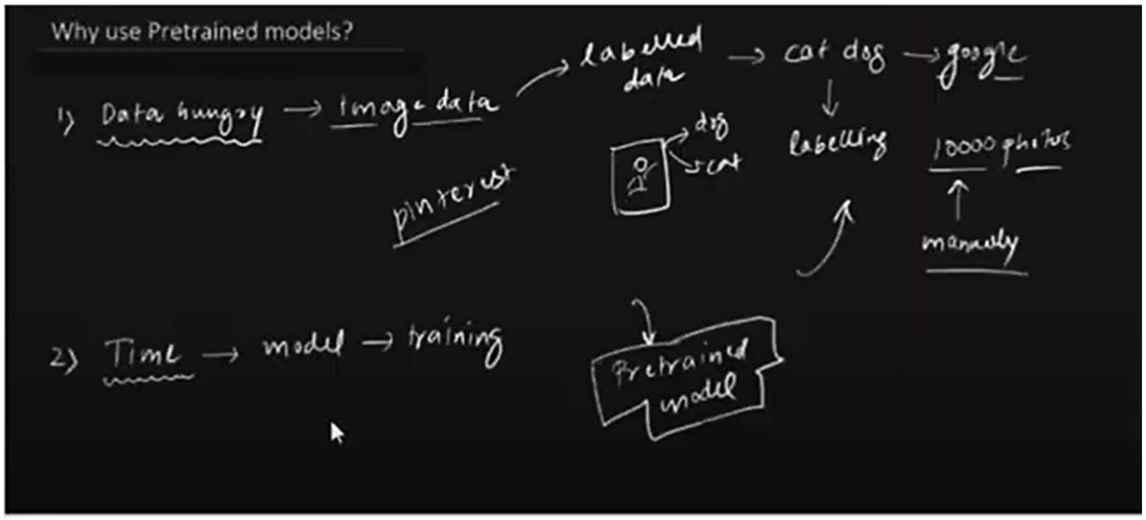
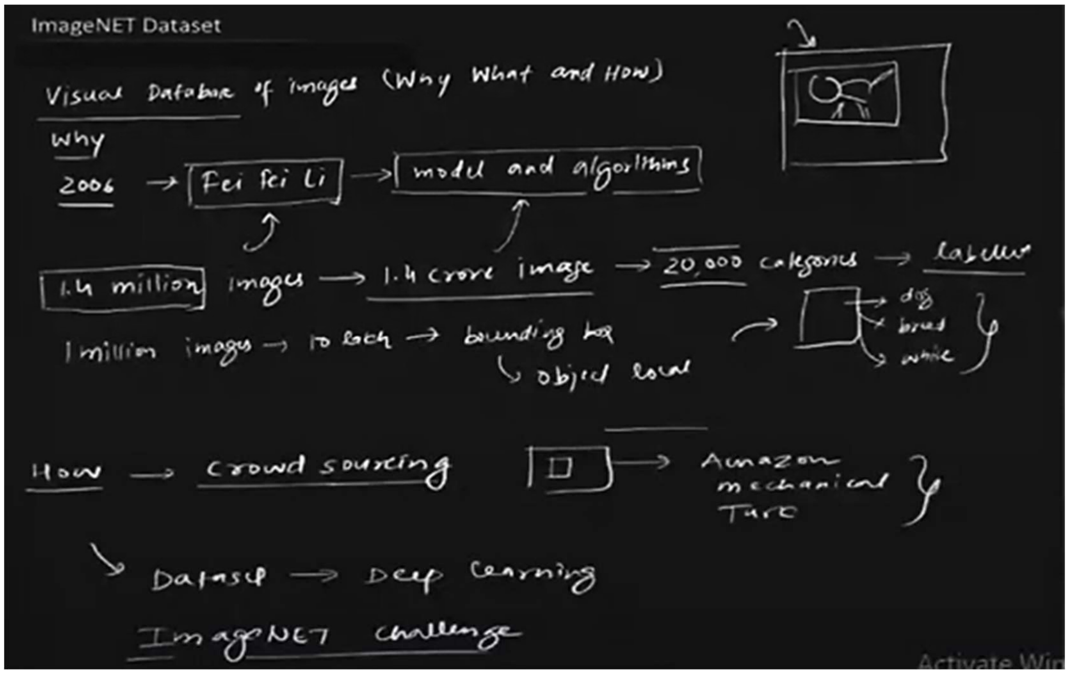
Pre-Trained Models

Why we use pre-trained models?

