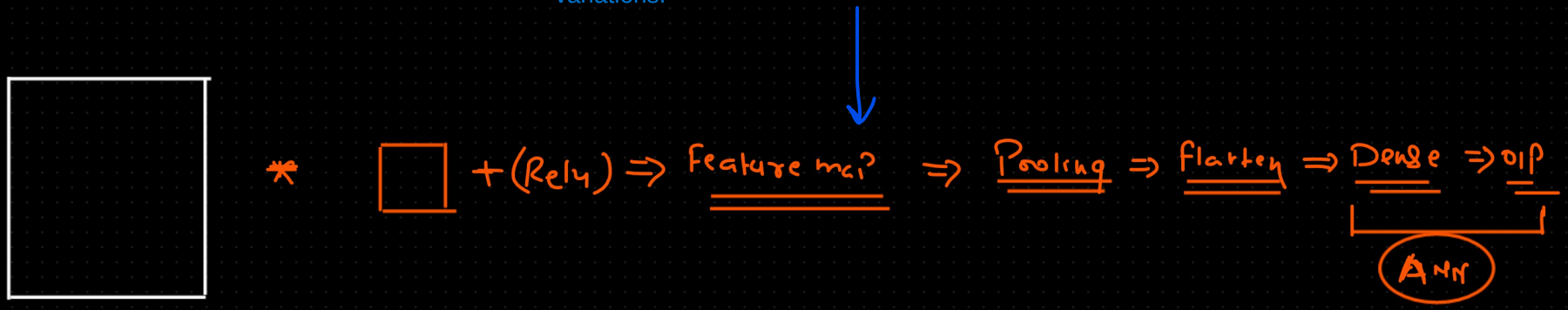
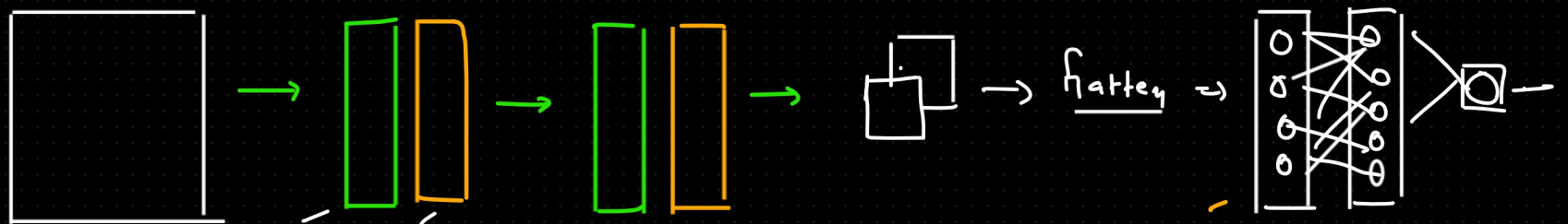
Agenda:
Pre trained CNN architecture and Fine Tuning them for different use cases

CNN Architecture

Overall hierarchy of the CNN. So, whatever CNN architecture we are going to create or learn (pre defined one) will follow the same hierarchy with some little variations.





This section of CNN architecture formed after removing the ANN FC part is called the Convolution Base

{Conv, filters, Stride, Padding, Pooling} Activation, optimizer, Layer, Model, DO, BN

Experimental | Research | Project

1. LENET
2. Alexnet
3. VGG 16, 19
4. Googlenet
5. Resnet

14 million ⇒ 1.4 cr ⇒ 20,000

IMAGENET ⇒ Deep Learning

(ILSVRC)

Large Scale Visual Recognition challenge

- Over the time as part of Experiment/Research/Project different CNN architectures came into picture.

