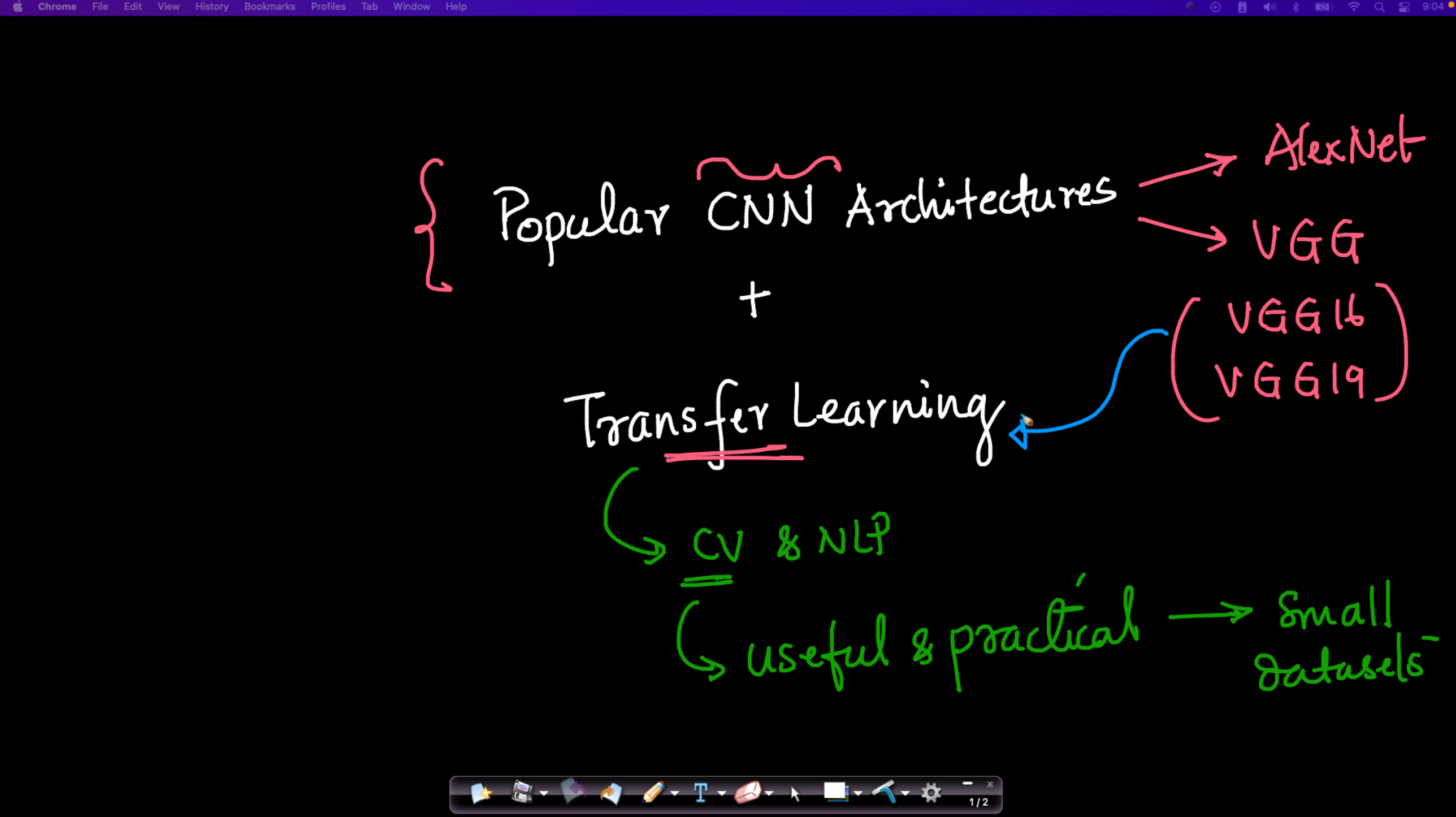
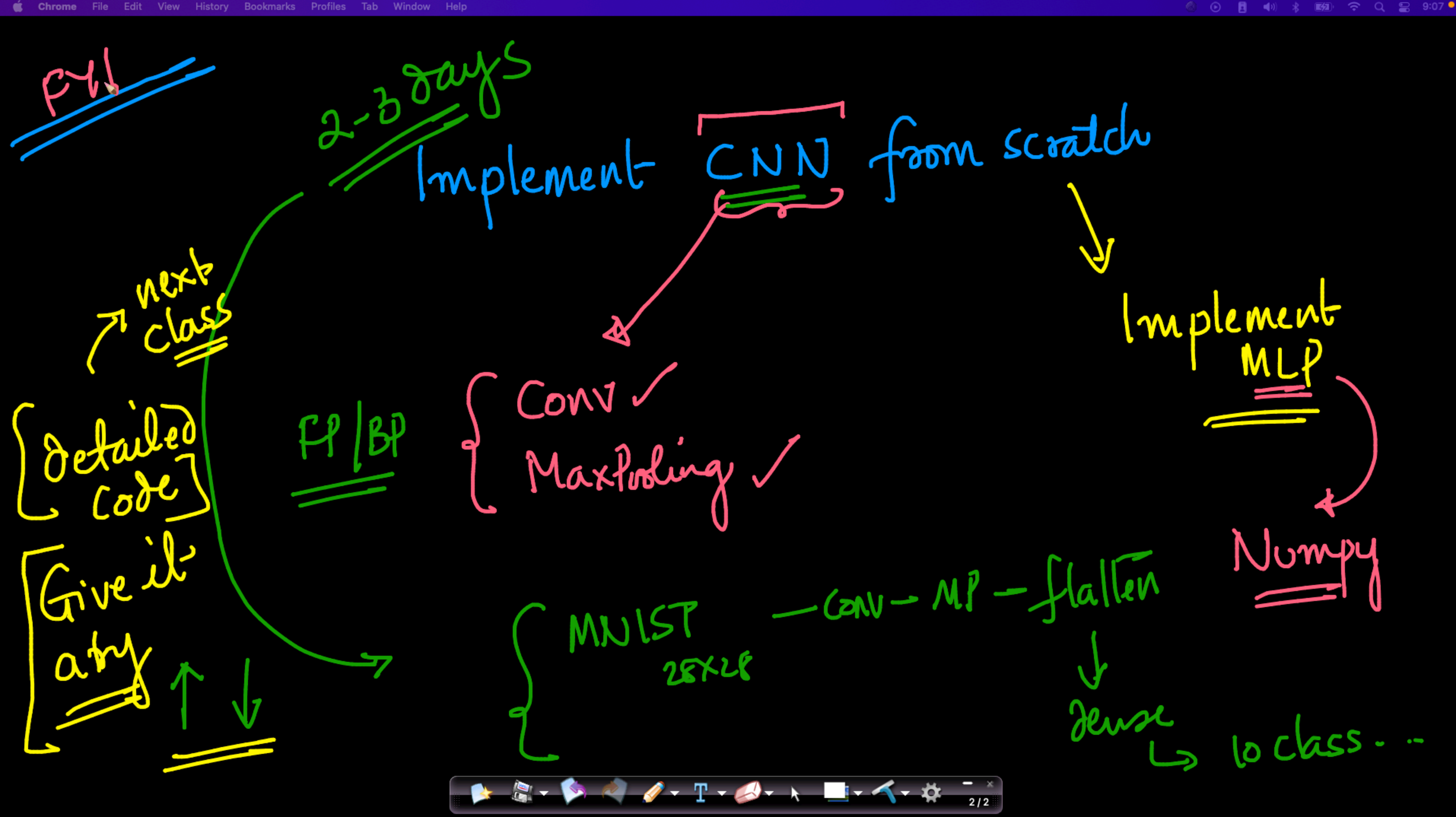
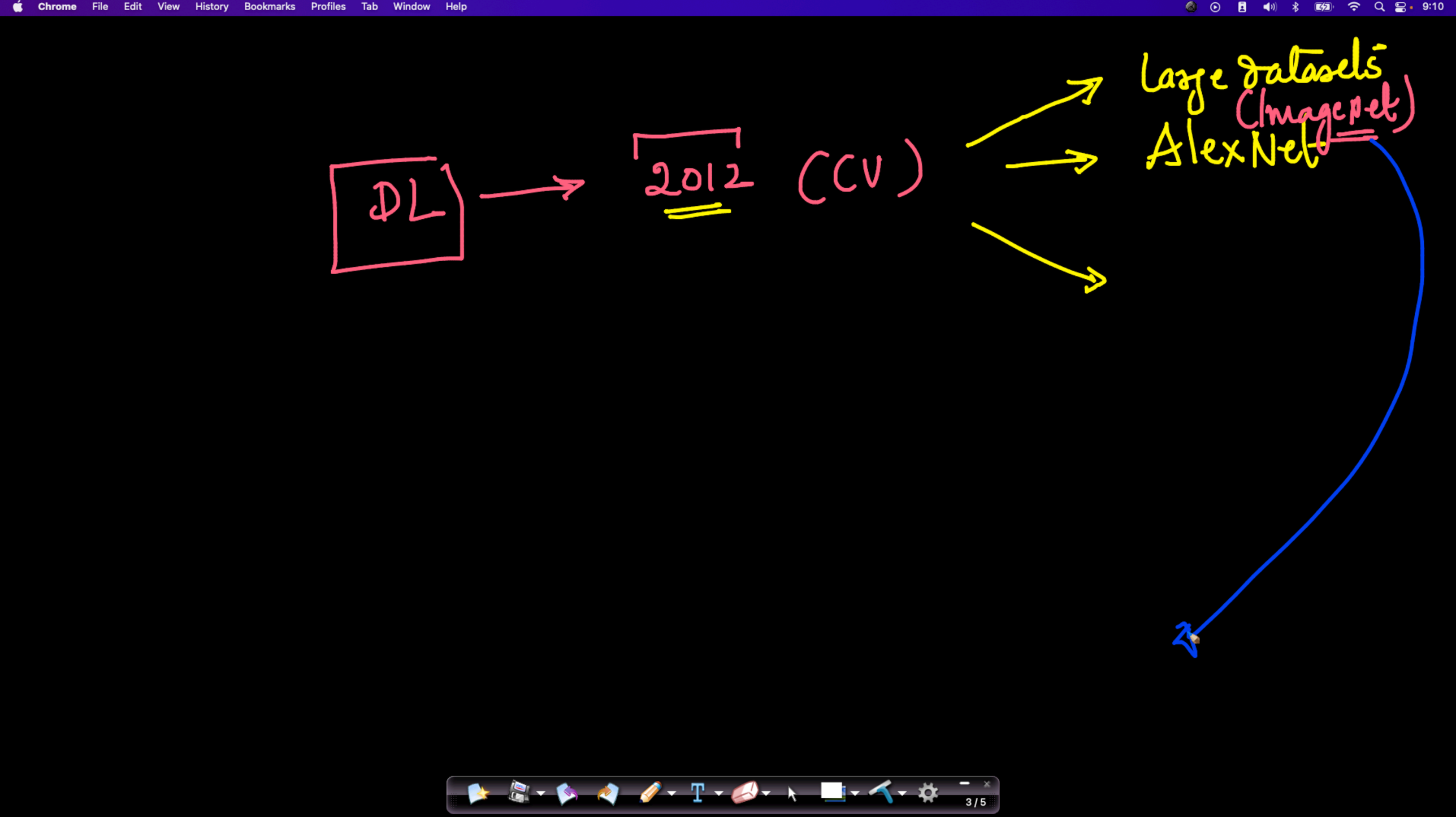

Popular CNN Architectures
AlexNet
VGG
+
Transfer Learning
VGG16
VGG19
CV & NLP
useful & practical → Small datasets


Chrome File Edit View History Bookmarks Profiles Tab Window Help
9:07
FYI
2-3 days
Implement- CNN from scratch
next class
[detailed code]
[Give it a try]
FP/BP
{ Conv ✓
MaxPooling ✓
Implement MLP
Numpy
{ MNIST 28x28 — Conv - MP - flatten
dense
to class...
2/2


DL
2012
(CV)
Large datasets
(Imagenet)
AlexNet

2010 - 2012

→ Large Dataset (annotated) : ImageNet,
{ 1000 categories
~1M images

→ Large Complex models (10's or 100's MM)
CNN (90's); Relu; BN; pooling

→ GPU for DL

Main page
Contents
Current events
Random article
About Wikipedia
Contact us
Donate

Contribute
Help
Learn to edit
Community portal
Recent changes
Upload file

Tools
What links here
Related changes
Special pages
Permanent link
Page information
Cite this page
Wikidata item



# ImageNet

From Wikipedia, the free encyclopedia

The **ImageNet** project is a large visual database designed for use in visual object recognition software research. More than 14 million[1][2] images have been hand-annotated by the project to indicate what objects are pictured and in at least one million of the images, bounding boxes are also provided.[3] ImageNet contains more than 20,000 categories,[2] with a typical category, such as "balloon" or "strawberry", consisting of several hundred images.[4] The database of annotations of third-party image URLs is freely available directly from ImageNet, though the actual images are not owned by ImageNet.[5] Since 2010, the ImageNet project runs an annual software contest, the ImageNet Large Scale Visual Recognition Challenge (ILSVRC), where software programs compete to correctly classify and detect objects and scenes. The challenge uses a "trimmed" list of one thousand non-overlapping classes.[6]

**Contents** [hide]

# Neural Networks

**Alex Krizhevsky**
University of Toronto
kriz@cs.utoronto.ca

**Ilya Sutskever**
University of Toronto
ilya@cs.utoronto.ca

**Geoffrey E. Hinton**
University of Toronto
hinton@cs.utoronto.ca

## Abstract

We trained a large, deep convolutional neural network to classify the 1.2 million high-resolution images in the ImageNet LSVRC-2010 contest into the 1000 different classes. On the test data, we achieved top-1 and top-5 error rates of 37.5% and 17.0% which is considerably better than the previous state-of-the-art. The neural network, which has 60 million parameters and 650,000 neurons, consists of five convolutional layers, some of which are followed by max-pooling layers, and three fully-connected layers with a final 1000-way softmax. To make training faster, we used non-saturating neurons and a very efficient GPU implementation of the convolution operation. To reduce overfitting in the fully-connected layers we employed a recently-developed regularization method called "dropout" that proved to be very effective. We also entered a variant of this model in the ILSVRC-2012 competition and achieved a winning top-5 test error rate of 15.3%, compared to 26.2% achieved by the second-best entry.

## 1 Introduction

Current approaches to object recognition make essential use of machine learning methods. To improve their performance, we can collect larger datasets, learn more powerful models, and use better techniques for preventing overfitting. Until recently, datasets of labeled images were relatively small — o[illegible] [8, 9], and CIFAR-10[illegible] this size.

## Agenda:

- Recognition of world famous landmark
- Standard CNN architecture like AlexNet and VGGNet
- Transfer learning using VGGNet

## Problem Statement:

- Suppose you are working as Data Scientist in Google. You have been given with a task to automate the categorization of famous landmarks.


AlexNet-
{ VGG16
VGG19
Transfer
learning

```
[ ] train_data = tf.keras.utils.image_dataset_from_directory("New_landmark_images/train/",shuffle =True, seed=123,image_size=(227, 227),batch
    val_data   = tf.keras.utils.image_dataset_from_directory("New_landmark_images/validation/",shuffle =False, seed=123,image_size=(227, 227)
    test_data = tf.keras.utils.image_dataset_from_directory("New_landmark_images/test/",shuffle =False, seed=123,image_size=(227, 227),batch_

    from tensorflow.keras import layers
    data_preprocess = tf.keras.Sequential(
        name="data_preprocess",
        layers=[ layers.Rescaling(1.0/255),]
    )

    # Perform Data Processing on the train, val, test dataset
    train_ds = train_data.map(lambda x, y: (data_preprocess(x), y))
    val_ds = val_data.map(lambda x, y: (data_preprocess(x), y))
    test_ds = test_data.map(lambda x, y: (data_preprocess(x), y))
```

```
Found 737 files belonging to 10 classes.
Found 155 files belonging to 10 classes.
Found 43 files belonging to 10 classes.
```

- Is the dataset balanced ?
- ◦ Yes

> Each class in the training dataset has almost 70 images. Each class in the validation dataset has almost 15 images.
> Each class in the test dataset has almost 5 images.

8 / 8

```
train_data = tf.keras.utils.image_dataset_from_directory("New_landmark_images/train/",shuffle =True, seed=123,image_size=(227, 227),batch
val_data   = tf.keras.utils.image_dataset_from_directory("New_landmark_images/validation/",shuffle =False, seed=123,image_size=(227, 227)
test_data = tf.keras.utils.image_dataset_from_directory("New_landmark_images/test/",shuffle =False, seed=123,image_size=(227, 227),batch_

from tensorflow.keras import layers
data_preprocess = tf.keras.Sequential(
    name="data_preprocess",
    layers=[ layers.Rescaling(1.0/255),]
)

# Perform Data Processing on the train, val, test dataset
train_ds = train_data.map(lambda x, y: (data_preprocess(x), y))
val_ds = val_data.map(lambda x, y: (data_preprocess(x), y))
test_ds = test_data.map(lambda x, y: (data_preprocess(x), y))
```

```
Found 737 files belonging to 10 classes.
Found 155 files belonging to 10 classes.
Found 43 files belonging to 10 classes.
```

- Is the dataset balanced ?
- ◦ Yes

> Each class in the training dataset has almost 70 images. Each class in the validation dataset has almost 15 images.
> Each class in the test dataset has almost 5 images.