First problem is CNN models are data hungry models to perform accurately. Second problem is we need label data. (as we take image from google, it is not labelled it need to be done manually) – so it take so much time and need so much computational power. So to resolve this we use pre-trained models.


Why use Pretrained models?
1) Data hungry → Image data
labelled data → cat dog → google
labelling
dog
cat
10000 photos
manually
pinterest
2) Time → model → training
Pretrained model

In 2006, Fei Fei Li collect images data – known as imagenet dataset. He collect 1.4 million images at that time that are now currently at 1.4 crore images. He labelled 20,00 categories (classes) with its bread, and provide **bounding boxes** to 1 million images – which helps in **object localization** used cloud sourcing of Amazon.


ImageNET Dataset
Visual Database of images (Why What and HOW)
Why
2006 → Fei Fei Li → model and algorithms
1.4 million images → 1.4 crore image → 20,000 categories → labeller
dog
breed
white
1 million images → to each → bounding box
object local
How → crowd sourcing → Amazon mechanical Turk
Dataset → Deep learning
ImageNET challenge

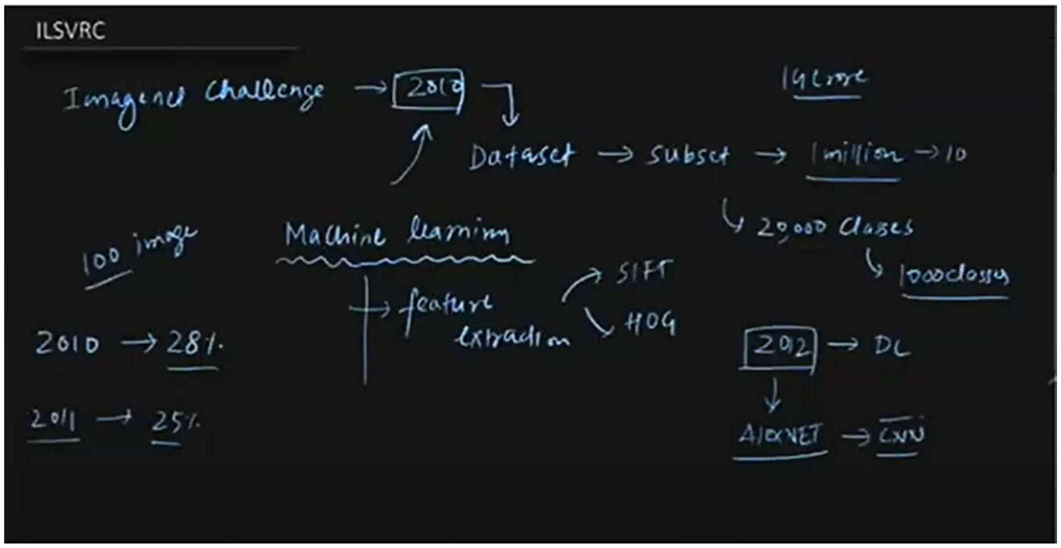
Some definition:

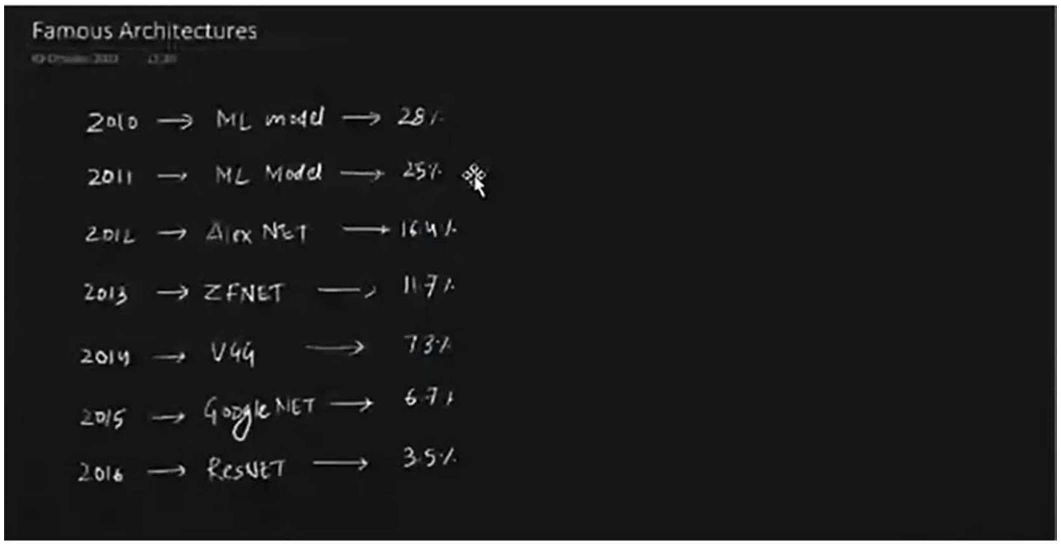
Bounding boxes: A bounding box is a rectangular border used to define the position and extent of an object in an image.