- LENET was trained over MNIST dataset

- All other CNN architectures are pre trained models for CNN. And the training was done over IMAGENET dataset which is very important dataset in deep learning

- IMAGENET dataset consists on 14 million or 1.4 crore images distributed over 20,000 classes

- ILSVRC is the competition where using IMAGENET dataset, people create different CNN architectures. And the one with best accuracy wins the competition. This competition over the years gave birth to different CNN architectures

- LENET was made for the US Navy, where Navy wishes recognize digits that are flashed over some machine in an automated fashion without human intervention

- ILSVRC stands for ImageNet Large Scale Visual Recognition Challenge

- Main idea behind creating IMAGENET dataset was to make a more general image dataset where we have all the different kind of images under single umbrella

LENET / LENET-5

MNIST

0, 1, 2 ... 9

1989 ⇒ Yann LeCun

1998 ⇒ Lenet

CNN ⇒ Learning (filter) (BP)

FP ⇔ BP

US-Navy ⇒ Machine ⇒ Number

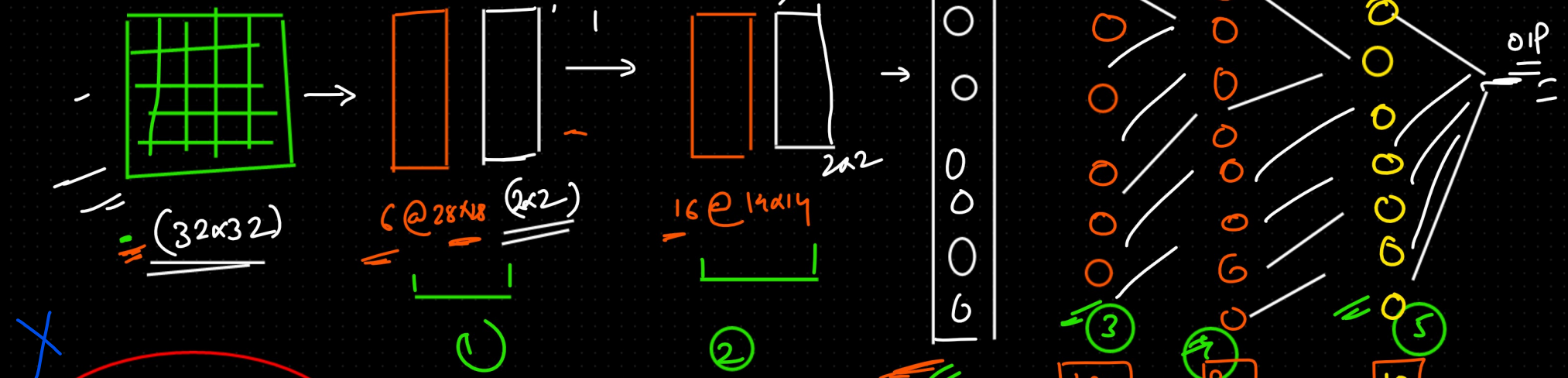
1. SIFT
2. HOG

→ pdf → openCV

Feature extraction

CNN

LENET-5 ⇒ tanh v/s Sigmoid.



Assignment

Tf.keras ⇒ Architecture

Sigmoid

Level

Image ⇒ ⇒ Level

Parameter ?

Alexnet ⇒ 2012

Why Pretrained

1. time consuming → training (10000) (100000) ⇒ Deep network
2. resource consuming
3. cost expensive.
4. Data ⇒ huge ⇒ well manner

2 - 20 mins

Week

?