9 / 9

```
train_data = tf.keras.utils.image_dataset_from_directory("New_landmark_images/train/",shuffle =True, seed=123,image_size=(227, 227),batch
val_data   = tf.keras.utils.image_dataset_from_directory("New_landmark_images/validation/",shuffle =False, seed=123,image_size=(227, 227)
test_data = tf.keras.utils.image_dataset_from_directory("New_landmark_images/test/",shuffle =False, seed=123,image_size=(227, 227),batch_

from tensorflow.keras import layers
data_preprocess = tf.keras.Sequential(
    name="data_preprocess",
    layers=[ layers.Rescaling(1.0/255),]
)

# Perform Data Processing on the train, val, test dataset
train_ds = train_data.map(lambda x, y: (data_preprocess(x), y))
val_ds = val_data.map(lambda x, y: (data_preprocess(x), y))
test_ds = test_data.map(lambda x, y: (data_preprocess(x), y))
```

```
Found 737 files belonging to 10 classes.
Found 155 files belonging to 10 classes.
Found 43 files belonging to 10 classes.
```

Tr
Val

Small Dataset

- Is the dataset balanced ?
- ◦ Yes

> Each class in the training dataset has almost 70 images. Each class in the validation dataset has almost 15 images.
> Each class in the test dataset has almost 5 images.

# 1. Creating a new model from scratch

- You can build a Convolution neural network(CNN) model from scratch and use them to classify the data
- But
    - How many conv layers do you need to use?
    - Which **architecture should you choose ?**
    - How should you combine CNN / Pooling / FC layers to get the best performance ?
    - Should you use 5x5 kernel or 3x3 kernel or 11x11 kernel?
    - Can you achieve a high accuracy if dataset is small in size ?
    - This will require lot of experimentation to get a model of very high accuracy
    - Also, training a deep neural network will require a lot of time
- Instead of doing all the above experiment from scratch, we will use State-of-the-Art(SOTA) models and train it from scratch like VGGNet
- Before going to VGGNet, we will first understand AlexNet

Small dataset

↓

CNN

## What is AlexNet ?

- AlexNet is the **first CNN to win ImageNet Competition in 2012.**
- It was primarily designed by **Alex** Krizhevsky.
- They achieved a **top 5 accuracy of 84.7%,** while runner-up was at 74!
- So, it was a huge achievement and We can say that, Alexnet started the CNN revolution.
- Neural networks had been around since 1970s and Theroy of CNN first came in 1989
- Why couldnt CV Reseachers implement CNNs earlier

# 1. Creating a new model from scratch

- You can build a Convolution neural network(CNN) model from scratch and use them to classify the data
- But
  - How many conv layers do you need to use?
  - Which **architecture should you choose ?**
  - How should you combine CNN / Pooling / FC layers to get the best performance ?
  - Should you use 5x5 kernel or 3x3 kernel or 11x11 kernel?
  - Can you achieve a high accuracy if dataset is small in size ?
  - This will require lot of experimentation to get a model of very high accuracy
  - Also, training a deep neural network will require a lot of time
- Instead of doing all the above experiment from scratch, we will use State-of-the-Art(SOTA) models and train it from scratch like VGGNet
- Before going to VGGNet, we will first understand AlexNet

## What is AlexNet ?

- AlexNet is the **first CNN to win ImageNet Competition in 2012.**
- It was primarily designed by **Alex** Krizhevsky.
- They achieved a **top 5 accuracy of 84.7%,** while runner-up was at 74!
- So, it was a huge achievement and We can say that, Alexnet started the CNN revolution.
- Neural networks had been around since 1970s and Theroy of CNN first came in 1989
- Why couldnt CV Reseachers implement CNNs earlier

# 1. Creating a new model from scratch

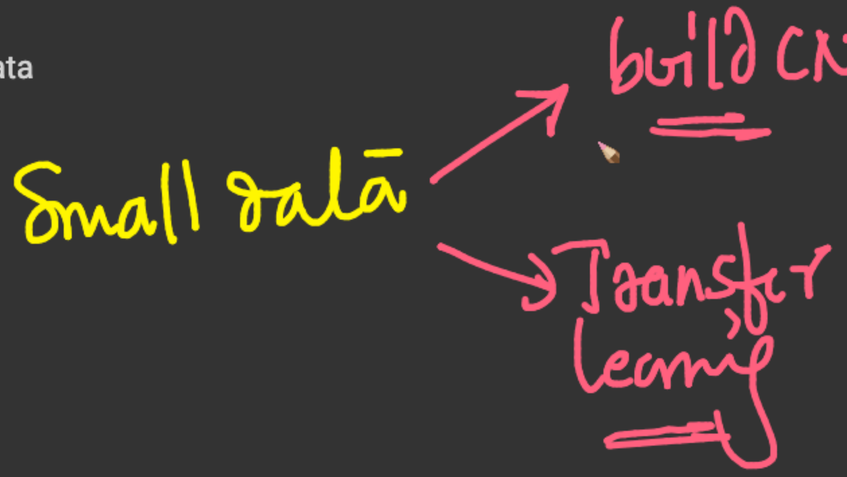
- You can build a Convolution neural network(CNN) model from scratch and use them to classify the data
- But
  - How many conv layers do you need to use?
  - Which **architecture should you choose ?**
  - How should you combine CNN / Pooling / FC layers to get the best performance ?
  - Should you use 5x5 kernel or 3x3 kernel or 11x11 kernel?
  - Can you achieve a high accuracy if dataset is small in size ?
  - This will require lot of experimentation to get a model of very high accuracy
  - Also, training a deep neural network will require a lot of time
- Instead of doing all the above experiment from scratch, we will use State-of-the-Art(SOTA) models and train it from scratch like VGGNet
- Before going to VGGNet, we will first understand AlexNet

## What is AlexNet ?

- AlexNet is the **first CNN to win ImageNet Competition in 2012.**
- It was primarily designed by **Alex** Krizhevsky.
- They achieved a **top 5 accuracy of 84.7%,** while runner-up was at 74!
- So, it was a huge achievement and We can say that, Alexnet started the CNN revolution.
- Neural networks had been around since 1970s and Theroy of CNN first came in 1989
- Why couldnt CV Reseachers implement CNNs earlier

# 1. Creating a new model from scratch

- You can build a Convolution neural network(CNN) model from scratch and use them to classify the data
- But
  - How many conv layers do you need to use?
  - Which **architecture should you choose ?**
  - How should you combine CNN / Pooling / FC layers to get the best performance ?
  - Should you use 5x5 kernel or 3x3 kernel or 11x11 kernel?
  - Can you achieve a high accuracy if dataset is small in size ?
  - This will require lot of experimentation to get a model of very high accuracy
  - Also, training a deep neural network will require a lot of time
- Instead of doing all the above experiment from scratch, we will use State-of-the-Art(SOTA) models and train it from scratch like VGGNet
- Before going to VGGNet, we will first understand AlexNet




## What is AlexNet ?

- AlexNet is the **first CNN to win ImageNet Competition in 2012.**
- It was primarily designed by **Alex** Krizhevsky.
- They achieved a **top 5 accuracy of 84.7%,** while runner-up was at 74!
- So, it was a huge achievement and We can say that, Alexnet started the CNN revolution.
- Neural networks had been around since 1970s and Theroy of CNN first came in 1989
- Why couldnt CV Reseachers implement CNNs earlier

- Should you use 5x5 kernel or 3x3 kernel or 11x11 kernel?
- Can you achieve a high accuracy if dataset is small in size ?
- This will require lot of experimentation to get a model of very high accuracy
- Also, training a deep neural network will require a lot of time

- Instead of doing all the above experiment from scratch, we will use State-of-the-Art(SOTA) models and train it from scratch like VGGNet
- Before going to VGGNet, we will first understand AlexNet

## What is AlexNet ?

- AlexNet is the **first CNN to win ImageNet Competition in 2012.**
- It was primarily designed by **Alex** Krizhevsky.
- They achieved a **top 5 accuracy of 84.7%,** while runner-up was at 74!
- So, it was a huge achievement and We can say that, Alexnet started the CNN revolution.
- Neural networks had been around since 1970s and Theroy of CNN first came in 1989
- Why couldnt CV Reseachers implement CNNs earlier

top-k accuracy

**What is ImageNet Competition ?**

- The ImageNet Large Scale Visual Recognition Challenge, or ILSVRC, is an annual competition that uses subsets from the ImageNet dataset (1000 classes) and is designed to foster the development and benchmarking of state-of-the-art algorithms.
- ImageNet is a large database or dataset of over 14 million images with 21000 labels. It was designed by academics intended for computer vision research. It was the first of its kind in terms of scale. Images are organized and labelled in a hierarchy.

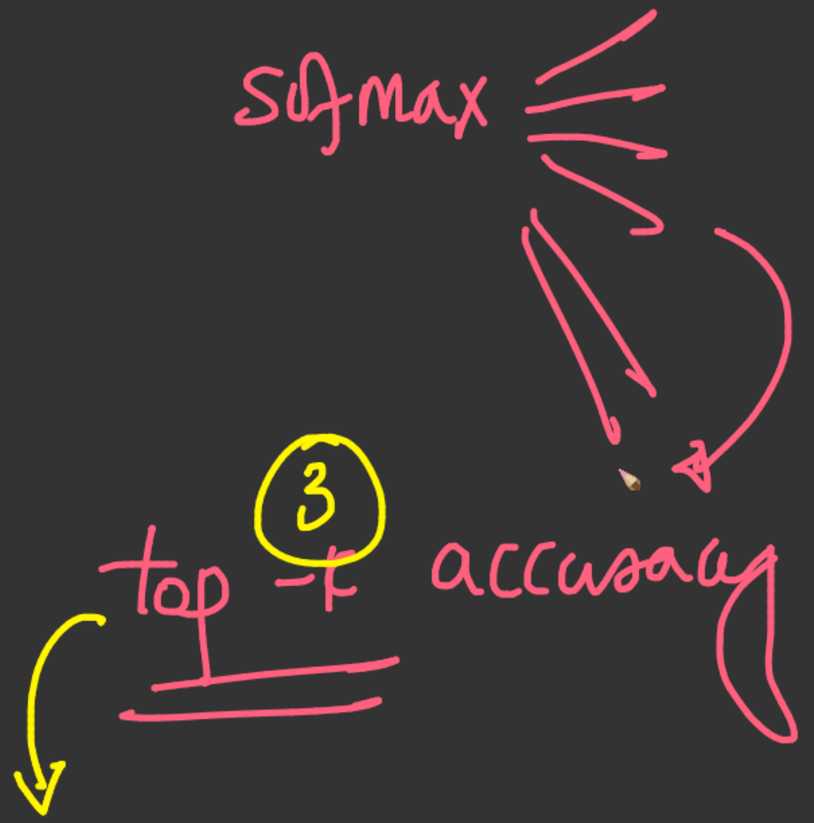
For more details about imagenet:

1.https://www.image-net.org/challenges/LSVRC/ 2.https://devopedia.org/imagenet

- Example, in recommender system it makes more sense to show top 3 or 5 recommended products to customer rather than just single top most recimmended product.
- In top-k accuracy, we get score if the right answer appears in top k guesses.

| True label | Top-3 Predicted labels |
|---|---|
| Cat | **Cat**, Lion, Dog |
| Dog | Giraffe, Lion, Cat |
| Lion | Cat, **Lion**, Dog |
| Giraffe | **Giraffe**, Dog, Cat |
| Dolphin | **Dolphin**, Cat, Giraffe |

| True label | Top-3 Predicted labels | Correct |
|---|---|---|
| Cat | **Cat**, Lion, Dog | ✓ |
| Dog | Giraffe, Lion, Cat | ✗ |
| Lion | Cat, **Lion**, Dog | ✓ |
| Giraffe | **Giraffe**, Dog, Cat | ✓ |
| Dolphin | **Dolphin**, Cat, Giraffe | ✓ |




How Top-k accuracy helpful working with ima

Q

Small dataset

~70 images per class : DTV

↓ Augmentation

~700/1000 images per class

⇓

7000 - 10k images ← 50 MM params

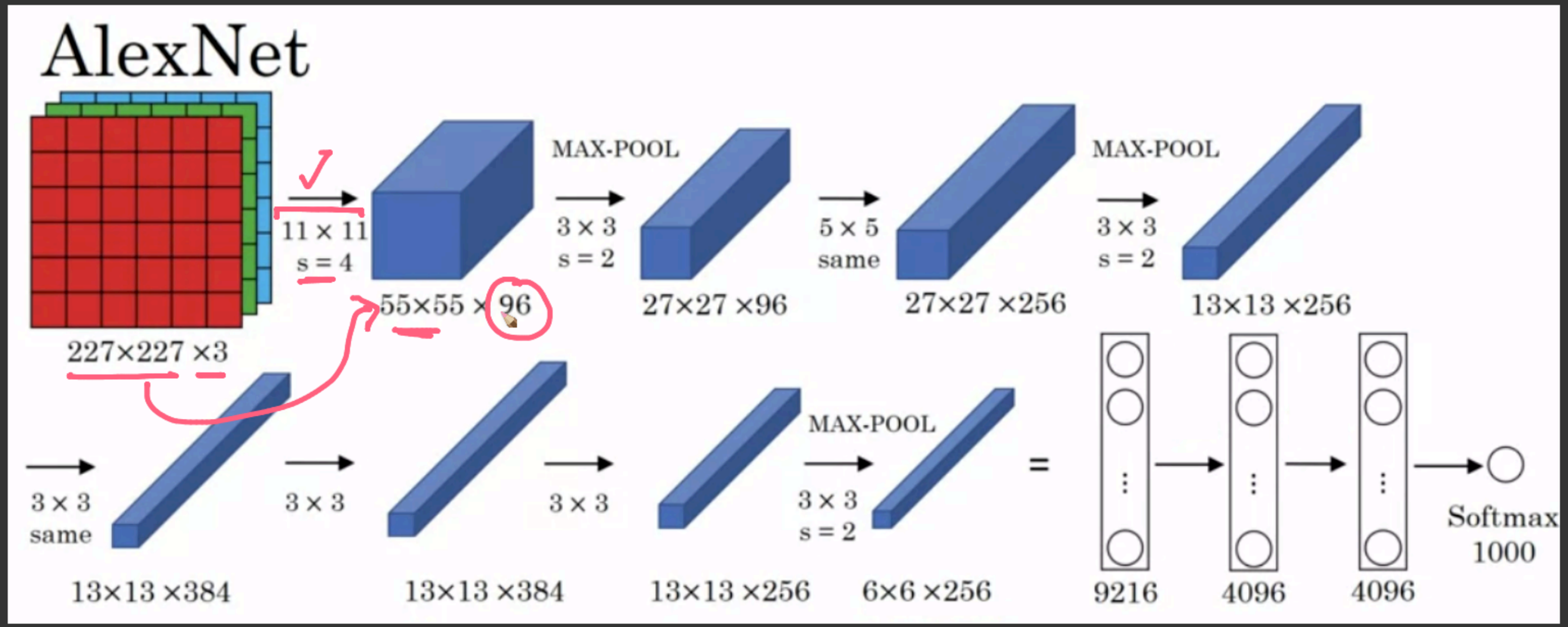



Figure: : AlexNet Architecture

| Layer | Filter/neurons | Filter size | Stride | Padding | Size of feature map |
|---|---|---|---|---|---|
| Input | | | | | 227 x 227 x3 |
| Conv1 | 96 | 11x11 | 4 | | 55 x 55 x 96 |