Object location: Object localization is the process of determining the precise location and boundaries of an object within an image, typically by predicting a bounding box that encloses the object.

Imagenet challenge starts in 2010, in which subset of that data that is 1 million images used in this to do feature extraction.

First time, it is observed that 28% error in 2010 in image classification. In 2012, first time deep learning methodology used for solving imagenet dataset. In that, first time CNN artitcture ALEXNET used along using GPU first time in the world, and the error moves from 25% to 16% shows a drastic change.

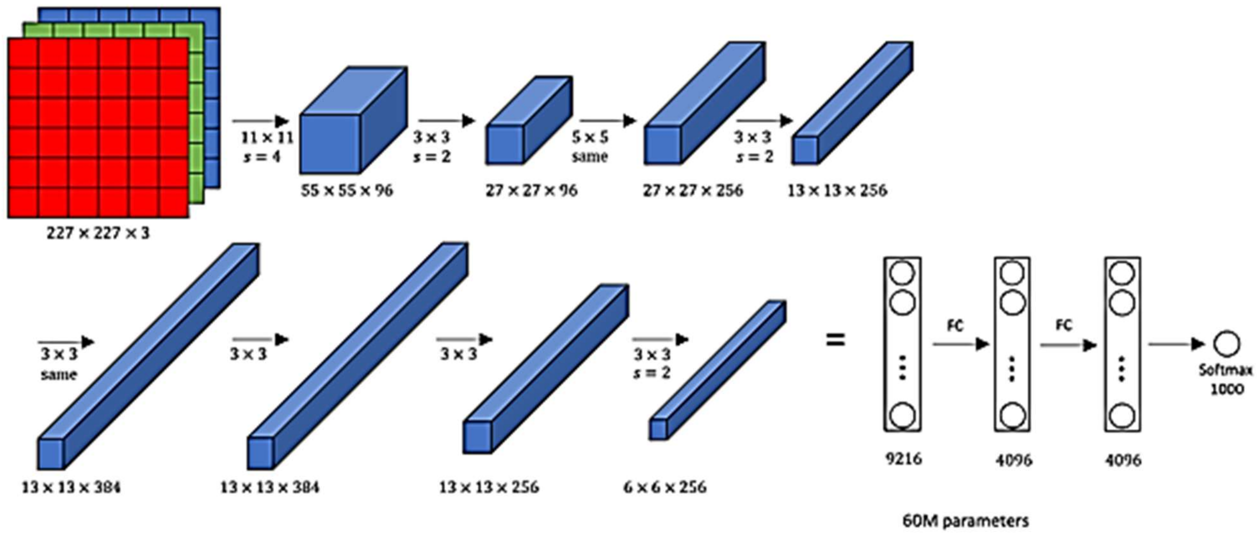



Famous Architectures

2010 → ML model → 28%
2011 → ML Model → 25%
2012 → AlexNET → 16.4%
2013 → ZFNET → 11.7%
2014 → VGG → 7.3%
2015 → GoogleNET → 6.7%
2016 → ResNET → 3.5%

AlexNet Architecture

AlexNet was the first convolutional network which used GPU to boost performance. 1. AlexNet architecture consists of 5 convolutional layers, 3 max-pooling layers, 2 normalization layers, 2 fully connected layers, and 1 softmax layer.