Data hungry

100000 , 1000000 ⇒ training

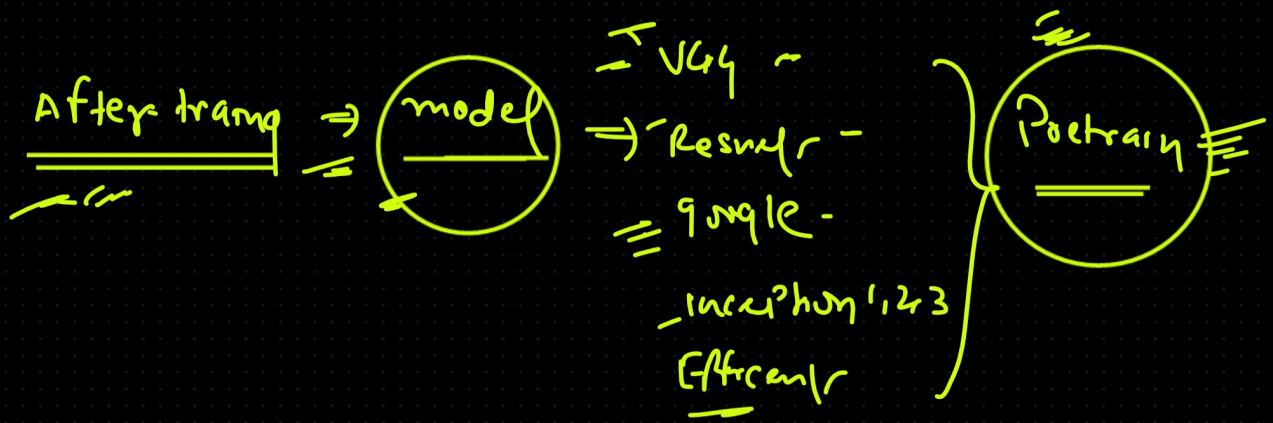
Scrape , Arrange , Label

training ⇒ why easy ⇒ Data (clean, label)

↳ resource

Ques: Why do we need pre trained models?
Ans: Actually these CNN models are very huge in terms of learnable parameters. One can have parameters in billion or trillions. To train such models from scratch will require a lot of computation, cost, memory and other resources. So, to overcome High Computation time, memory, resources and cost expensive we are using pre trained model and performing fine tuning on top of it based on our use case

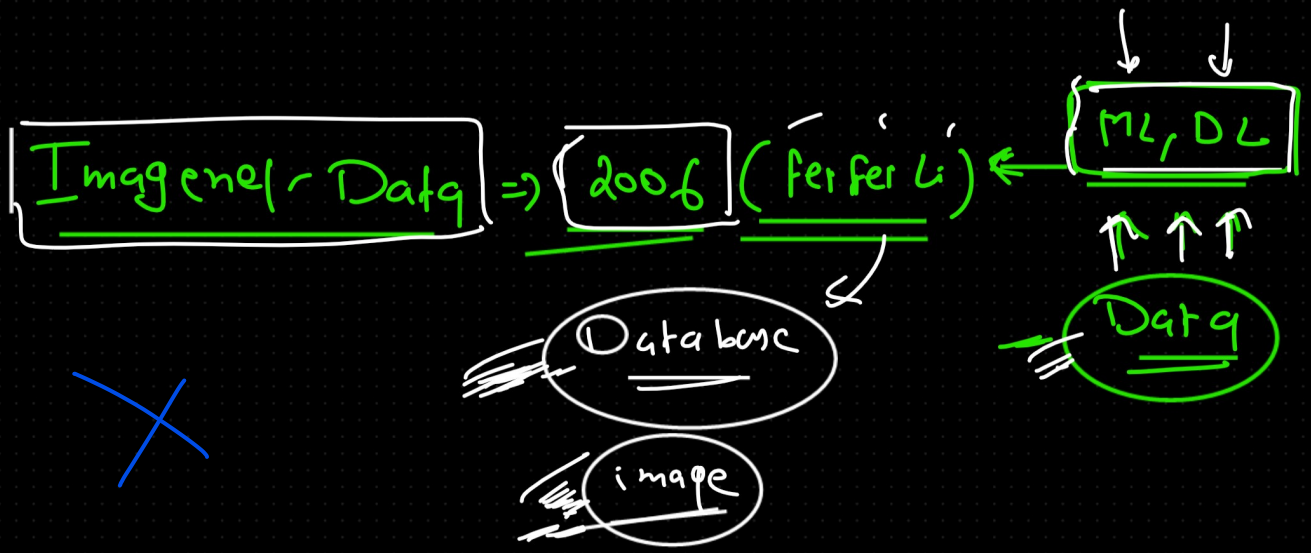
See development is not a big challenge but productionizing such a huge models is very huge challenge. For Eg; ChatGPT are still going in looses since they are paying huge amount of money daily to basically keep up their GPT model for public use



TIP:
- Google Keras datasets for getting the image based datasets that can be basically used for the learning and experimenting purpose on top different CNN architectures or our self CNN architectures.

- Google keras pre-trained model to get the list of pre-train CNN models available for use

Which Data ?