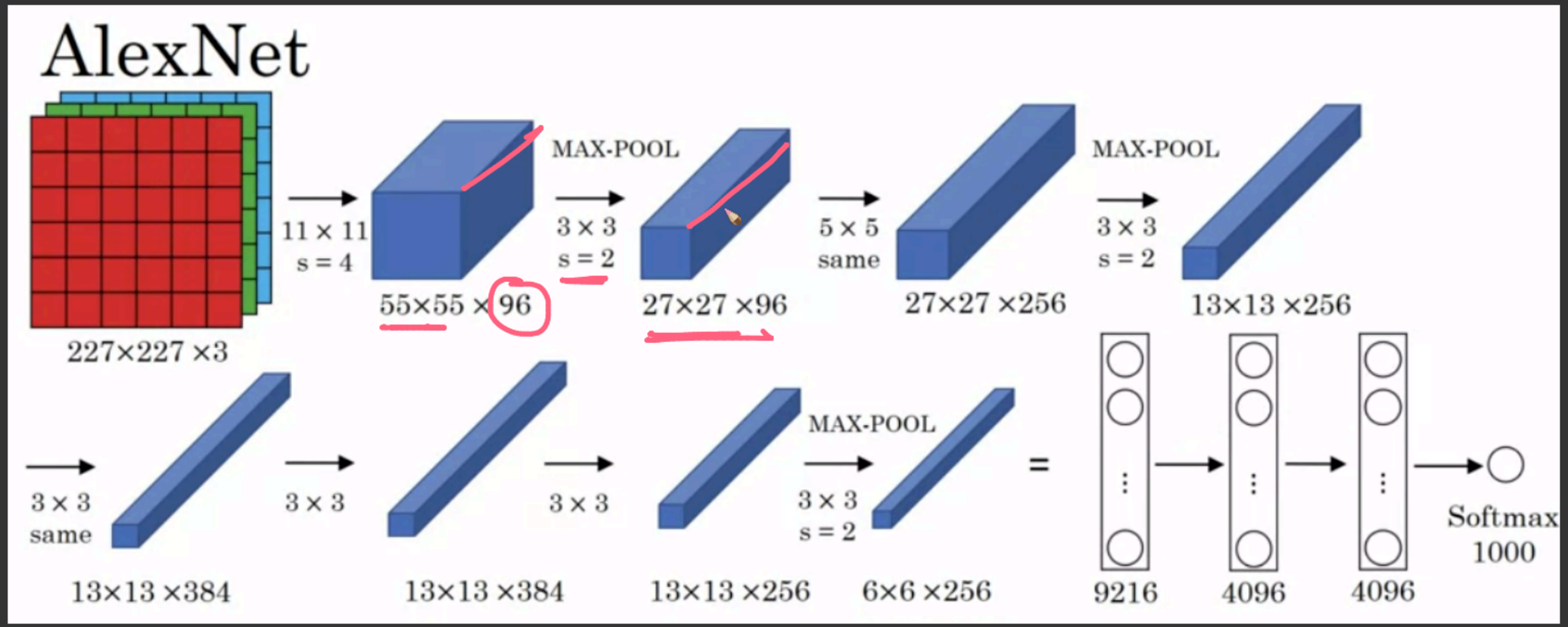



Figure: : AlexNet Architecture

| Layer | Filter/neurons | Filter size | Stride | Padding | Size of feature map |
|---|---|---|---|---|---|
| Input | | | | | 227 x 227 x3 |
| Conv1 | 96 | 11x11 | 4 | | 55 x 55 x 96 |

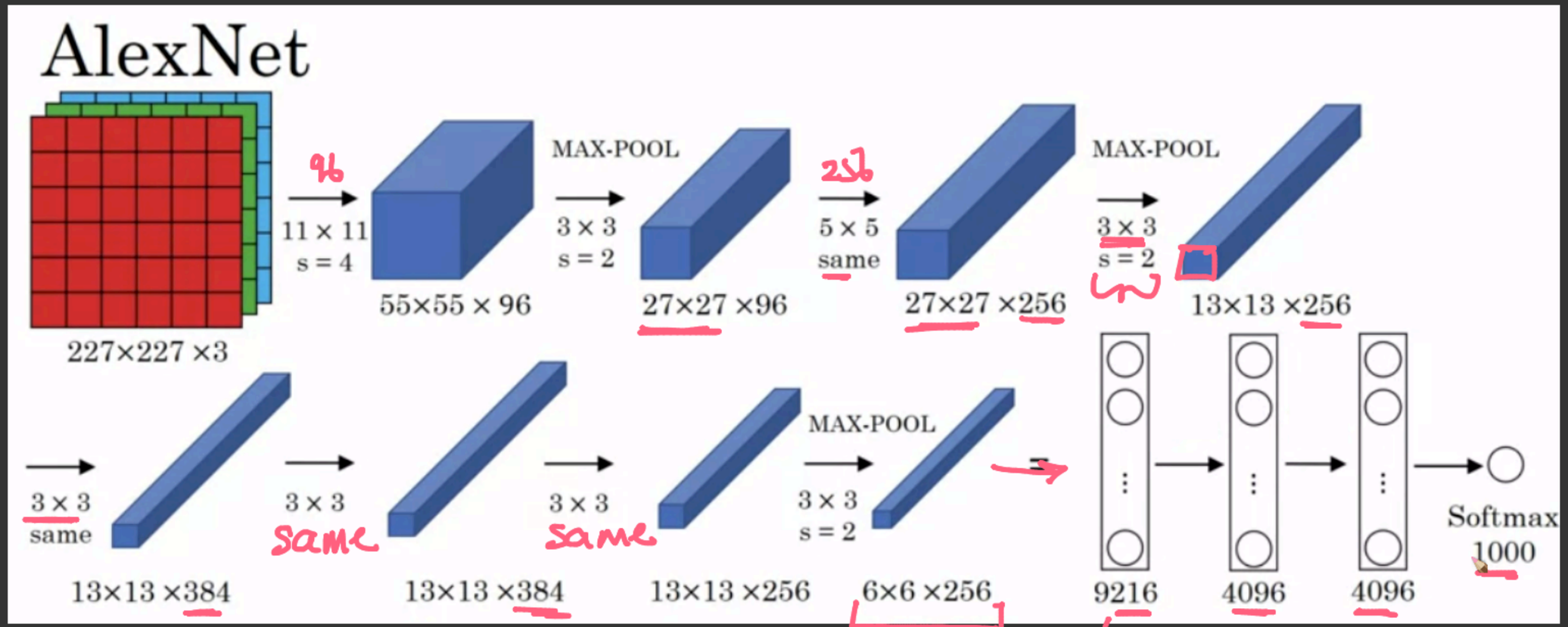



Figure: : AlexNet Architecture

| Layer | Filter/neurons | Filter size | Stride | Padding | Size of feature map |
|---|---|---|---|---|---|
| Input | | | | | 227 x 227 x3 |
| Conv1 | 96 | 11x11 | 4 | | 55 x 55 x 96 |

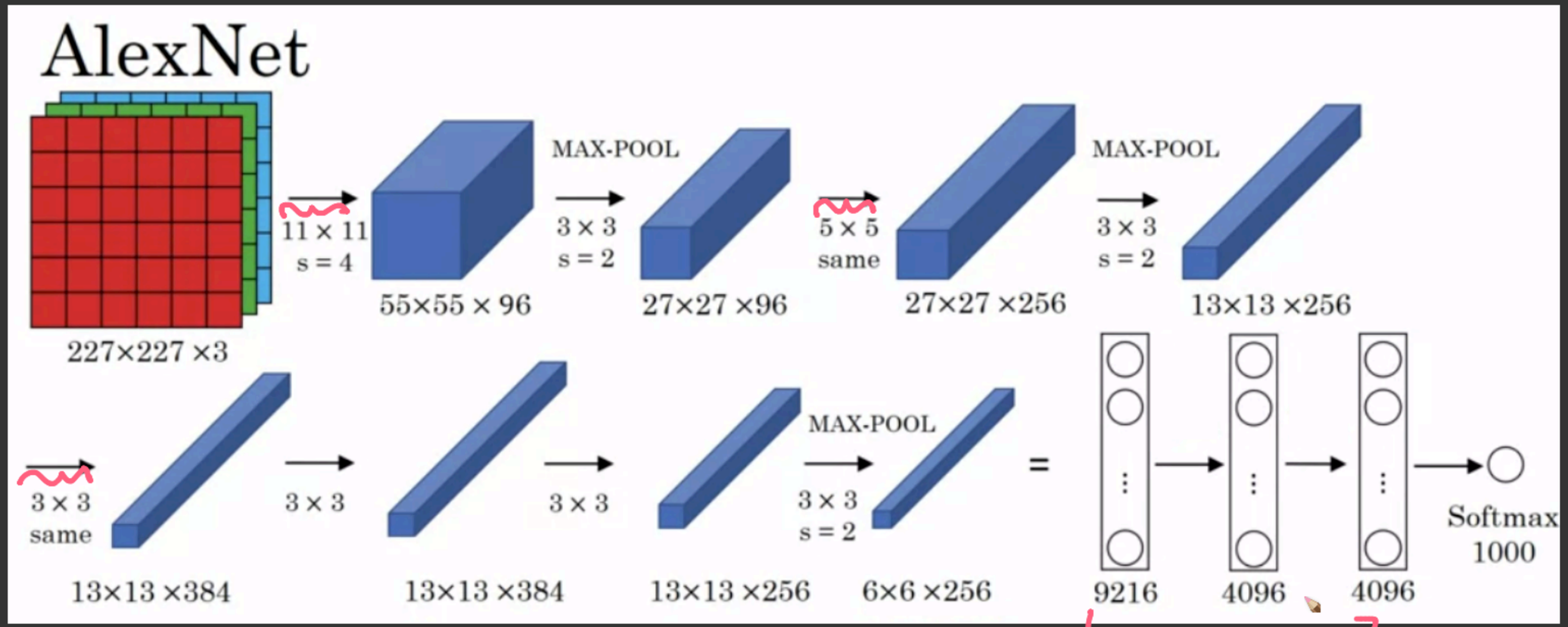



Figure: : AlexNet Architecture

| Layer | Filter/neurons | Filter size | Stride | Padding | Size of feature map |
|---|---|---|---|---|---|
| Input | | | | | 227 x 227 x3 |
| Conv1 | 96 | 11x11 | 4 | | 55 x 55 x 96 |

| | | | | | |
|---|---|---|---|---|---|
| Conv 3 | 384 | 3 x 3 | 1 | 1 | 13 x 13 x 384 |
| Conv 4 | 384 | 3 x 3 | 1 | 1 | 13 x 13 x 384 |
| Conv 5 | 256 | 3 x 3 | 1 | 1 | 13 x 13 x 256 |
| Max Pool 3 | | 3 x 3 | 2 | | 6 x 6 x 256 |
| Dropout 1 | Rate = 0.5 | | | | 6 x 6 x 256 |
| FC 1 | | | | | 4096 |
| Dropout 2 | Rate = 0.5 | | | | 4096 |
| FC2 | | | | | 4096 |
| FC3 | | | | | 1000 |

Figure: : AlexNet layers Computation

- AlexNet has 5 CONV layers and 3 FC layers
- AlexNet has 60M parameters.The following table shows it's performance on ImageNet dataset Click to know more

| Model Name | Number of params | Top 1 Acc | Top 5 Acc |
|---|---|---|---|
| Alexnet | 60M | 63.3 | 84.6 |

## Quiz-3

Which of the following options can be used to reduce overfitting in deep learning models?

1. Increase architectural complexity
2. Use data augmentation

Ans : (a) (11 * 11 * 3 +1) *96. The total parameter = weight + bias . weight = 11 * 11 * 3 * 96 and bias = 96

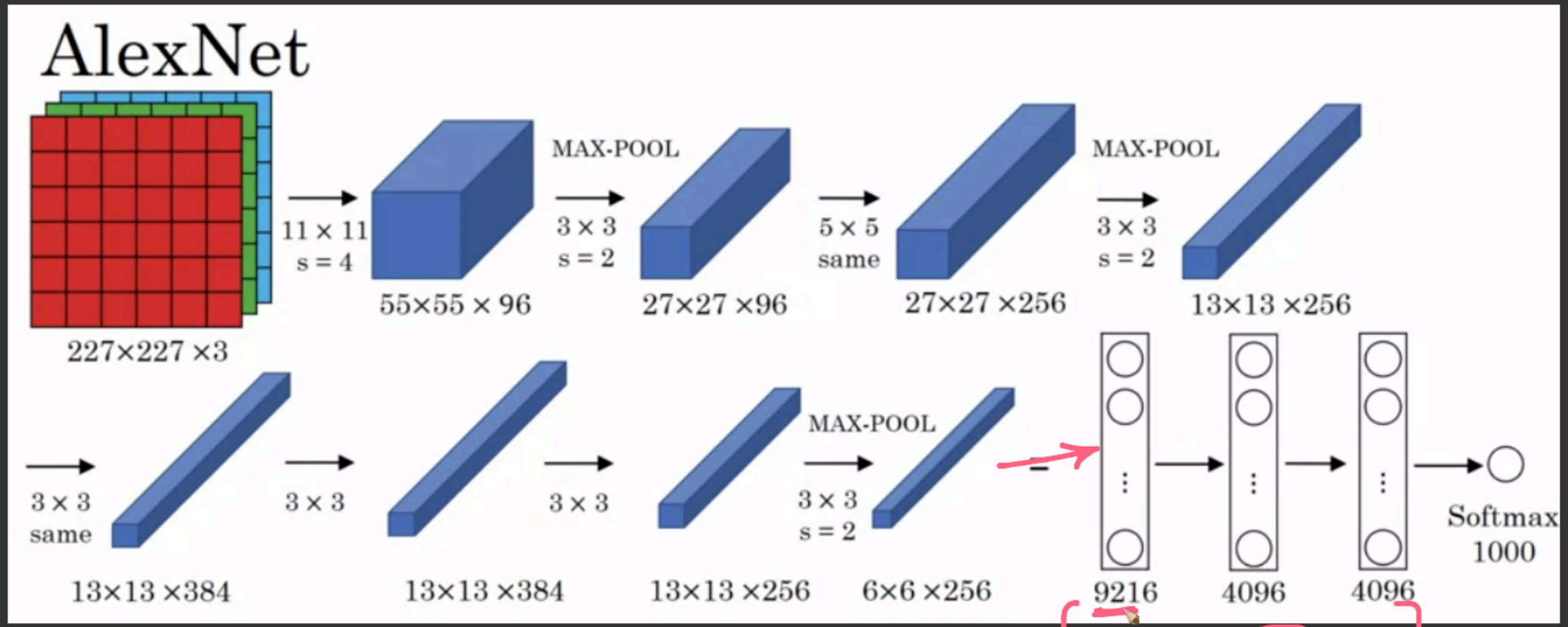



Figure: : AlexNet Architecture

| Layer | Filter/neurons | Filter size | Stride | Padding | Size of feature map |
|---|---|---|---|---|---|
| Input | | | | | 227 x 227 x3 |

Ans : (a) (11 * 11 * 3 +1) *96. The total parameter = weight + bias . weight = 11 * 11 * 3 * 96 and bias = 96

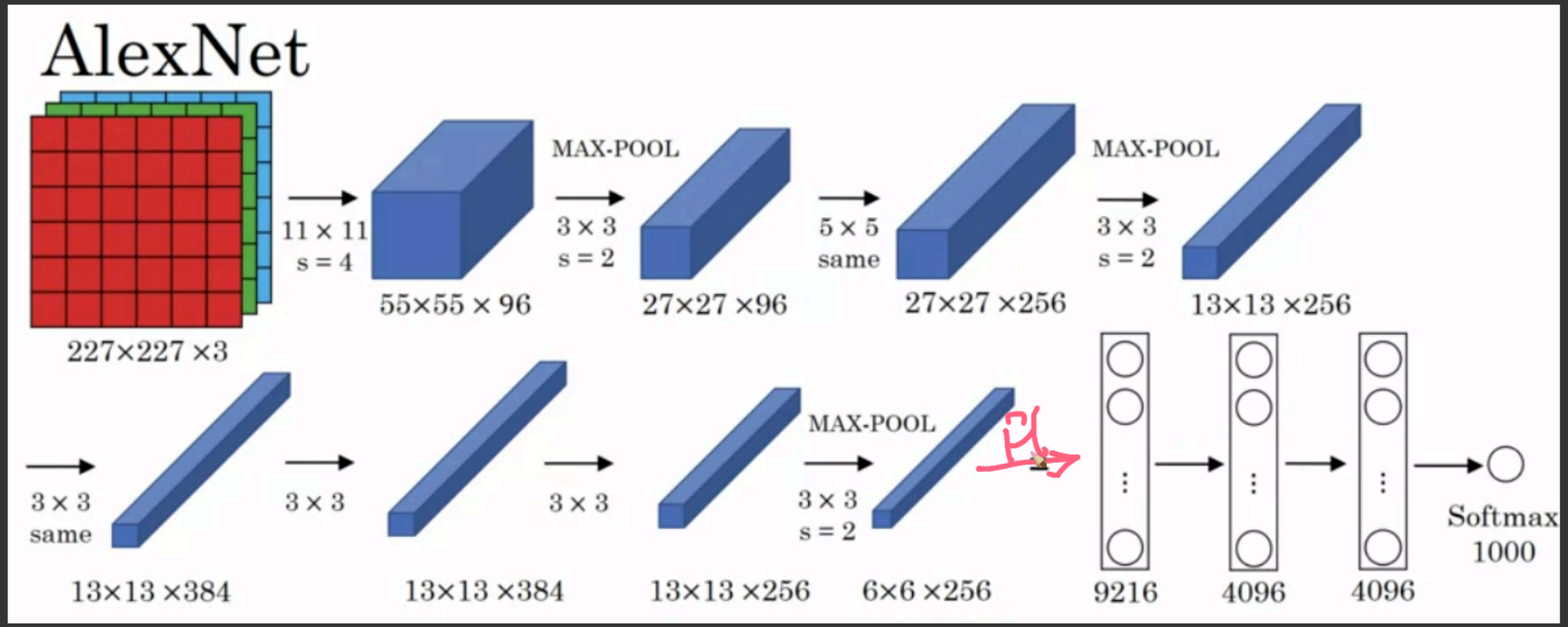



Figure: : AlexNet Architecture

| Layer | Filter/neurons | Filter size | Stride | Padding | Size of feature map |
|---|---|---|---|---|---|
| Input | | | | | 227 x 227 x3 |

Ans : (a) (11 * 11 * 3 +1) *96. The total parameter = weight + bias . weight = 11 * 11 * 3 * 96 and bias = 96