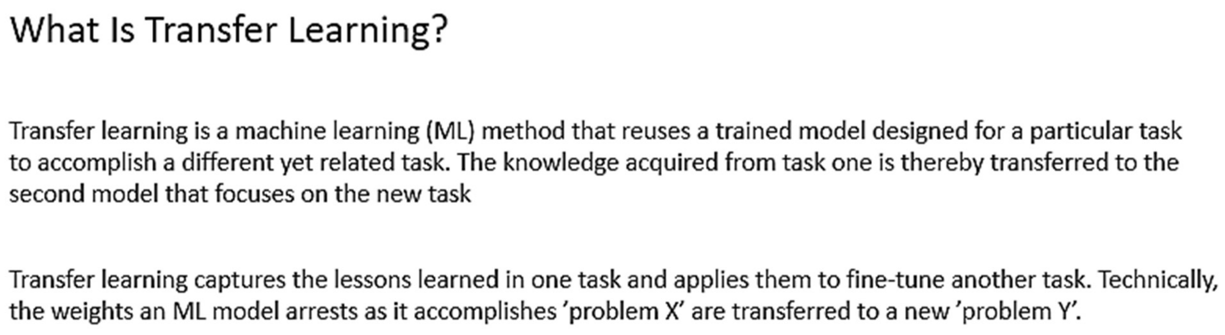


Key Features:

- ‘ReLU’ is used as an activation function rather than ‘tanh’
- Batch size of 128
- SGD Momentum is used as a learning algorithm
- Data Augmentation is been carried out like flipping, jittering, cropping, colour normalization, etc.

Transfer Learning

Pre-trained models are trained on imagenet dataframe, so it give results only on that dataset not on actual real dataset. So to resolve this we use transfer learning.



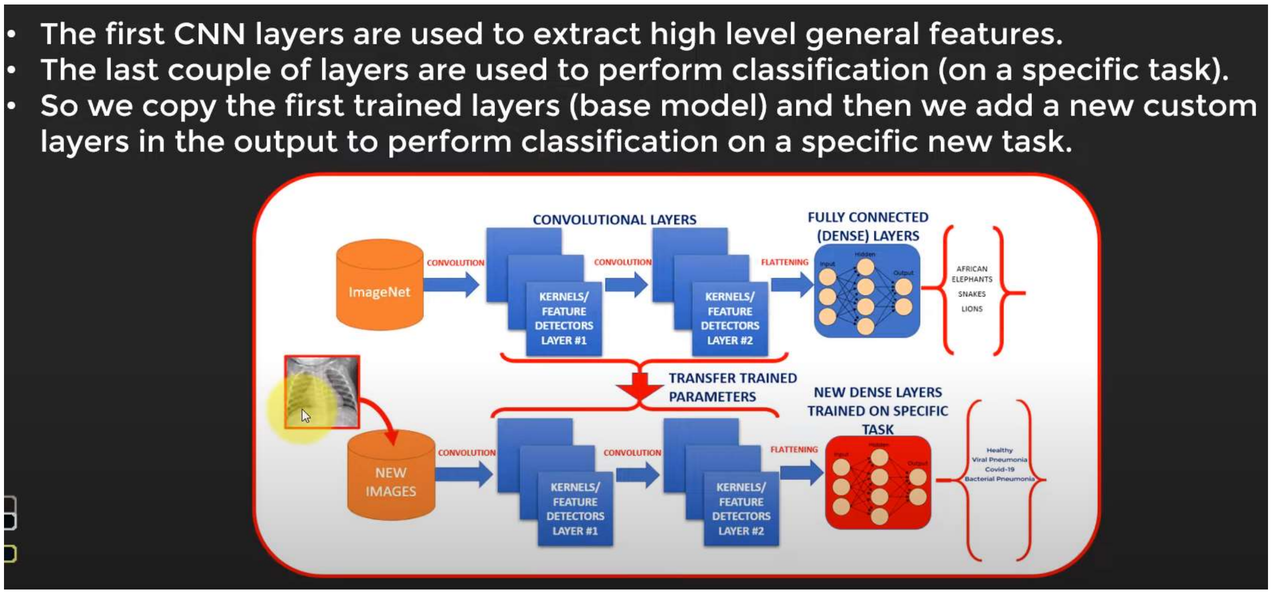
What Is Transfer Learning?