Database => 14 million images => 20,000 category

1.4 color

Dog
- GR
- CGLRG
- PUG
- RB

Proper manner

Imagenet → 2009

Real Data subset

{10 million, 1000 class}

ILSVRC

2010

Flower
- Rose
-

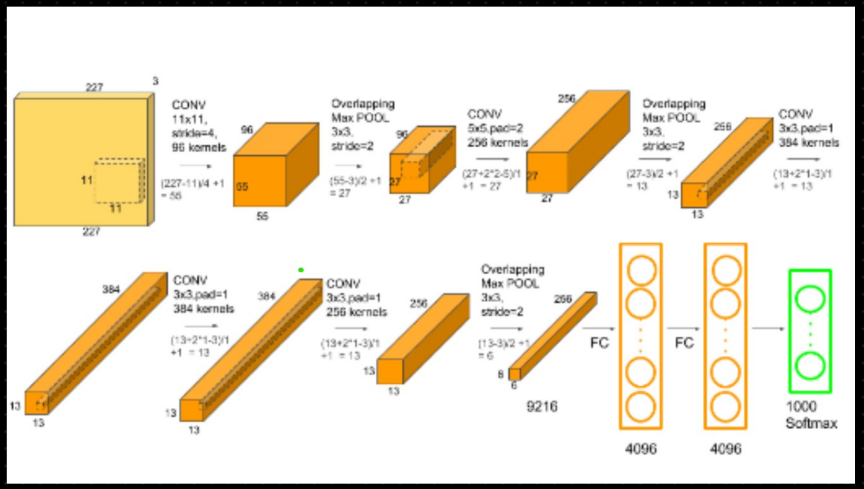
2010 ⇒ ML (SIFT, HOG) ⇒ 28-30 %. Loss

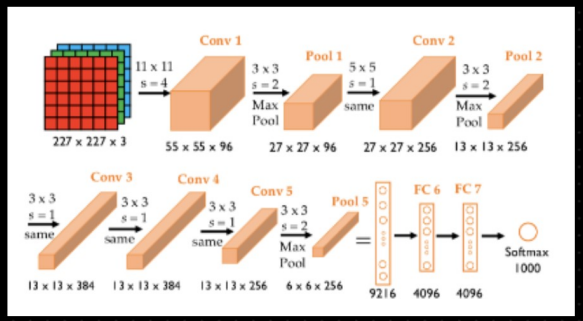
2011 ⇒ 25 %

2012 ⇒ Jeffrey Hinton → Alexnet ⇒ 16 %.