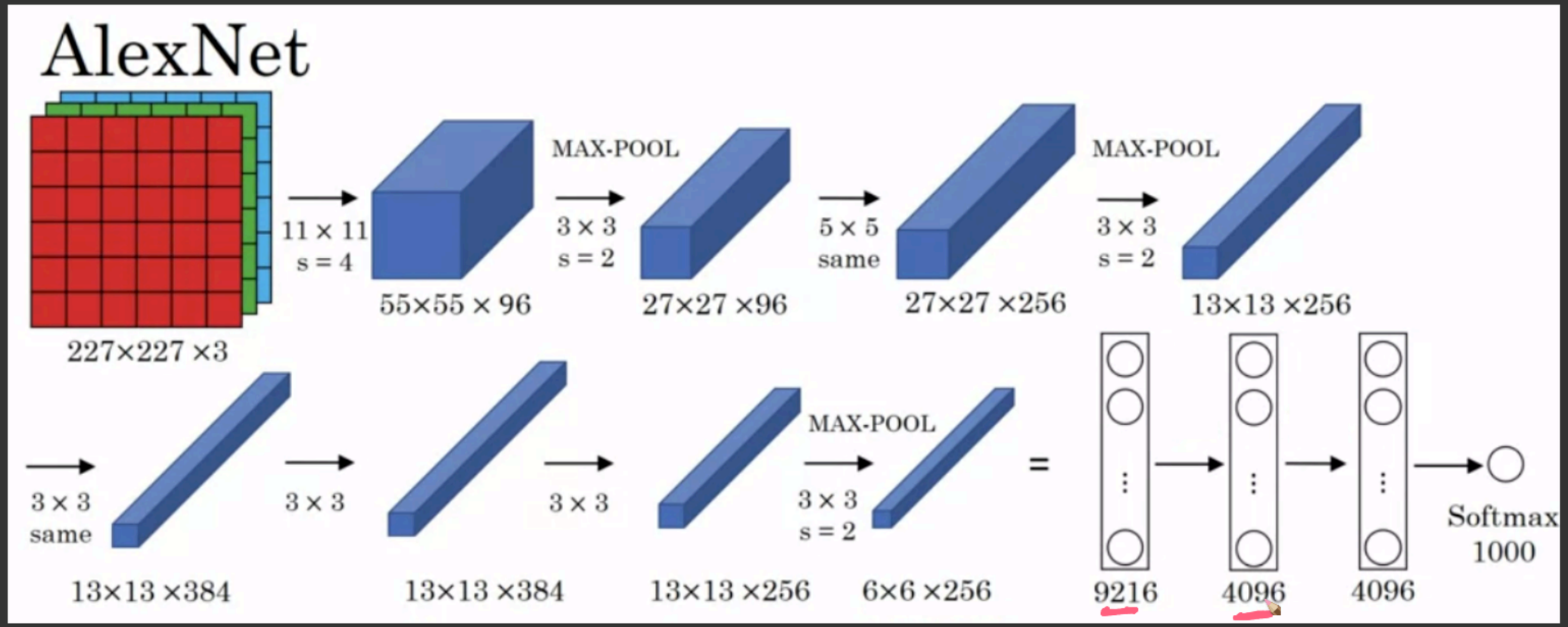



Figure: : AlexNet Architecture

| Layer | Filter/neurons | Filter size | Stride | Padding | Size of feature map |
|---|---|---|---|---|---|
| Input | | | | | 227 x 227 x3 |

Ans : (a) (11 * 11 * 3 +1) *96. The total parameter = weight + bias . weight = 11 * 11 * 3 * 96 and bias = 96

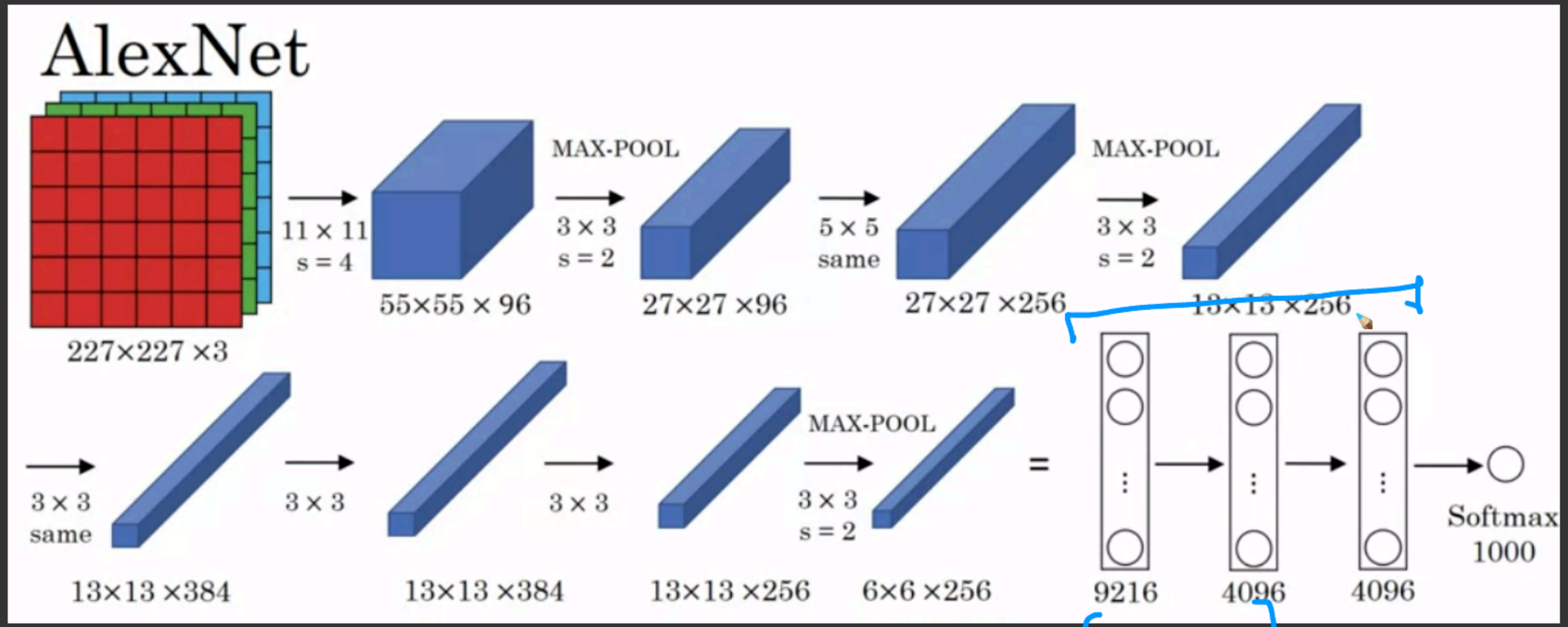



Figure: : AlexNet Architecture

| Layer | Filter/neurons | Filter size | Stride | Padding | Size of feature map |
|---|---|---|---|---|---|
| Input | | | | | 227 x 227 x3 |

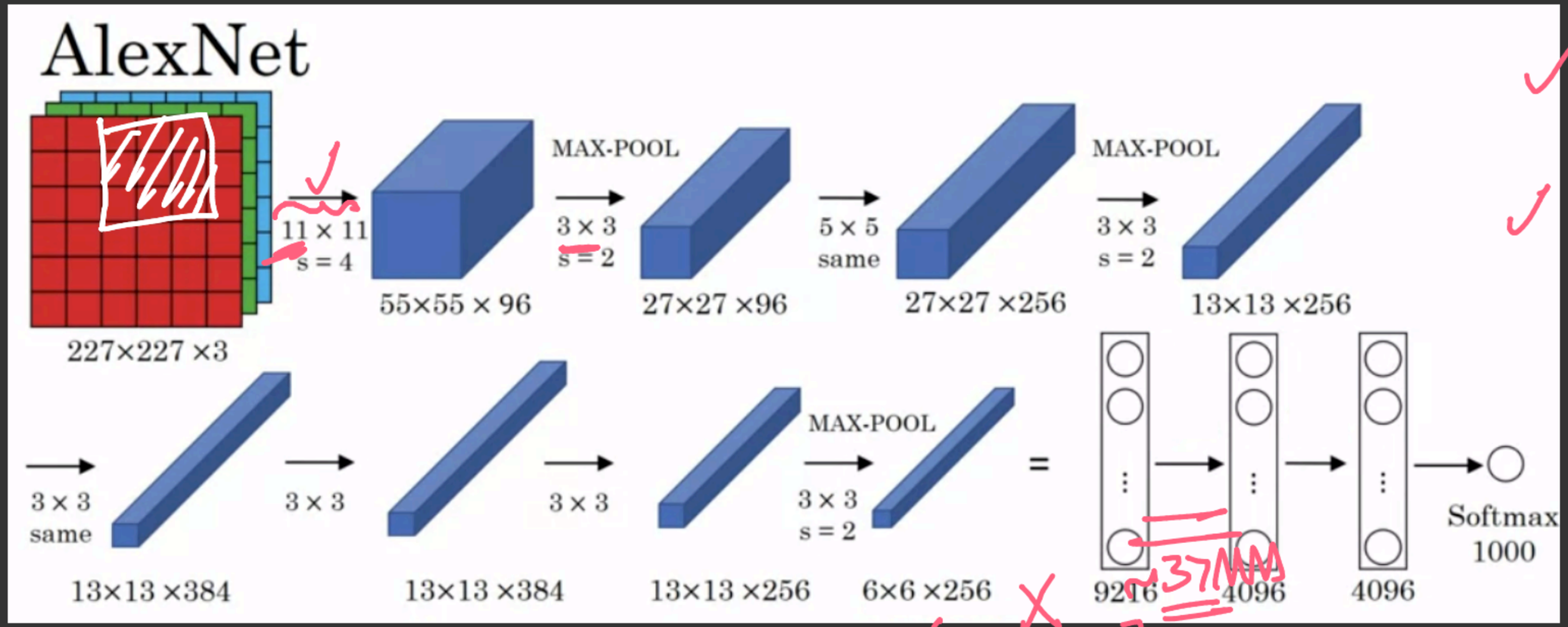



Figure: : AlexNet Architecture

| Layer | Filter/neurons | Filter size | Stride | Padding | Size of feature map |
|---|---|---|---|---|---|
| Input | | | | | 227 x 227 x3 |
| Conv1 | 96 | 11x11 | 4 | | 55 x 55 x 96 |

5 Conv, 3 FC

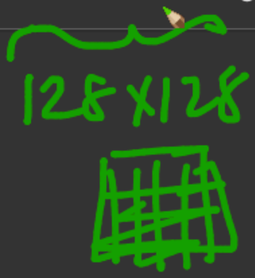
16 / 19

## What are the shortcomings of AlexNet?

- AlexNet is not very deep compared to later models like VGGNet.
- Deep neural network are prefered due to the following reasons:
  - Expressive : A single layer is a linear function . Activation function is used to learn complex non-linear function.
    - Stacking multiple layers results in multiple successive nonlinearities and has a better chance to approximate higher complex functionality
  - Perceptive field : Consider that a cat's head cover 128x128 pixel region in image.
    - Using a single convolution network will require 128x128 filter to capture it which will have a very high no of parameter.
    - Stacked layers on the other hand can use 3x3 filter to see 128x128 pixel, with smaller number of parameters.
  - Generalization : Adding parameter to a single layer increase the memory of neural network .
    - The neural network will memorize input examples and will not generalize well.
    - On the other hand, stacking many layers helps the network to breakdown the input into hierarchial structure of features.
    - For example, the intial initial layers will recognize the edge of face, eye, nose while the latter layer will assemble them and focus whether it is cat or dog
- AlexNet also uses a **filter size of 11x11** which is very **large and has high parameter**. Thus, it is **computantionally expensive**
- Although, AlexNet performed better than its previous architecture and achieved a top-5 accuracy of 84% on ImageNet,the accuracy of the model can be further improved by increasing the number of layers.

Can you achieve the same functionality by a smaller filter ?

## What are the shortcomings of AlexNet?

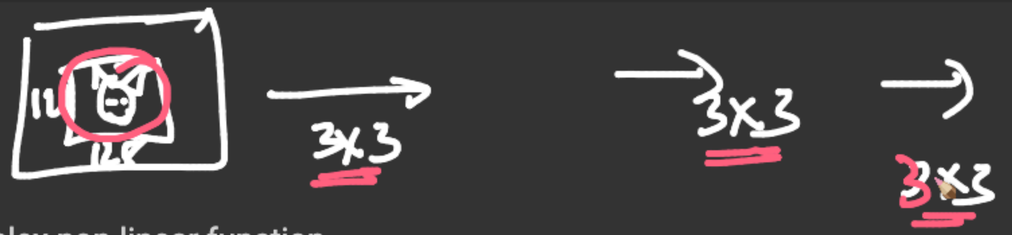



- AlexNet is not very deep compared to later models like VGGNet.
- Deep neural network are prefered due to the following reasons:
  - Expressive : A single layer is a linear function . Activation function is used to learn complex non-linear function.
    - Stacking multiple layers results in multiple successive nonlinearities and has a better chance to approximate higher complex functionality
  - Perceptive field : Consider that a cat's head cover 128x128 pixel region in image.
    - Using a single convolution network will require 128x128 filter to capture it which will have a very high no of parameter.
    - Stacked layers on the other hand can use 3x3 filter to see 128x128 pixel, with smaller number of parameters.
  - Generalization : Adding parameter to a single layer increase the memory of neural network .
    - The neural network will memorize input examples and will not generalize well.
    - On the other hand, stacking many layers helps the network to breakdown the input into hierarchial structure of features.
    - For example, the intial initial layers will recognize the edge of face, eye, nose while the latter layer will assemble them and focus whether it is cat or dog
- AlexNet also uses a **filter size of 11x11** which is very **large and has high parameter**. Thus, it is **computantionally expensive**
- Although, AlexNet performed better than its previous architecture and achieved a top-5 accuracy of 84% on ImageNet,the accuracy of the model can be further improved by increasing the number of layers.

Can you achieve the same functionality by a smaller filter ?

- We can achieve this by using 3*3 fil...

## What are the shortcomings of AlexNet?

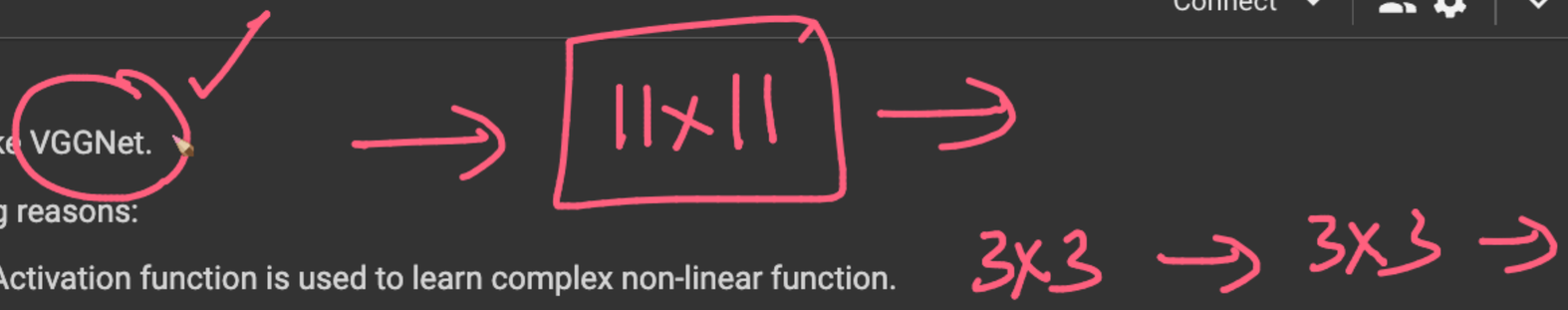



- AlexNet is not very deep compared to later models like VGGNet.
- Deep neural network are prefered due to the following reasons:
  - Expressive : A single layer is a linear function . Activation function is used to learn complex non-linear function.
    - Stacking multiple layers results in multiple successive nonlinearities and has a better chance to approximate higher complex functionality
  - Perceptive field : Consider that a cat's head cover 128x128 pixel region in image.
    - Using a single convolution network will require 128x128 filter to capture it which will have a very high no of parameter.
    - Stacked layers on the other hand can use 3x3 filter to see 128x128 pixel, with smaller number of parameters.
  - Generalization : Adding parameter to a single layer increase the memory of neural network .
    - The neural network will memorize input examples and will not generalize well.
    - On the other hand, stacking many layers helps the network to breakdown the input into hierarchial structure of features.
    - For example, the intial initial layers will recognize the edge of face, eye, nose while the latter layer will assemble them and focus whether it is cat or dog
- AlexNet also uses a **filter size of 11x11** which is very **large and has high parameter**. Thus, it is **computantionally expensive**
- Although, AlexNet performed better than its previous architecture and achieved a top-5 accuracy of 84% on ImageNet,the accuracy of the model can be further improved by increasing the number of layers.

Can you achieve the same functionality by a smaller filter ?

- We can achieve this by using 3*3 fil...