Transfer learning is a machine learning (ML) method that reuses a trained model designed for a particular task to accomplish a different yet related task. The knowledge acquired from task one is thereby transferred to the second model that focuses on the new task

Transfer learning captures the lessons learned in one task and applies them to fine-tune another task. Technically, the weights an ML model arrests as it accomplishes ’problem X’ are transferred to a new ’problem Y’.

Transfer learning is a machine learning technique where a pre-trained model on a large dataset is fine-tuned or adapted to perform a related task on a different, often smaller, dataset, leveraging the knowledge gained from the initial training.

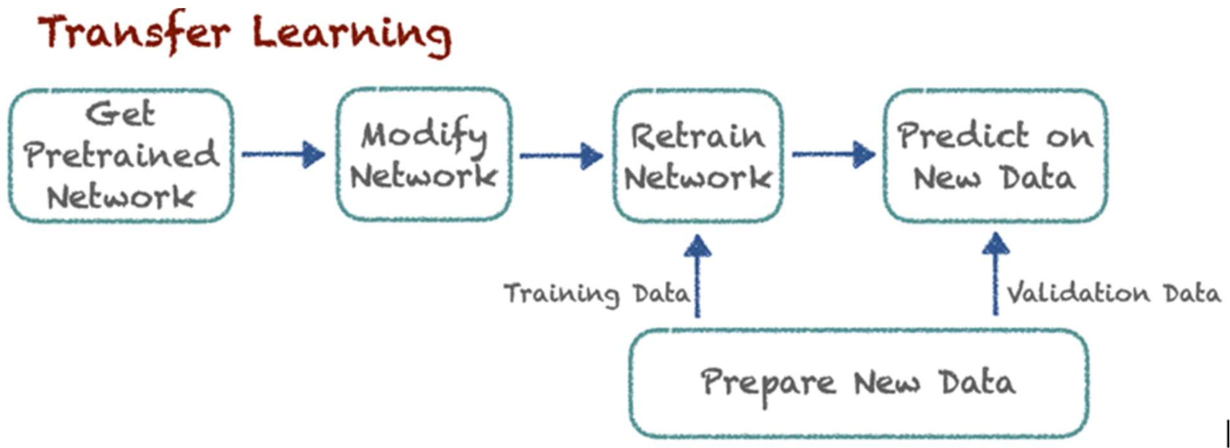
Transfer Learning Process



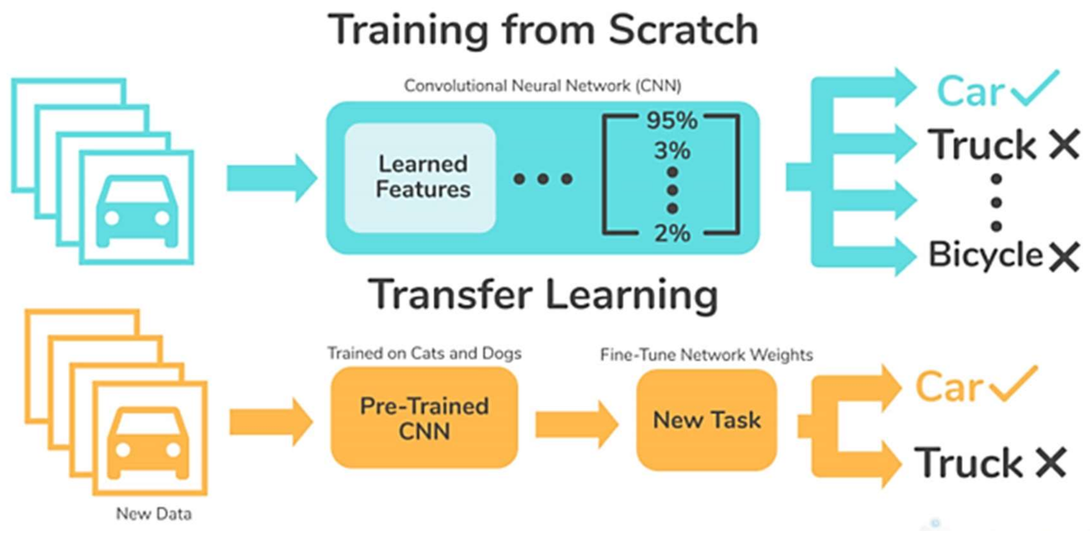
- The first CNN layers are used to extract high level general features.
- The last couple of layers are used to perform classification (on a specific task).
- So we copy the first trained layers (base model) and then we add a new custom layers in the output to perform classification on a specific new task.