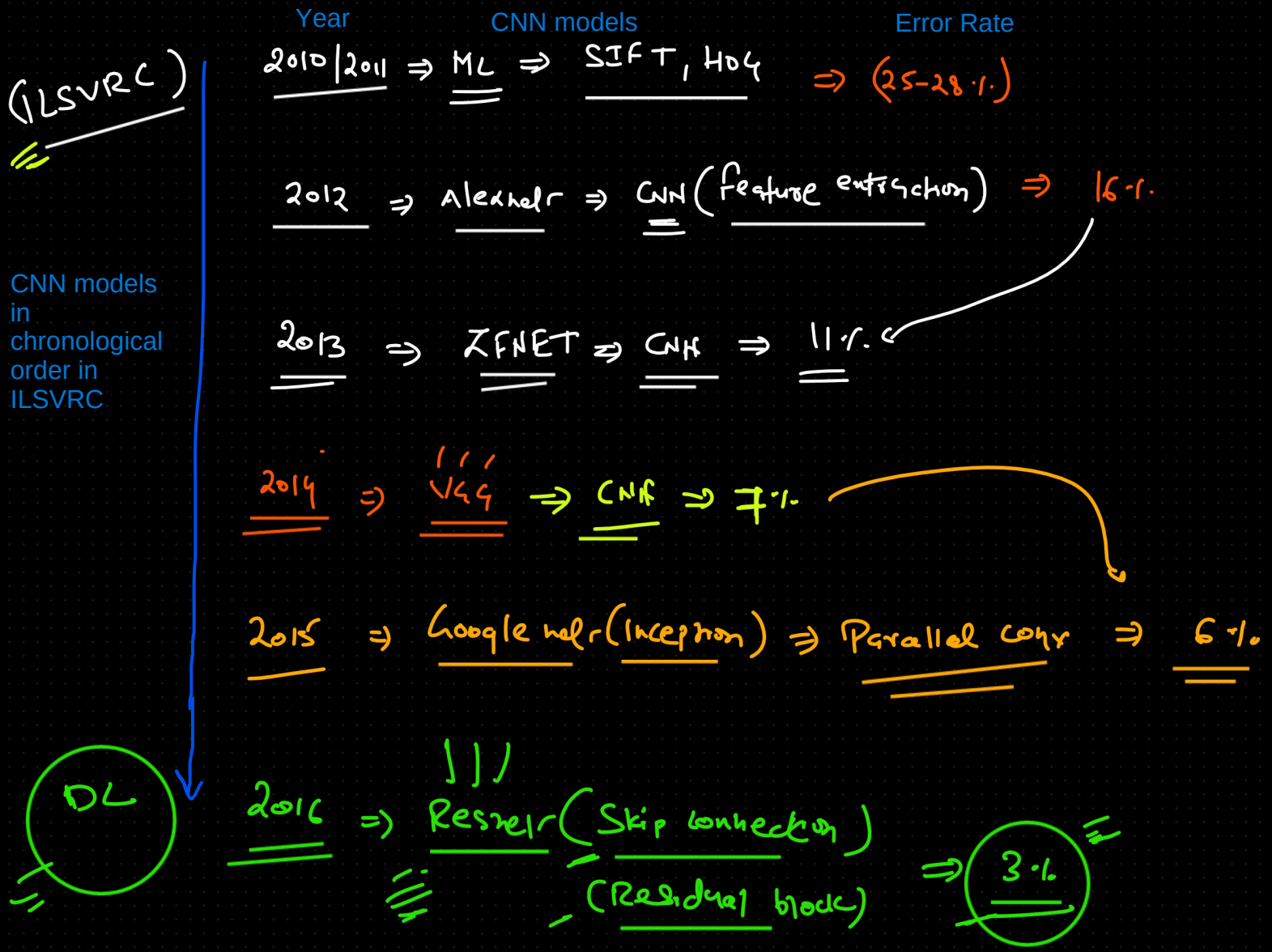
Alexnet
- GPU
- Relu
- Deep Architecture

16 %. error rate





Lener, Alexnet


Year
CNN models
Error Rate
(ILSVRC)
2010|2011 ⇒ ML ⇒ SIFT, HOG ⇒ (25-28 %)
2012 ⇒ Alexnet ⇒ CNN (feature extraction) ⇒ 16 %
CNN models in chronological order in ILSVRC
2013 ⇒ ZFNET ⇒ CNN ⇒ 11 %
2014 ⇒ VGG ⇒ CNN ⇒ 7 %
2015 ⇒ Google net (Inception) ⇒ Parallel conv ⇒ 6 %
DL
2016 ⇒ Resnet (Skip connection)
(Residual block)
⇒ 3 %

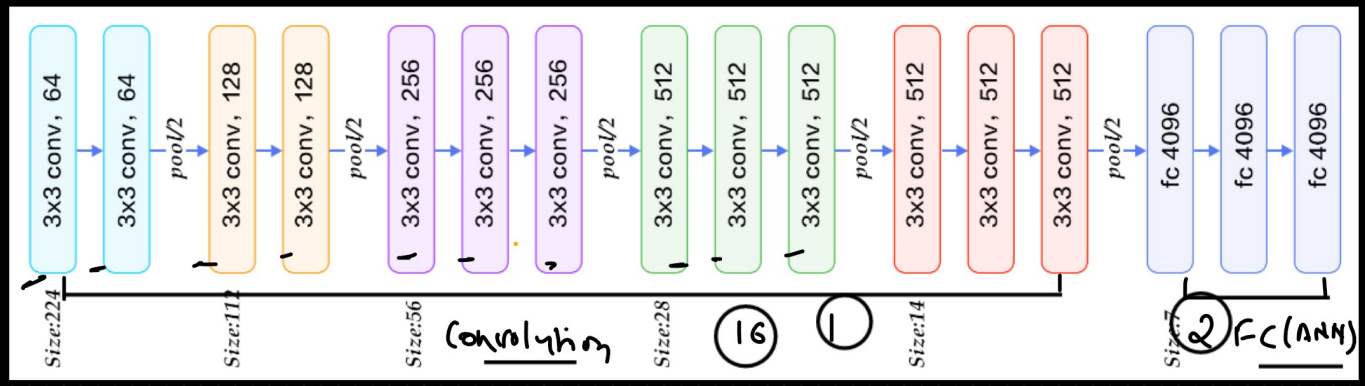
Transfer Learning

Transfer Learning is a part of Fine tuning.