## What are the shortcomings of AlexNet?



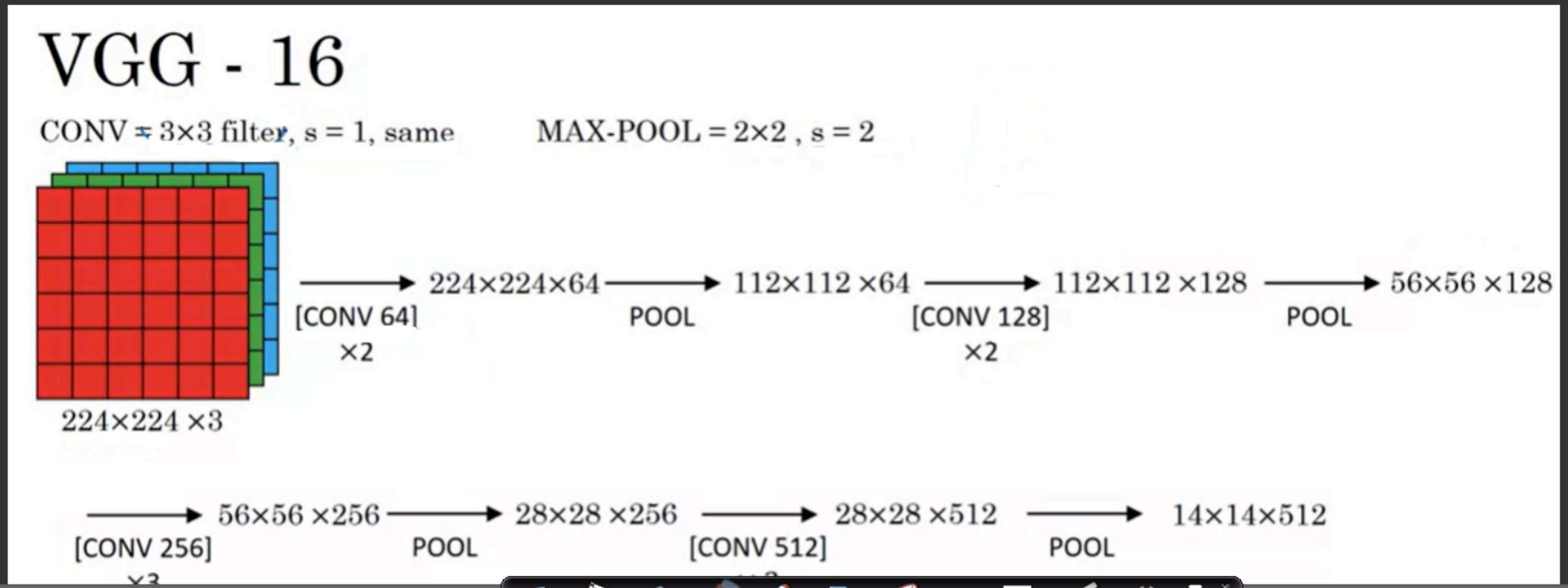

- AlexNet is not very deep compared to later models like VGGNet.
- Deep neural network are prefered due to the following reasons:
  - Expressive : A single layer is a linear function . Activation function is used to learn complex non-linear function.
    - Stacking multiple layers results in multiple successive nonlinearities and has a better chance to approximate higher complex functionality
  - Perceptive field : Consider that a cat's head cover 128x128 pixel region in image.
    - Using a single convolution network will require 128x128 filter to capture it which will have a very high no of parameter.
    - Stacked layers on the other hand can use 3x3 filter to see 128x128 pixel, with smaller number of parameters.
  - Generalization : Adding parameter to a single layer increase the memory of neural network .
    - The neural network will memorize input examples and will not generalize well.
    - On the other hand, stacking many layers helps the network to breakdown the input into hierarchial structure of features.
    - For example, the intial initial layers will recognize the edge of face, eye, nose while the latter layer will assemble them and focus whether it is cat or dog
- AlexNet also uses a **filter size of 11x11** which is very **large and has high parameter**. Thus, it is **computantionally expensive**
- Although, AlexNet performed better than its previous architecture and achieved a top-5 accuracy of 84% on ImageNet,the accuracy of the model can be further improved by increasing the number of layers.

Can you achieve the same functionality by a smaller filter ?

- We can achieve this by using 3*3 fil...

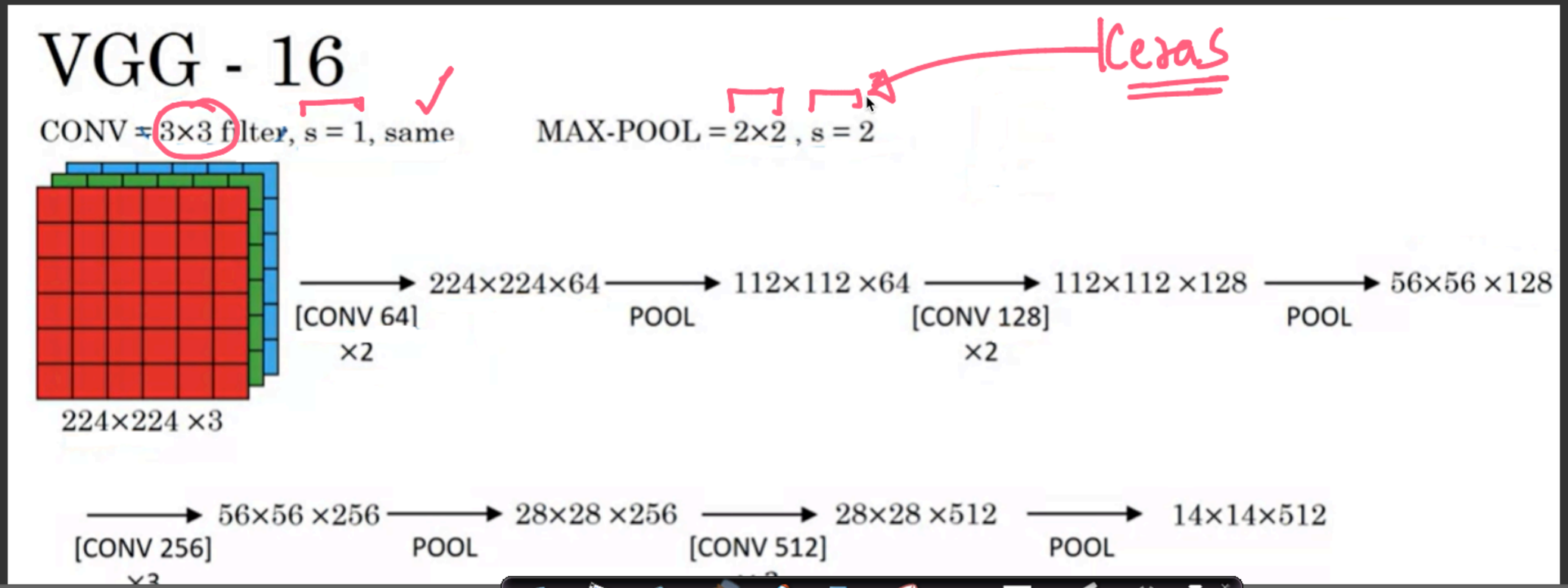
## VGGNet

- VGG19 won second place in 2014
- If we look at the architecture, it is very similar with AlexNet
- **It improves on AlexNet by being much deeper.**
- VGG16 had 16 layers(13 convolution layers(CONV) +3 Fully connected (FC)).
- VGG19 has 19 layers(16 CONV +3 FC).

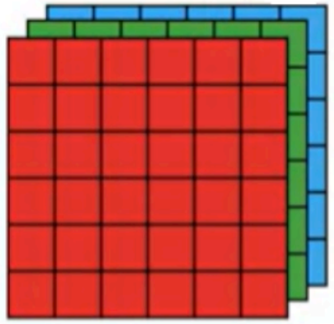


## VGGNet

- VGG19 won second place in 2014
- If we look at the architecture, it is very similar with AlexNet
- **It improves on AlexNet by being much deeper.**
- VGG16 had 16 layers(13 convolution layers(CONV) +3 Fully connected (FC)).
- VGG19 has 19 layers(16 CONV +3 FC).



CONV = 3×3 filter, s = 1, same MAX-POOL = 2×2 , s = 2



224×224 ×3 → [CONV 64] ×2 → 224×224×64 → POOL → 112×112 ×64 → [CONV 128] ×2 → 112×112 ×128 → POOL → 56×56 ×128

→ [CONV 256] ×3 → 56×56 ×256 → POOL → 28×28 ×256 → [CONV 512] × 3 → 28×28 ×512 → POOL → 14×14×512

→ [CONV 512] ×3 → 14×14 ×512 → POOL → 7×7×512 → FC 4096 → FC 4096 → Softmax 1000

Figure: : VGG16 Architecture

| Layer | Filter/neurons | Filter size | Stride | Padding | Size of feature map |
|---|---|---|---|---|---|

| | | | | | |
|---|---|---|---|---|---|
| FC | | | | | 1000 |

Figure: VGG16 layers Computation

- **If we consider both architectures VGG and Alexnet, What are the differences b/w them ?**
  - It was uses only 3x3 filters, whereas Alexnet had 11x11, 3x3 and 5x5 filters
- **Which one is better: a 5x5 convolutional filter or two 3x3 filters applied in sequence?**
  - The difference is that two 3x3 filters applied in sequence have a total of 2 * 3 * 3 = 18 learnable parameters whereas a single 5x5 filter has 5 * 5 = 25 learnable weights. So, **two 3x3 filters has less parameter and so it is less computationally expensive.**
  - Two 3x3 filters has more non-linearity than 5x5 filters due to the fact that activation function has been applied 2 times. Having more non-linearity means the neural network is able to learn more complex nonlinear representation of input.
- VGG16 has **top 5 acc of 91.9 and top 1 acc of 74.4 on ImageNet dataset** which is way better than AlexNet Click to know more

| Model Name | Number of params | Top 1 Acc | Top 5 Acc |
|---|---|---|---|
| Alexnet | 60M | 63.3 | 84.6 |
| VGG16 | 138M | 74.4 | 91.9 |
| VGG19 | 144M | 74.5 | 92.0 |

## Quiz-4

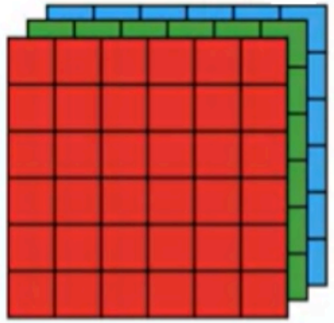
Increase the **number** of a convolutional layers would necessarily increase the performance of a convolutional neural network.

> (a) TRUE

> (b) FALSE

# VGG - 16

CONV = 3×3 filter, s = 1, same MAX-POOL = 2×2 , s = 2



224×224 ×3

→ [CONV 64] ×2 → 224×224×64 → POOL → 112×112 ×64 → [CONV 128] ×2 → 112×112 ×128 → POOL → 56×56 ×128

→ [CONV 256] ×3 → 56×56 ×256 → POOL → 28×28 ×256 → [CONV 512] × 3 → 28×28 ×512 → POOL → 14×14×512

→ [CONV 512] ×3 → 14×14 ×512 → POOL → 7×7×512 → FC 4096 → FC 4096 → Softmax 1000

16 MM

102 MM

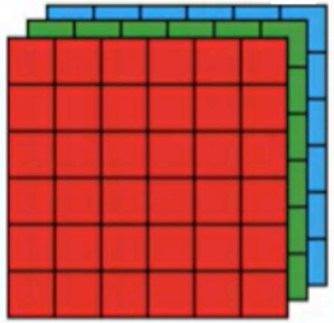
118M

138 MM

Figure: : VGG16 Architecture

# VGG - 16

CONV = 3×3 filter, s = 1, same MAX-POOL = 2×2 , s = 2



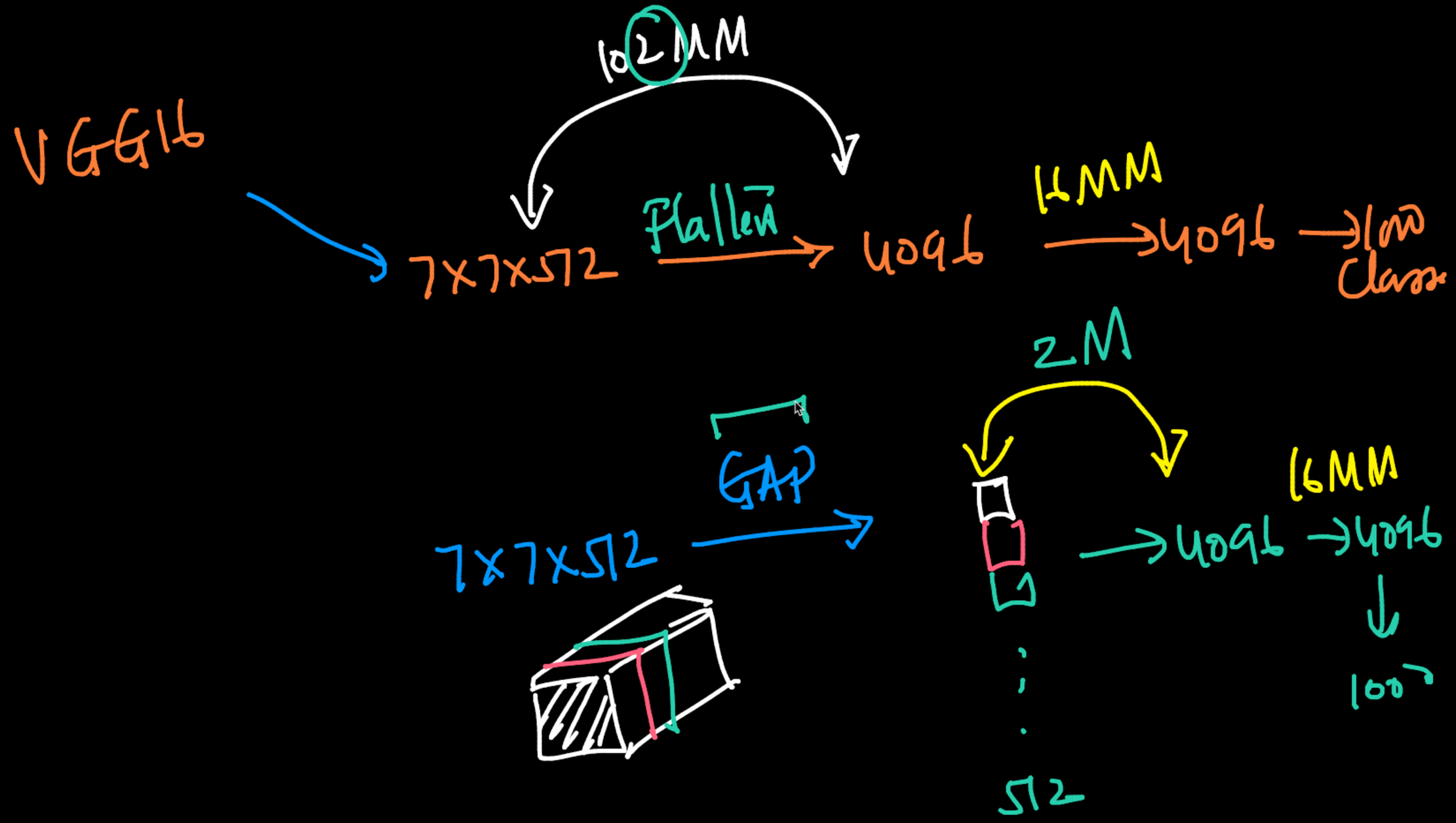
224×224 ×3 → [CONV 64] ×2 → 224×224×64 → POOL → 112×112 ×64 → [CONV 128] ×2 → 112×112 ×128 → POOL → 56×56 ×128

→ [CONV 256] ×3 → 56×56 ×256 → POOL → 28×28 ×256 → [CONV 512] × 3 → 28×28 ×512 → POOL → 14×14×512

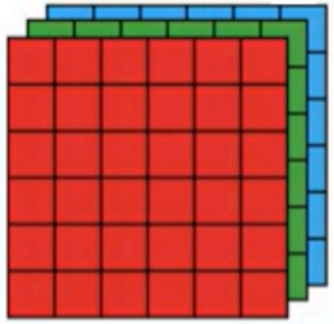
→ [CONV 512] ×3 → 14×14 ×512 → POOL → 7×7×512 → FC 4096 → FC 4096 → Softmax 1000