CONVOLUTIONAL LAYERS
FULLY CONNECTED (DENSE) LAYERS
ImageNet
CONVOLUTION
KERNELS/ FEATURE DETECTORS LAYER #1
CONVOLUTION
KERNELS/ FEATURE DETECTORS LAYER #2
FLATTENING
Input
Hidden
Output
AFRICAN ELEPHANTS
SNAKES
LIONS
TRANSFER TRAINED PARAMETERS
NEW DENSE LAYERS TRAINED ON SPECIFIC TASK
NEW IMAGES
CONVOLUTION
KERNELS/ FEATURE DETECTORS LAYER #1
CONVOLUTION
KERNELS/ FEATURE DETECTORS LAYER #2
FLATTENING
Healthy
Viral Pneumonia
Covid-19
Bacterial Pneumonia

In transfer learning, we take the initial layers of a pre-trained model (which capture general features) and add new layers on top to create a new model. These added layers are specifically trained on a new task or dataset, while the initial layers can be either frozen (kept unchanged) or fine-tuned (slightly adjusted) depending on the requirements of the task.



Example:



Training from Scratch:

Process:

- Start with raw, untrained data.
- A CNN model is built and trained from the ground up.
- The model learns features and patterns in the data during training.
- After training, the model classifies the data based on the features it has learned.

Transfer Learning:

Process:

- Start with a pre-trained model (a CNN that has already been trained on a large dataset, like "Cats and Dogs").
- The model has already learned useful features from the previous dataset.
- New layers are added, or the existing model is fine-tuned for a new, related task (like classifying cars).
- The model is then trained on the new task, with much less data and computational power required than training from scratch**.**

Key Takeaways:

Training from Scratch is resource-intensive but necessary when a suitable pre-trained model isn't available.

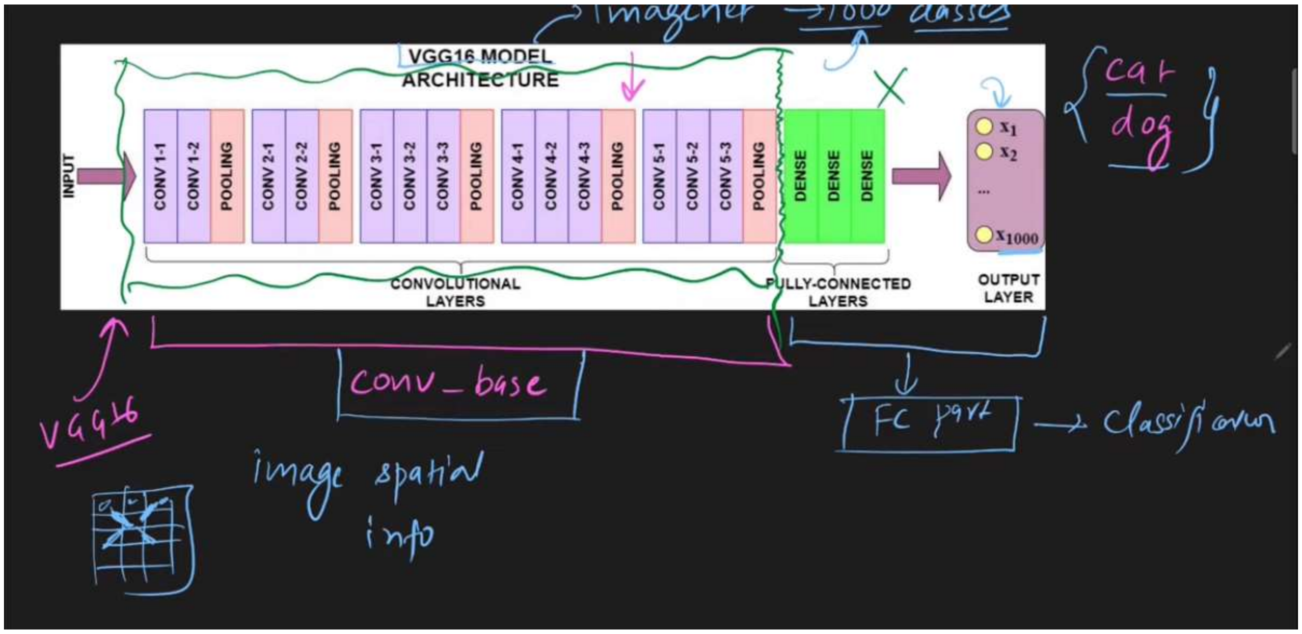
Transfer Learning leverages existing knowledge from a pre-trained model, saving time and computational power while often improving performance on the new task.

