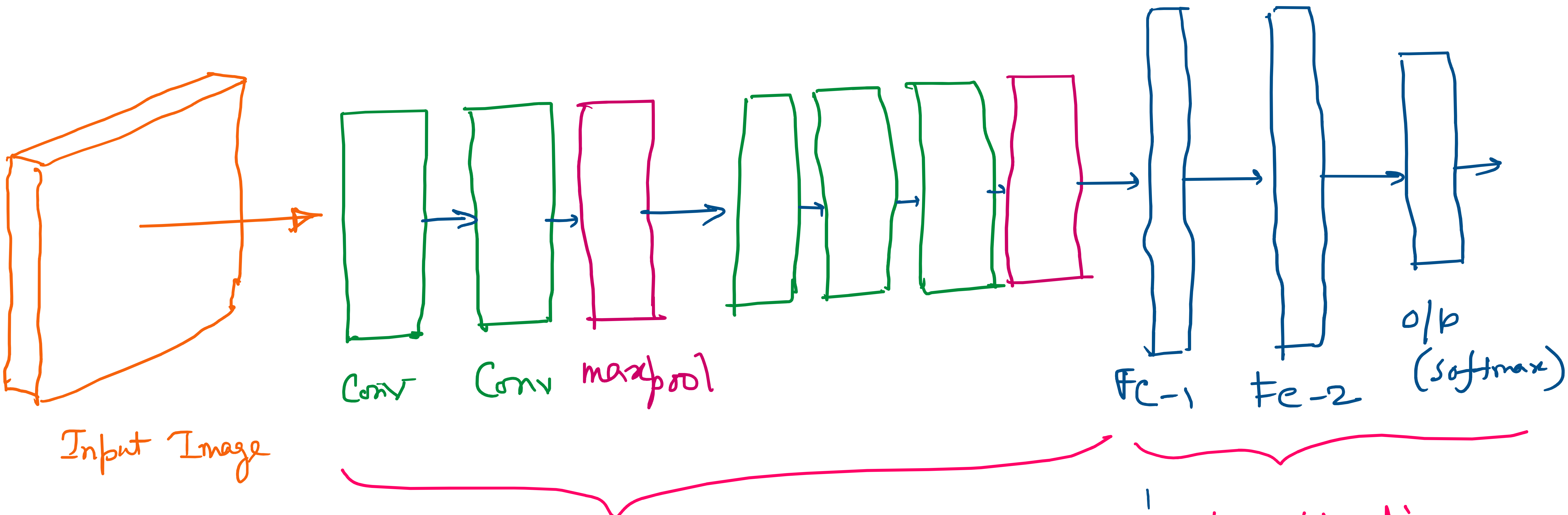
Batchnorm

↓

$$\frac{X - \mu}{\sigma(X)}$$

Transfer Learning:-

CNN architecture (General)



Feature Extractor

The feature Extractor part extracts useful feature from images (like edges, patches etc.)

classification.

This classification part take those features & classifies them

Steps for transfer learning

1. Take a pretrained neural network.

   This neural network is trained on large datset already & the learnt weights are available.

2. Freeze the weights of feature Extractor part of NN.

   During training on our dataset we will not modify the weights of this part of NN.

3. Add custom classification layer on top of it & train them on our own dataset.

4. Use the entire network for inference.

NN is pretrained on large dataset (like ImageNet)

(10 M parameters) trainable.

Feature Extractor.

(90M parameters) are non-trainable.

o/p

Fc-1 Fc-2

our own image dataset.

Freeze the weights of this part of NN

This custom classification layers are added with random weight initialization.

→ Train (fine-tuning) the NN on our own dataset where the weights of last few layers will be trained.

Why is it called transfer learning?

→ The original NN was trained to classify some images.

→ We are using the parameters learnt by the original NN & use that to classify our own dataset by applying fine-tuning techniques.

→ We are transfering the knowledge of classifying one set of images to classify another set of image.

→ That is why it is called transfer learning.