GAP

Figure: : VGG16 Architecture



CONV = 3×3 filter, s = 1, same MAX-POOL = 2×2 , s = 2



224×224 ×3 → [CONV 64] ×2 → 224×224×64 → POOL → 112×112 ×64 → [CONV 128] ×2 → 112×112 ×128 → POOL → 56×56 ×128

→ [CONV 256] ×3 → 56×56 ×256 → POOL → 28×28 ×256 → [CONV 512] × 3 → 28×28 ×512 → POOL → 14×14×512

→ [CONV 512] ×3 → 14×14 ×512 → POOL → 7×7×512 → FC 4096 → FC 4096 → Softmax 1000

AlexNet 60MM → VGG16 138M

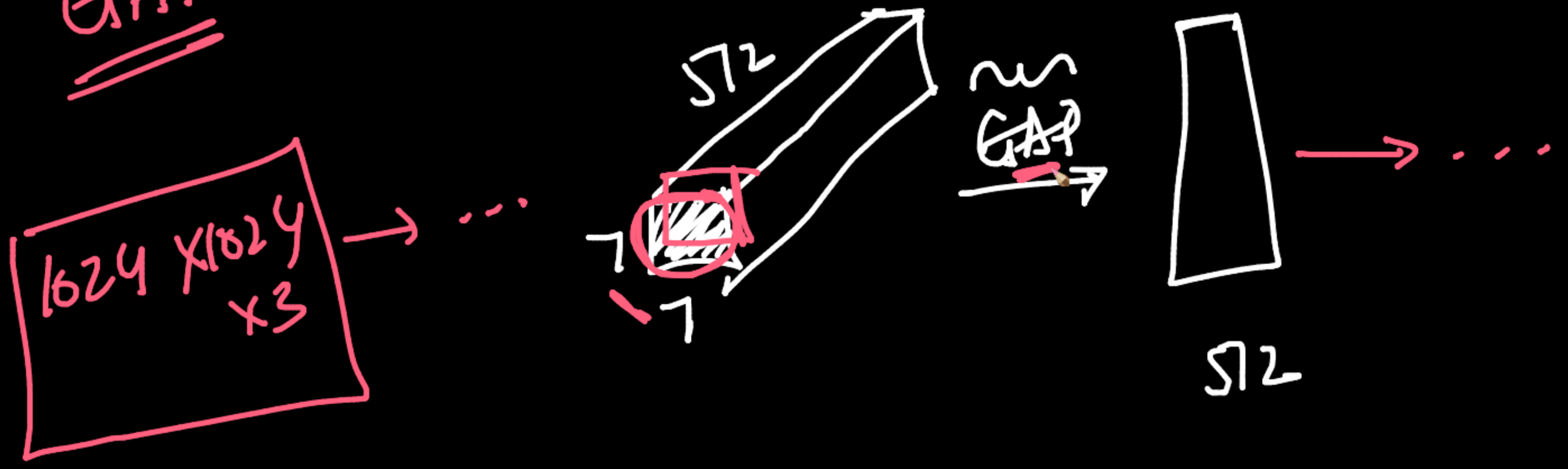
loss

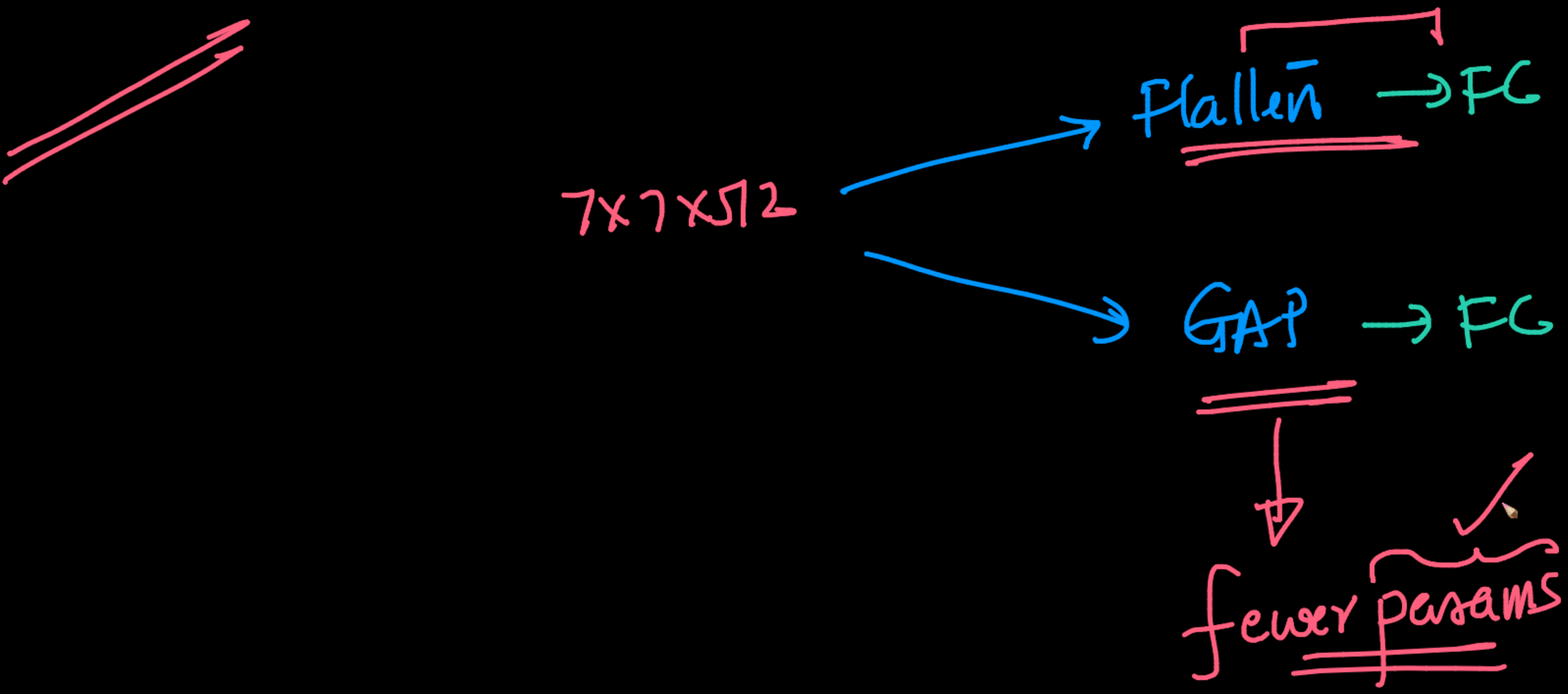
Hyperparameters

Figure: : VGG16 Architecture

| Layer | Filter/neurons | Filter size | Stride | Padding | Size of feature map |
|---|---|---|---|---|---|
| Input | | | | | 224x224x3 |
| Conv | 64 | | | | |

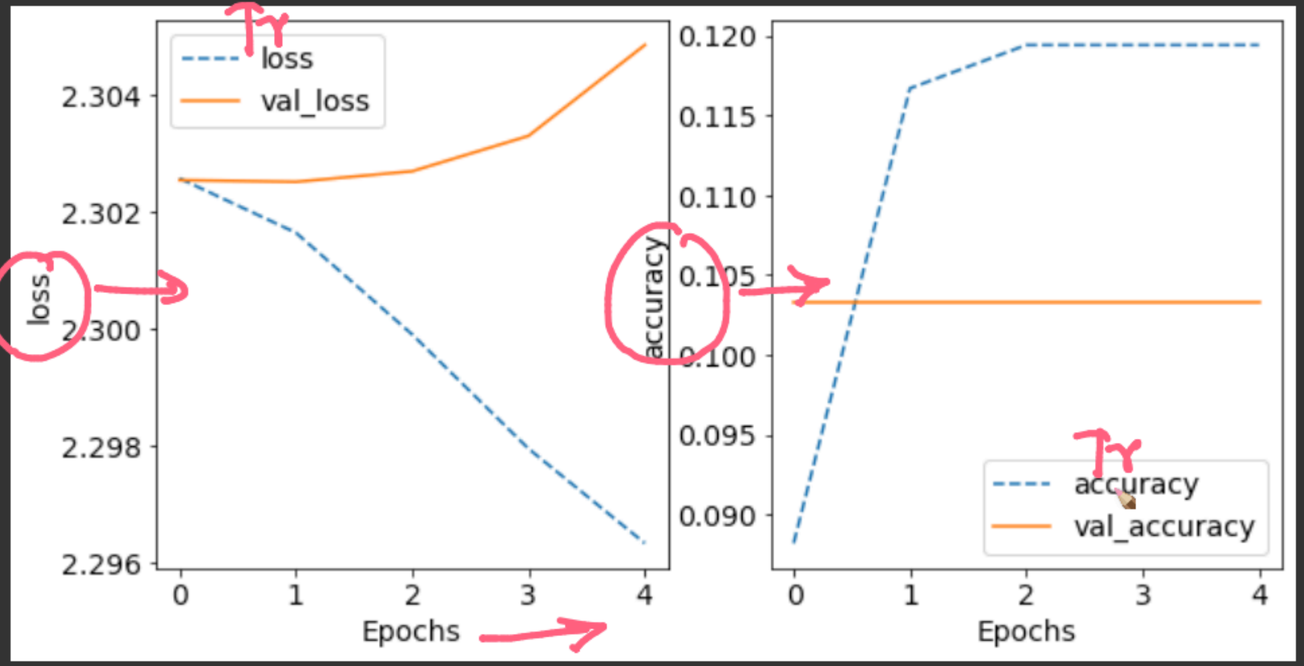
# GAP




7x7x512
Flatten → FC
GAP → FC
fewer params

```
[ ]     ax[idx].legend([metric, 'val_' + metric])
```

```
[ ] training_plot(['loss', 'accuracy'], history)
```

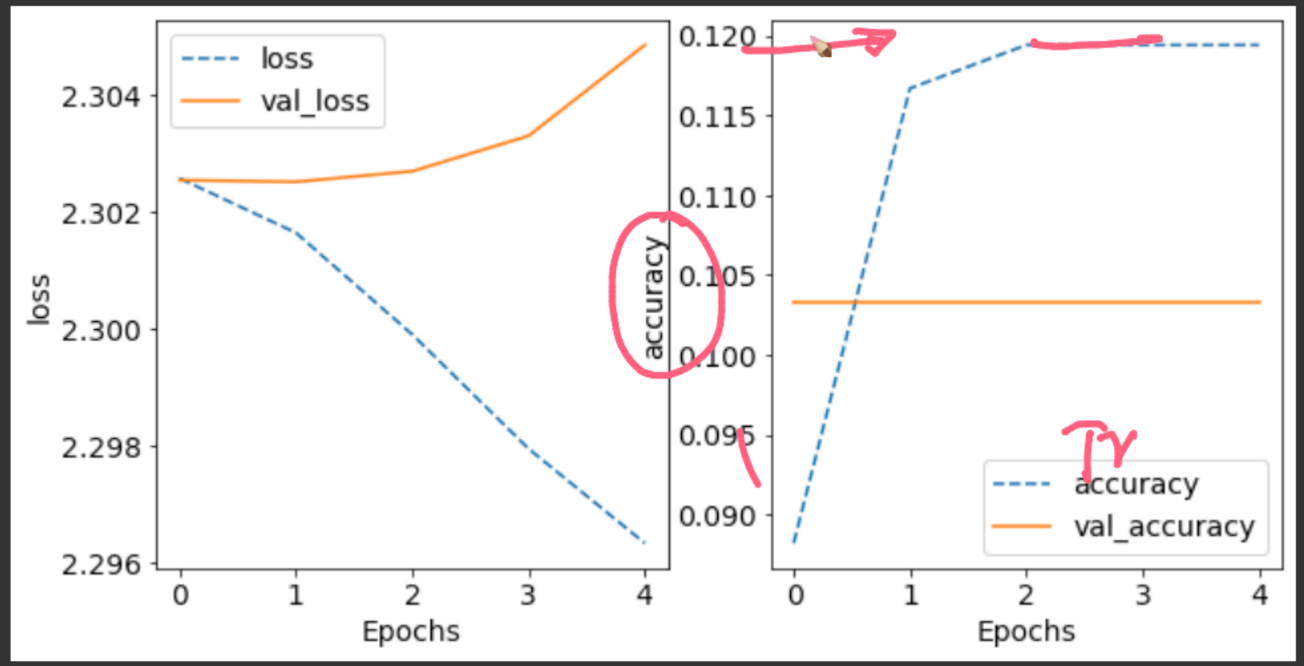



```
[ ] # Evaluate the model
    loss, acc = vgg16_model_scratch.evaluate(test_ds, verbose=2)
    print("Restored model, accuracy: {:5.2f}%".format(100 * acc))
```

```
1/1 - 4s - loss: 2.3041 - accuracy: 0.1163 - 4s/epoch - 4s/step
Restored model, accuracy: 11.63%
```

```
[ ]     ax[idx].legend([metric, 'val_' + metric])
```

```
[ ] training_plot(['loss', 'accuracy'], history)
```

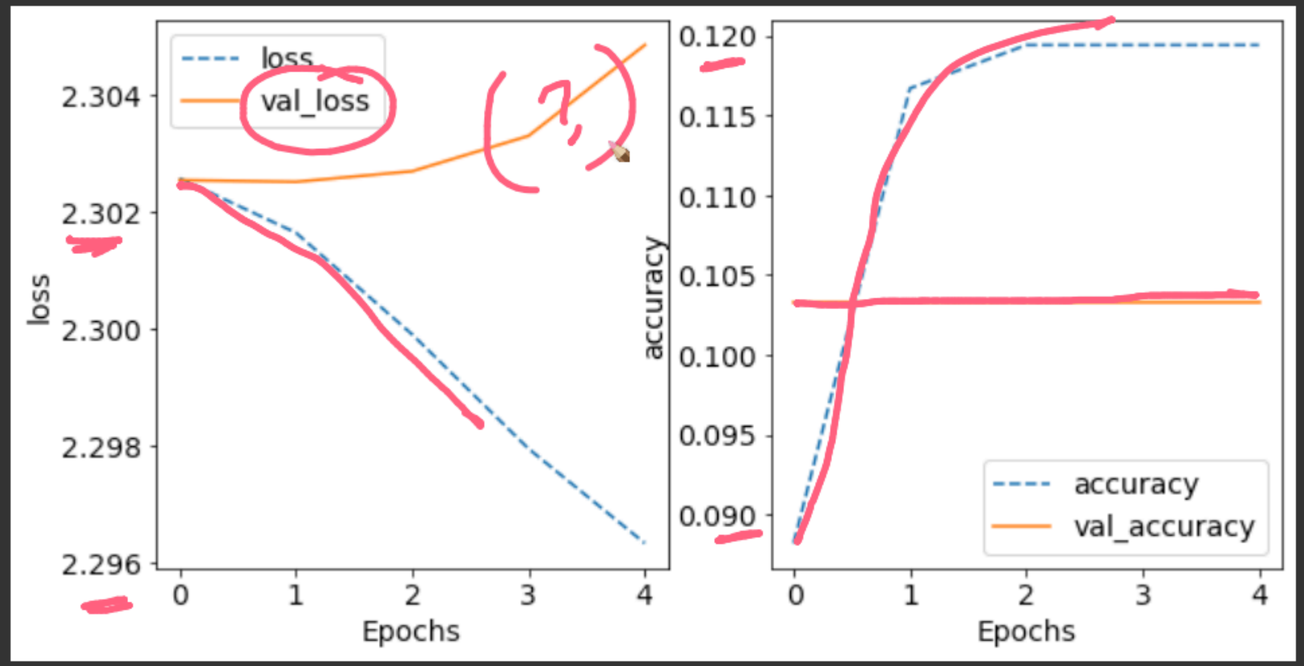



```
[ ] # Evaluate the model
    loss, acc = vgg16_model_scratch.evaluate(test_ds, verbose=2)
    print("Restored model, accuracy: {:5.2f}%".format(100 * acc))
```

```
1/1 - 4s - loss: 2.3041 - accuracy: 0.1163 - 4s/epoch - 4s/step
Restored model, accuracy: 11.63%
```

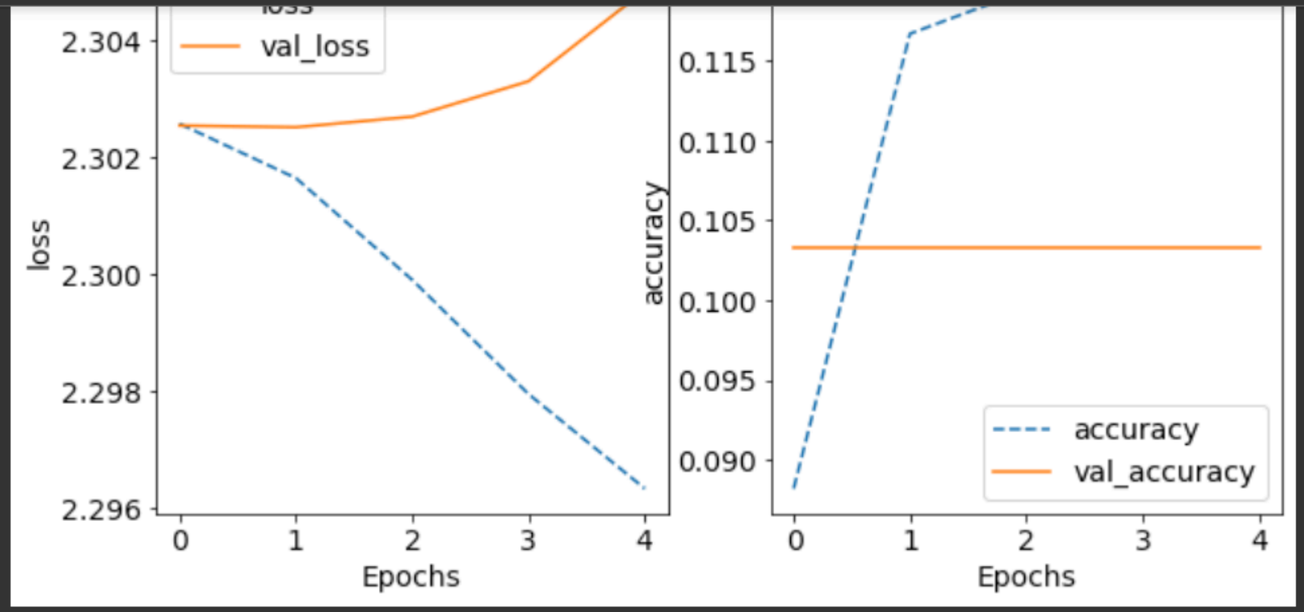
```
[ ]     ax[idx].legend([metric, 'val_' + metric])
```

```
[ ] training_plot(['loss', 'accuracy'], history)
```




```
[ ] # Evaluate the model
    loss, acc = vgg16_model_scratch.evaluate(test_ds, verbose=2)
    print("Restored model, accuracy: {:5.2f}%".format(100 * acc))
```

```
1/1 - 4s - loss: 2.3041 - accuracy: 0.1163 - 4s/epoch - 4s/step
Restored model, accuracy: 11.63%
```