https://dev.to/luxacademy/understanding-the-differences-fine-tuning-vs-transfer-learning-370

CNN ⇒ training → Data (clean, label, arrang) → cost

training → high resources



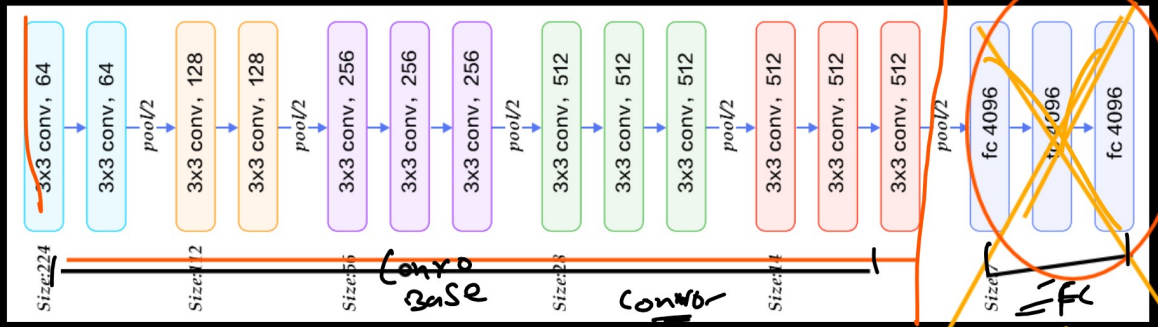
PhD → Ayz → IIT

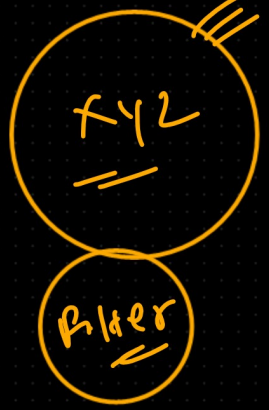
Data → Imagenet ⇒ 1000, 1 million ⇒ CNN Architecture (V44)