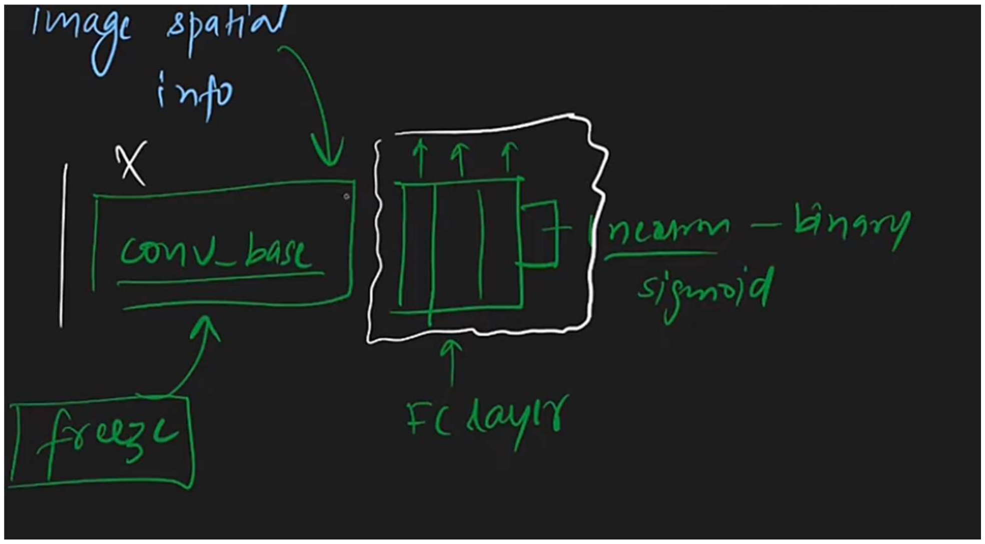
VGG 16 Model:

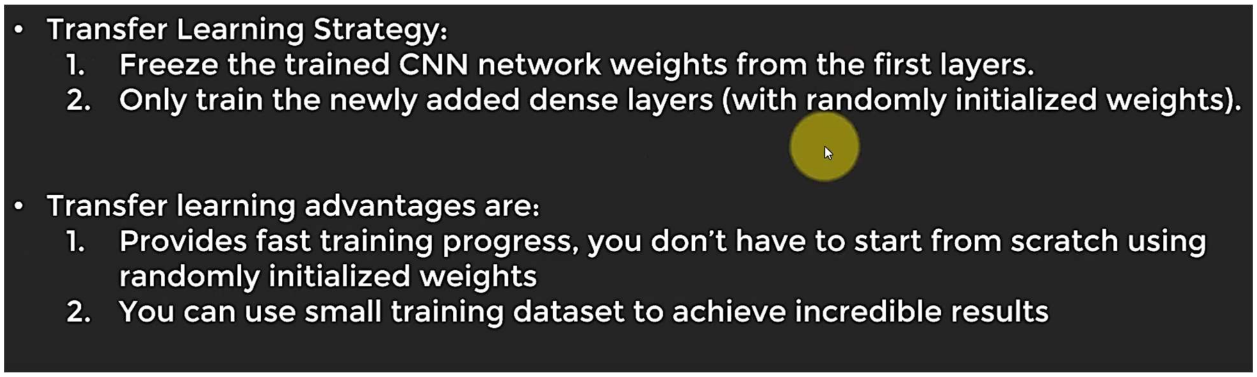
There are two parts Convolutional base and fully connected part.

- Conv-base part used for special / complex features like for edges detection.
- Fully connected part is for classification

So in pre-trained model we take conv-base part as it as (means freeze this part) and create fully connected part as required.