VGG16 : arch

~15 MM params

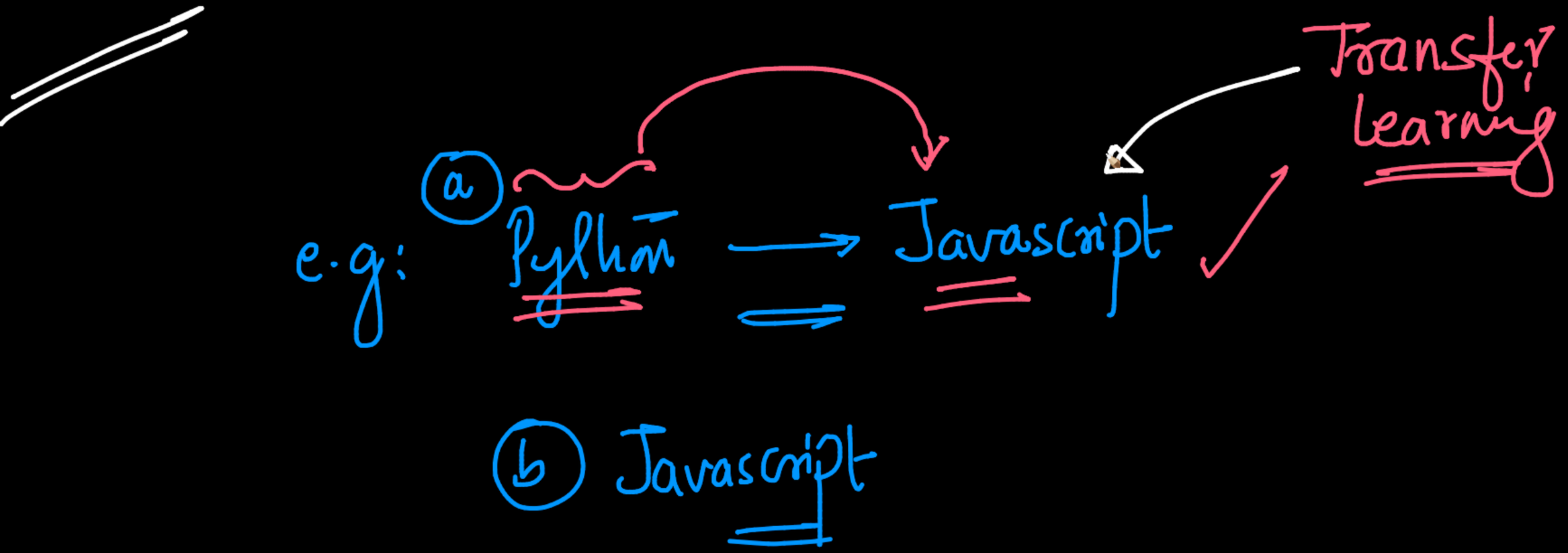
~700 images

```
# Evaluate the model
loss, acc = vgg16_model_scratch.evaluate(test_ds, verbose=2)
print("Restored model, accuracy: {:5.2f}%".format(100 * acc))
```

```
1/1 - 4s - loss: 2.3041 - accuracy: 0.1163 - 4s/epoch - 4s/step
Restored model, accuracy: 11.63%
```

- Our model is not doing good as we are getting only 11.63% accuracy on our test data
- This is because our dataset size is small when it comes to train SOTA CNN model such as VGG from scratch

## How you are going to improve the performance of model with small dataset ?



already trained

Image net data 1.2 MM

→ VGG16 (13 conv, 3 FC) → 1000 class- classifier

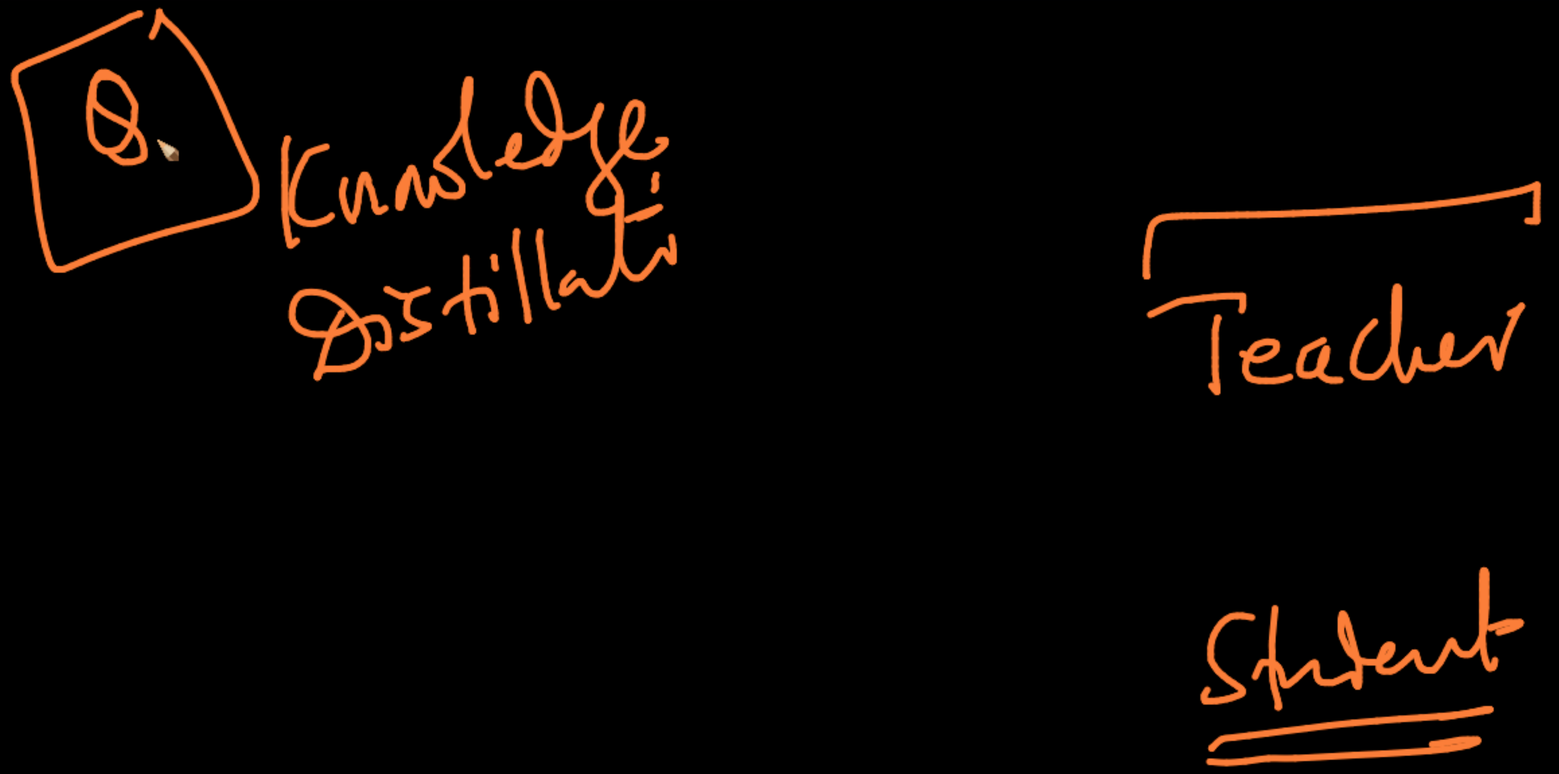
→ edges; objects; shapes..

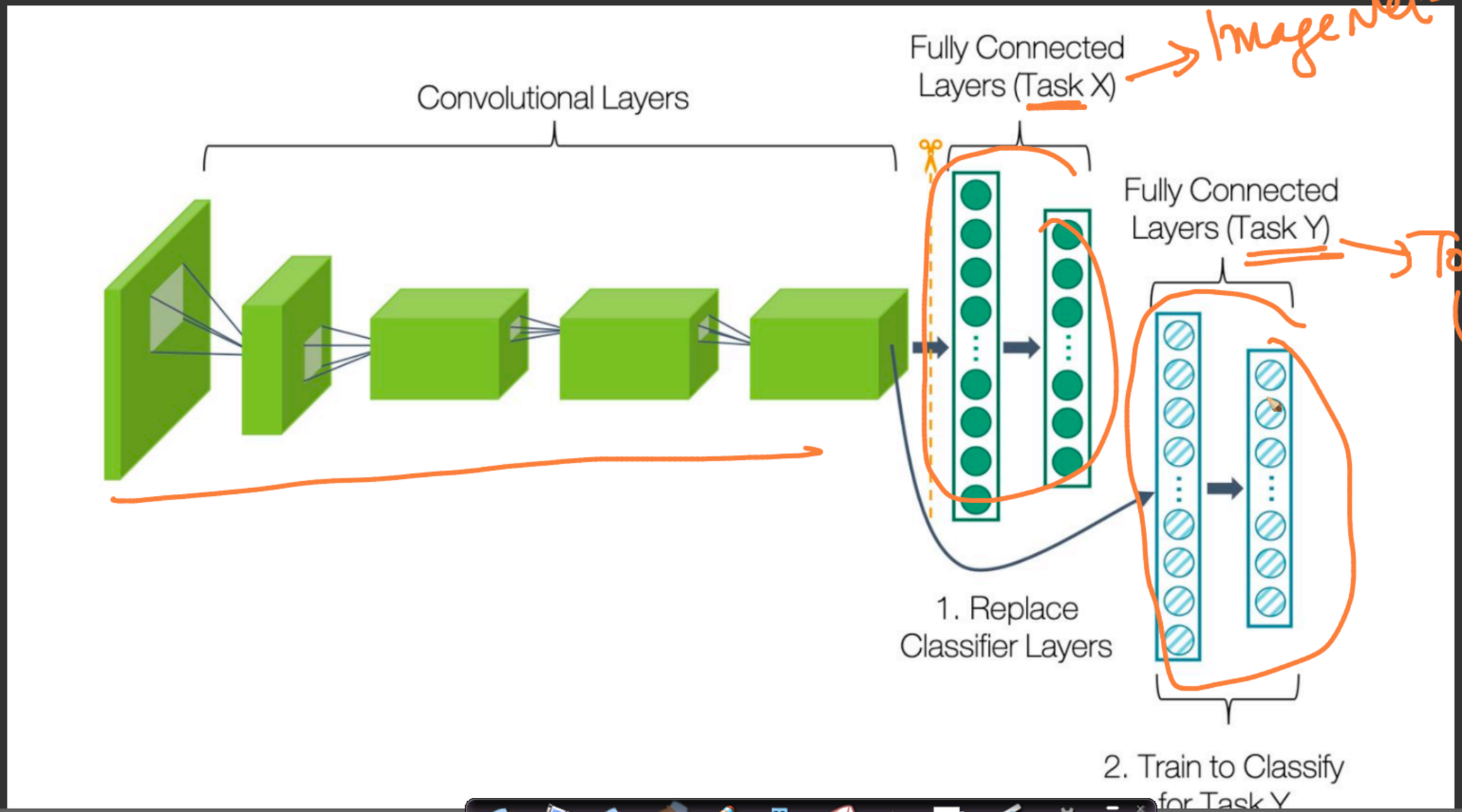
= Transfer learning =

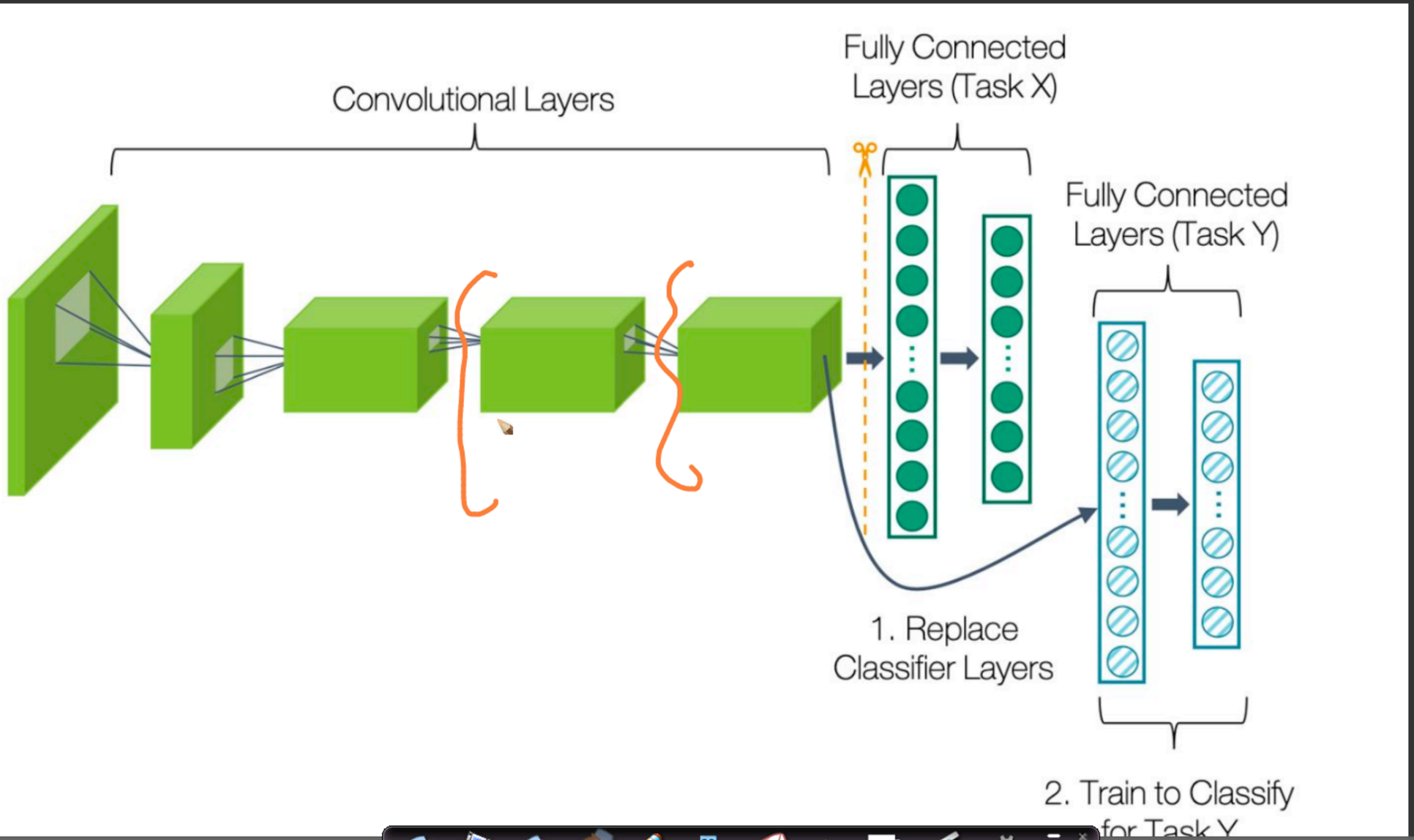
D tourism (700 images) → "reuse" this model