- Dog
- car
- lion
- man
- women
- Ball
- cricket
- chair

Transfer learning ⇒

1. Transfer
2. Learning



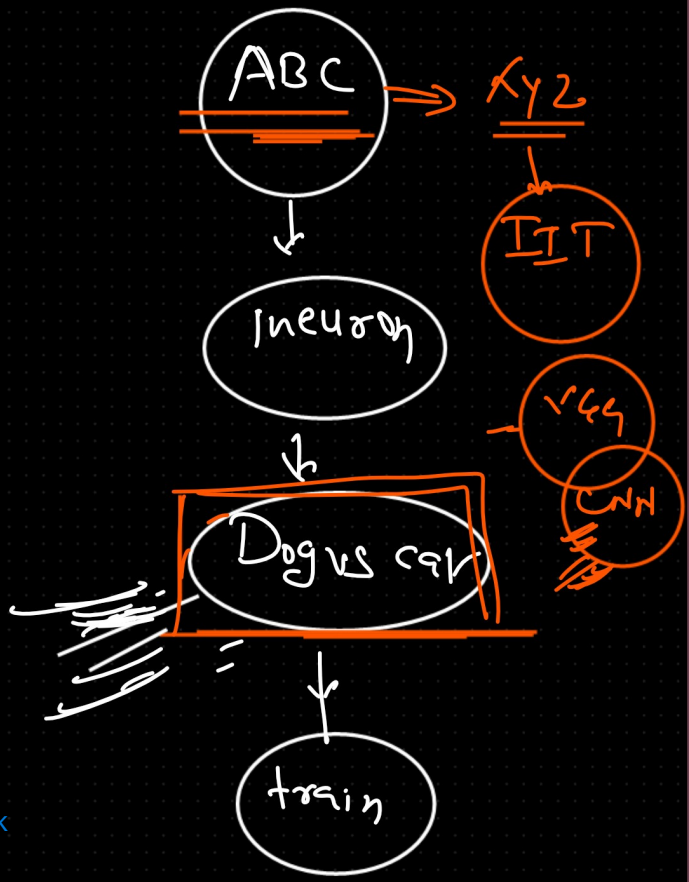


ConvoBase + myfc

transfer

Fully connected neural network

H_1	H_2	OIP
10	20	1



Transfer Learning

Transfer Learning ⇒

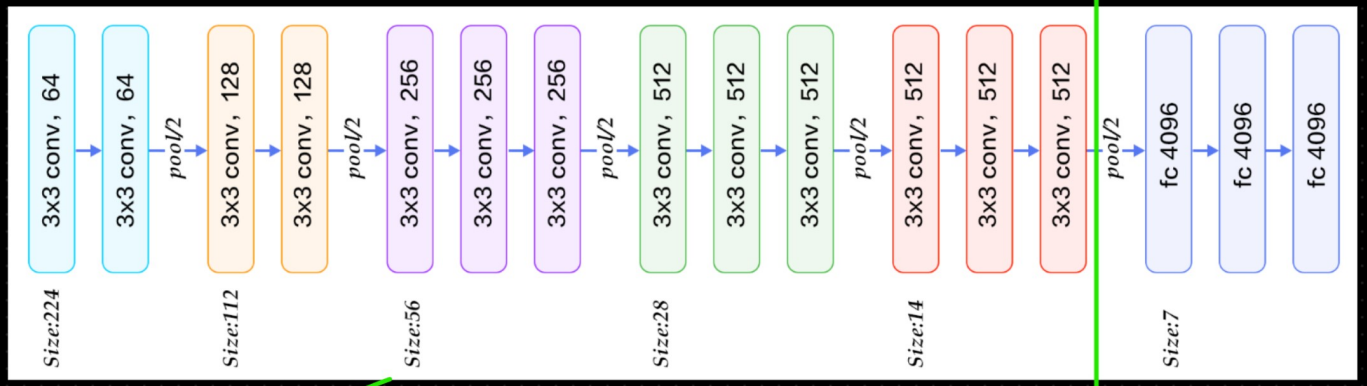
1. we extract the base Convolution network — Whose weights are frozen
2. then we connect our own fully connected layer. — Whose weights are allowed to train

Learning

Fine tuning

Whereas in Fine tuning we go one step further where we un-freeze some of the layers in the convoluted base as well. So, here we will be performing learning of parameters from scratch for both newly added fully connected ANN and parameters of few convolution base layers which we un-frozen