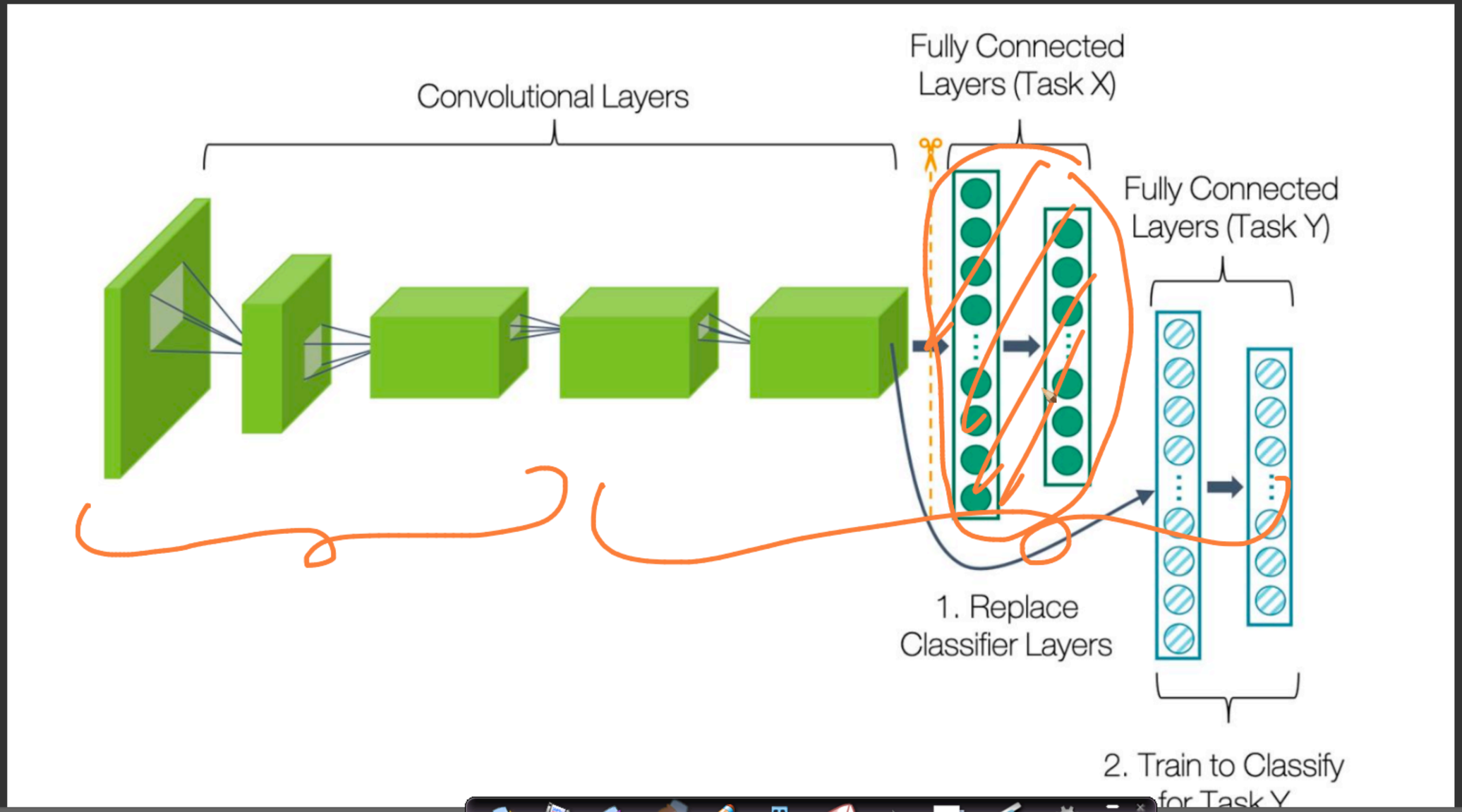
do not change initial layers → non-trainable & fixed

change last few layers (FC) typically few params









```
pretrained_model = tf.keras.applications.VGG16(weights='imagenet', include_top=False, input_shape=[224,224, 3])
pretrained_model.trainable=False
vgg16_model = tf.keras.Sequential([
    pretrained_model,
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(10, activation='softmax')
])
```


```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5
58892288/58889256 [==============================] - 0s 0us/step
58900480/58889256 [==============================] - 0s 0us/step
```

## Quiz-5

How did you lock or freeze a layer from retraining?

(a) tf.freeze(layer)

(b) tf.layer.frozen = True

(c) tf.layer.locked = True

(d) layer.trainable = False

all Conv layers

```
pretrained_model = tf.keras.applications.VGG16(weights='imagenet', include_top=False, input_shape=[224,224, 3])
pretrained_model.trainable=False
vgg16_model = tf.keras.Sequential([
    pretrained_model,
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(10, activation='softmax')
])
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5
58892288/58889256 [==============================] - 0s 0us/step
58900480/58889256 [==============================] - 0s 0us/step
```

## Quiz-5

How did you lock or freeze a layer from retraining?

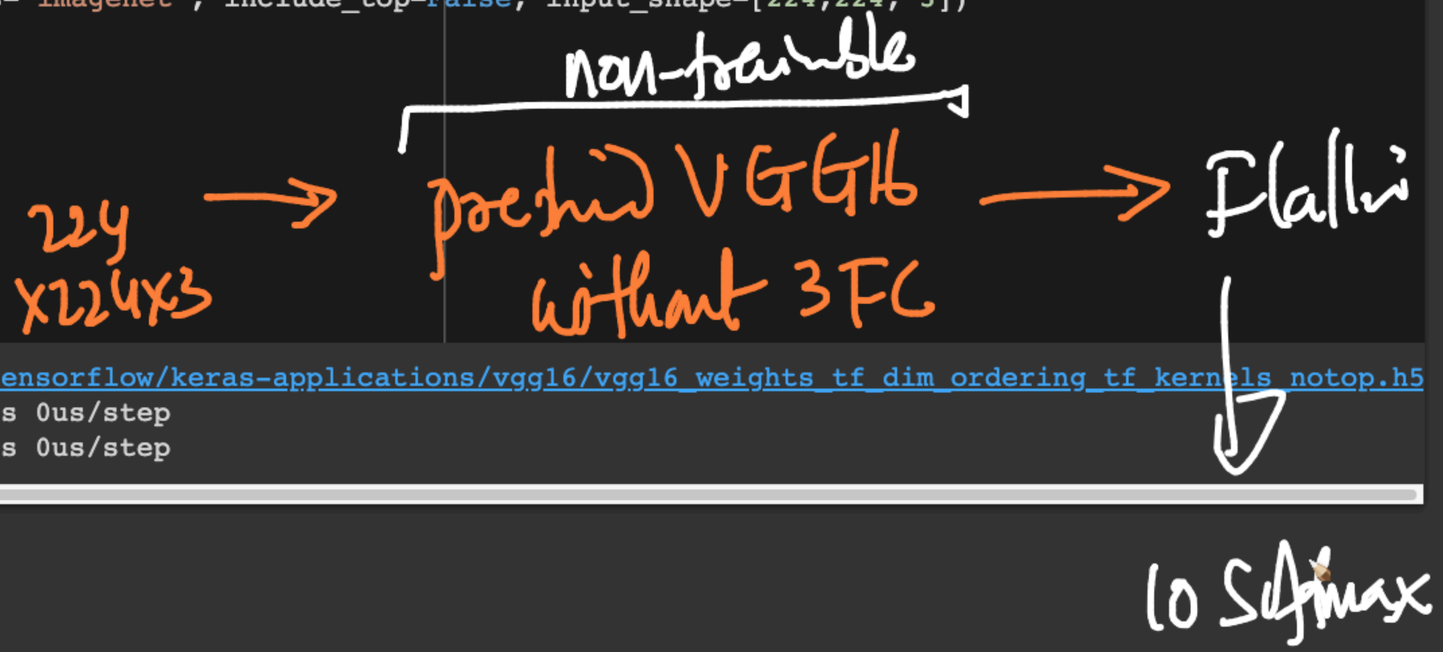
(a) tf.freeze(layer)

(b) tf.layer.frozen = True

(c) tf.layer.locked = True

(d) layer.trainable = False

```
pretrained_model = tf.keras.applications.VGG16(weights='imagenet', include_top=False, input_shape=[224,224, 3])
pretrained_model.trainable=False
vgg16_model = tf.keras.Sequential([
    pretrained_model,
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(10, activation='softmax')
])
```




```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5
58892288/58889256 [==============================] - 0s 0us/step
58900480/58889256 [==============================] - 0s 0us/step
```

## Quiz-5

How did you lock or freeze a layer from retraining?

(a) tf.freeze(layer)

(b) tf.layer.frozen = True

(c) tf.layer.locked = True

(d) layer.trainable = False

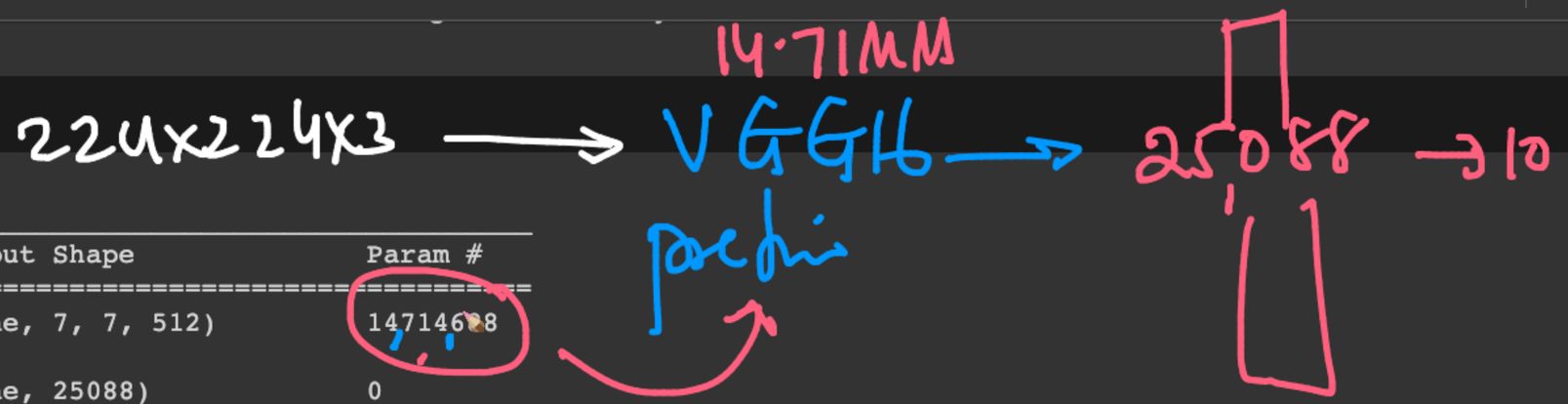
```
[ ] vgg16_model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 vgg16 (Functional)           (None, 7, 7, 512)         14714688

 flatten_1 (Flatten)          (None, 25088)             0

 dense_1 (Dense)              (None, 10)                250890

=================================================================
Total params: 14,965,578
Trainable params: 250,890
Non-trainable params: 14,714,688
_________________________________________________________________
```




```
[ ] import functools
    top5_acc = functools.partial(tf.keras.metrics.SparseTopKCategoricalAccuracy())

    vgg16_model.compile(
        optimizer='adam',
        loss = 'sparse_categorical_crossentropy',
        metrics=['accuracy']
    )
```

```
[ ] history = vgg16_model.fit(train_ds, epochs=5,
```

```
[ ] vgg16_model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 vgg16 (Functional)           (None, 7, 7, 512)         14714688

 flatten_1 (Flatten)          (None, 25088)             0

 dense_1 (Dense)              (None, 10)                250890

=================================================================
Total params: 14,965,578
Trainable params: 250,890
Non-trainable params: 14,714,688
_________________________________________________________________
```

VGG16: 13 ConV


```
[ ] import functools
    top5_acc = functools.partial(tf.keras.metrics.SparseTopKCategoricalAccuracy())

    vgg16_model.compile(
        optimizer='adam',
        loss = 'sparse_categorical_crossentropy',
        metrics=['accuracy']
    )
```

```
[ ] history = vgg16_model.fit(train_ds, epochs=5,
```

```
vgg16_model.compile(
    optimizer='adam',
    loss = 'sparse_categorical_crossentropy',
    metrics=['accuracy']
)
```

```
[ ] history = vgg16_model.fit(train_ds, epochs=5,
                              validation_data=val_ds)
```

```
Epoch 1/5
6/6 [==============================] - 7s 837ms/step - loss: 2.4012 - accuracy: 0.2456 - val_loss: 1.6077 - val_accuracy: 0.5871
Epoch 2/5
6/6 [==============================] - 6s 826ms/step - loss: 1.0682 - accuracy: 0.7001 - val_loss: 0.7705 - val_accuracy: 0.7355
Epoch 3/5
6/6 [==============================] - 7s 818ms/step - loss: 0.4569 - accuracy: 0.8711 - val_loss: 0.6433 - val_accuracy: 0.8258
Epoch 4/5
6/6 [==============================] - 6s 819ms/step - loss: 0.3141 - accuracy: 0.9267 - val_loss: 0.4948 - val_accuracy: 0.8129
Epoch 5/5
6/6 [==============================] - 6s 826ms/step - loss: 0.1970 - accuracy: 0.9579 - val_loss: 0.5019 - val_accuracy: 0.8387
```

- **By training VGG19 using transfer learning we were able to obtain validation accuracy of 0.8452 after 5 epochs where as VGGNet created from scratch had validation accuracy of 0.15**
- Hence using transfer learnng we obtain higher accuracy and require less training

Transfer learning

No Transfer learning VGG16 + FC

```
[ ] # Save the entire model as a SavedModel.
    # !mkdir -p saved_vgg16model_pretrained
    # vgg16_model.save('saved_vgg16model_pretrained/vgg16_model')
```

Multiple architectures → innovation → why?

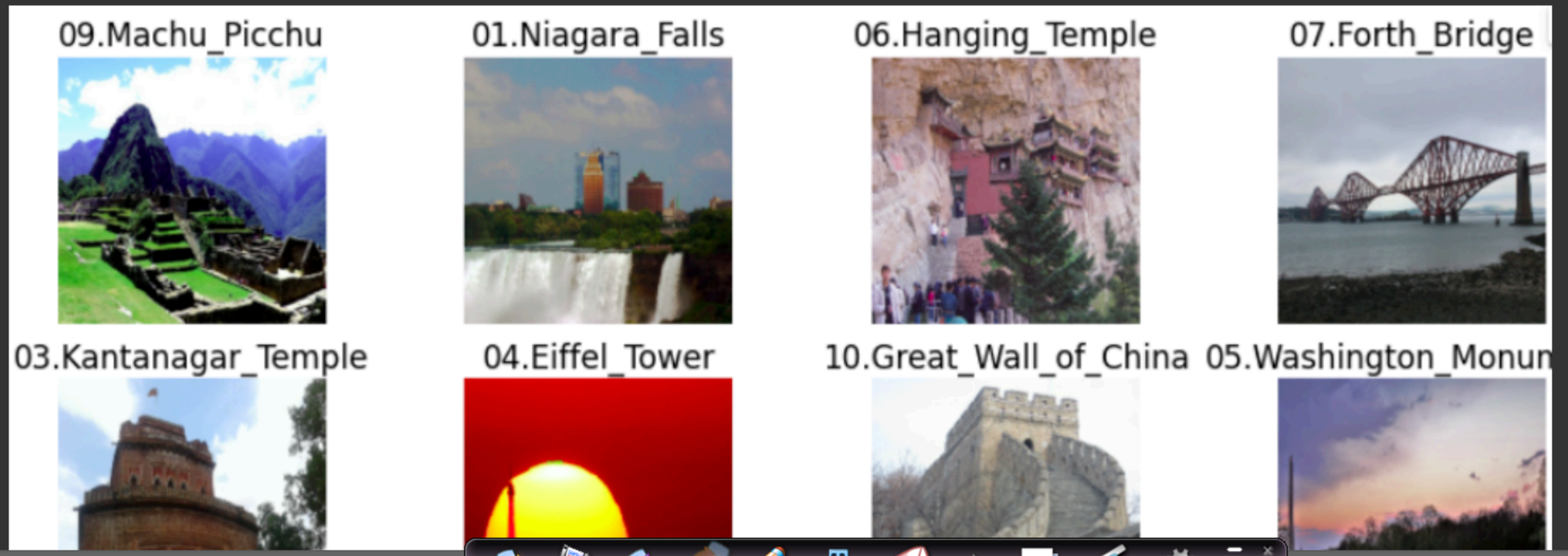
Transfer learning

**Agenda:**

- Recognition of world famous landmark
- Standard CNN architecture like AlexNet and VGGNet
- Transfer learning using VGGNet

## Problem Statement:

- Suppose you are working as Data Scientist in Google. You have been given with a task to automate the categorization of famous landmarks.



# TensorFlow.js is a library for machine learning in JavaScript

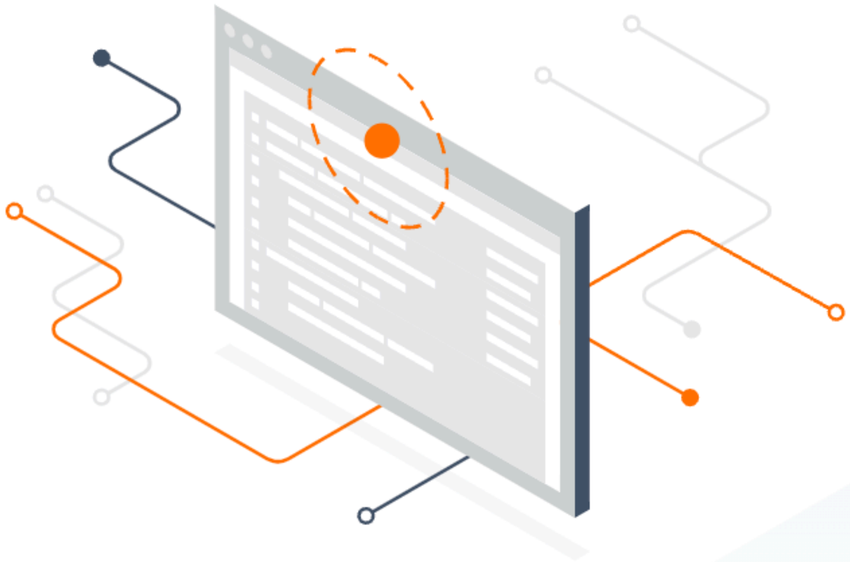
Develop ML models in JavaScript, and use ML directly in the browser or in Node.js.

See tutorials | See models | See demos

Tutorials show you how to use TensorFlow.js with complete, end-to-end examples.

Pre-trained, out-of-the-box models for common use cases.

Live demos and examples run in your browser using TensorFlow.js.



## How it works