Conv base | Fully Connected



Imagenet
- 13
- 14

major miss lead

Weight ⇒ 15

CNN

Iphone 13, 14, 15

Imagenet

↳ 14

↳ 13

↳ 15 X

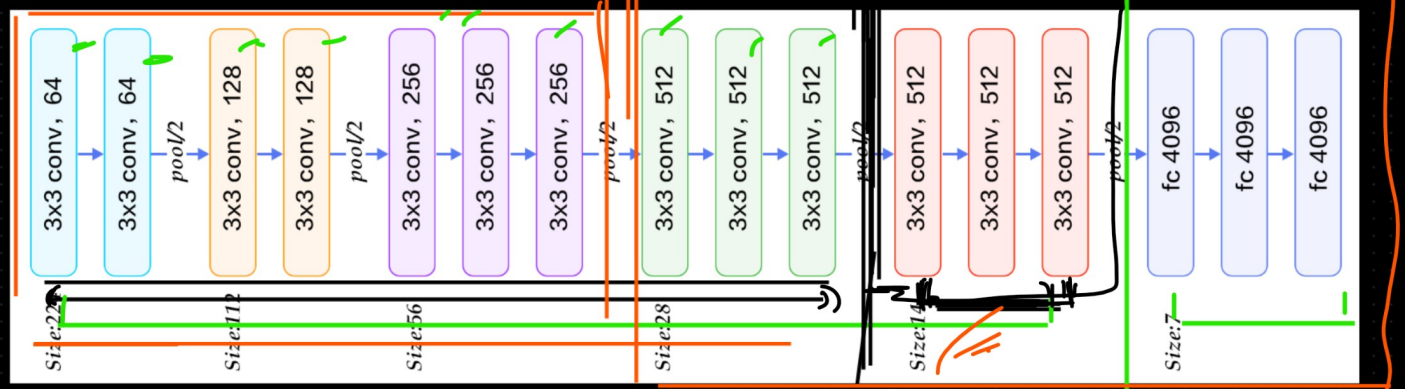
⇒ Convbase (Vgg) ← Image

finetune

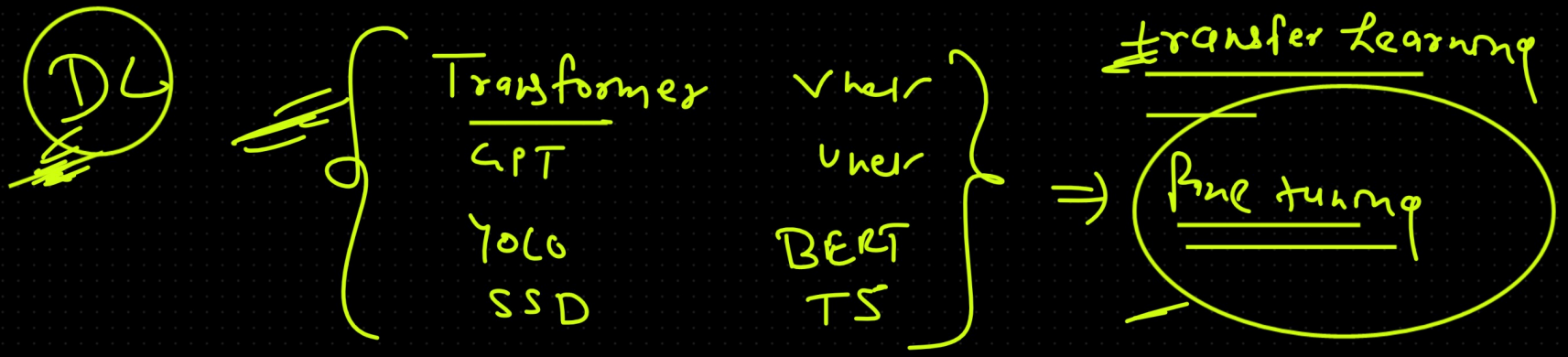
CNN ⇒ I 13, 14, 15

D, C, Alley

Vgg → image → 1000



fine tuning =



tensorspace.org is a beautiful java script enabled website where we can actually visualize all of these different CNN architectures

Key Differences between Fine-Tuning and Transfer Learning
Now that we have explored the implementation of both fine-tuning and transfer learning, let's summarize the key differences between the two techniques:

Training Approach: In transfer learning, we freeze all the pre-trained layers and only train the new layers added on top. In fine-tuning, we unfreeze some of the pre-trained layers and allow them to be updated during training.
Domain Similarity: Transfer learning is suitable when the new task or domain is somewhat similar to the original task or domain on which the pre-trained model was trained. Fine-tuning is more effective when the new dataset is large enough and closely related to the original dataset.
Computational Resources: Transfer learning requires fewer computational resources since only the new layers are trained. Fine-tuning, on the other hand, may require more resources, especially if we unfreeze and update a significant number of pre-trained layers.
Training Time: Transfer learning generally requires less training time since we are training fewer parameters. Fine-tuning may take longer, especially if we are updating a larger number of pre-trained layers.
Dataset Size: Transfer learning is effective when the new dataset is small, as it leverages the pre-trained model's knowledge on a large dataset. Fine-tuning is more suitable for larger datasets, as it allows the model to learn more specific features related to the new task.
It's important to note that the choice between fine-tuning and transfer learning depends on the specific task, dataset, and available computational resources. Experimentation and evaluation are key to determining the most effective approach for a given scenario.

There is word doc that sunny has where all the architectures are briefly discussed:
https://docs.google.com/document/d/1Y9K8z7MY7AMgnwo3wSZqgAxyKoCcevUVSYDBJ8aoFeU/edit