- Transfer Learning Strategy:
  1. Freeze the trained CNN network weights from the first layers.
  2. Only train the newly added dense layers (with randomly initialized weights).

- Transfer learning advantages are:
  1. Provides fast training progress, you don't have to start from scratch using randomly initialized weights
  2. You can use small training dataset to achieve incredible results

To adapt to the new dataset

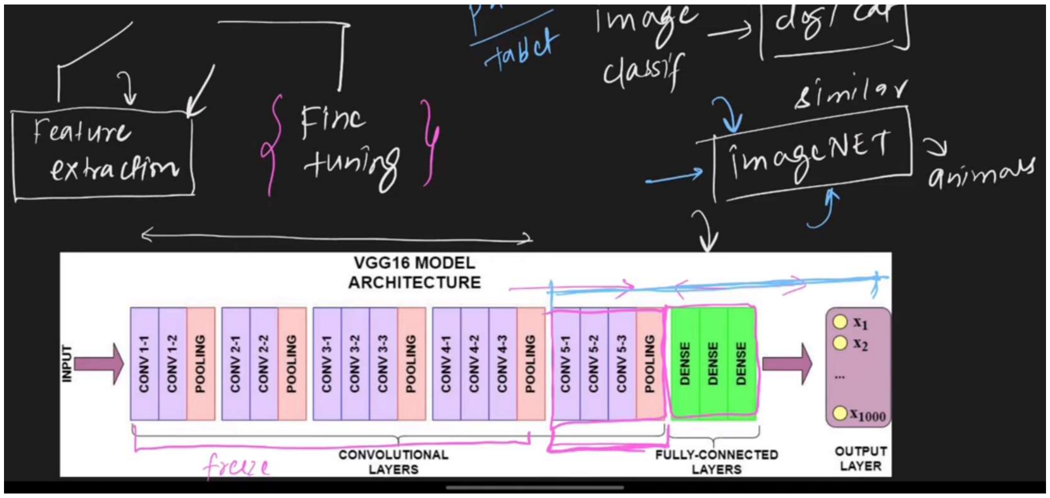
If we freeze all the earlier layers of a pre-trained model and only train the last convolutional layer, we allow the model to adapt to the new dataset by focusing on refining the most task-specific features, while still leveraging the general feature extraction capabilities already learned from the original dataset.

It means take last conv-layer in training with dense layer, while the rest of convolutional layers are freeze – to learn new dataset.

Example:

Imagine you have a model pre-trained on animal images, and you want to use it to classify different types of flowers. By freezing the early layers, which are good at detecting basic shapes and textures, and only training the last convolutional layer, you allow the model to learn the specific details of flowers while still using its general visual understanding developed from the animal images. This way, the model quickly adapts to the new task with minimal training.

Ways of doing Transfer Learning



Key Point:

- Feature extraction – In this conv layer do not participate in training only dense layer participate.
- Fine tuning – In this last conv layer and dense layer participate in training while rest of conv layers are freeze (means it does not participate).

Ways of doing Transfer Learning

Transfer learning can be approached in several ways, depending on the similarity between the original and new tasks, the size of the new dataset, and the computational resources available. Here are the main methods:

Feature Extraction:

Feature extraction in transfer learning involves using a pre-trained model to extract useful features from new data. The earlier layers of the model are frozen (their weights are not updated) because they capture general patterns like edges and textures, which are applicable across various tasks. Only the final layers, usually fully connected ones, are trained to classify the new data based on these extracted features.

Approach: Use the pre-trained model as a fixed feature extractor.

Process:

- Freeze all layers of the pre-trained model.
- Replace the final classification layer(s) with new ones that match the number of classes in the new task.
- Train only the new layer(s) while keeping the rest of the model unchanged.

Use Case: Effective when the new dataset is small and similar to the original dataset.

Fine-Tuning:

Fine-tuning goes a step further by not only training the final layers but also selectively unfreezing and retraining some of the pre-trained layers (often the later convolutional layers).

This allows the model to adjust the more specific features it learned from the original dataset to better fit the new dataset, making the model more specialized for the new task while still leveraging the knowledge gained from the original training.

Approach: Fine-tune some or all of the pre-trained model's layers to adapt them to the new task.

Process:

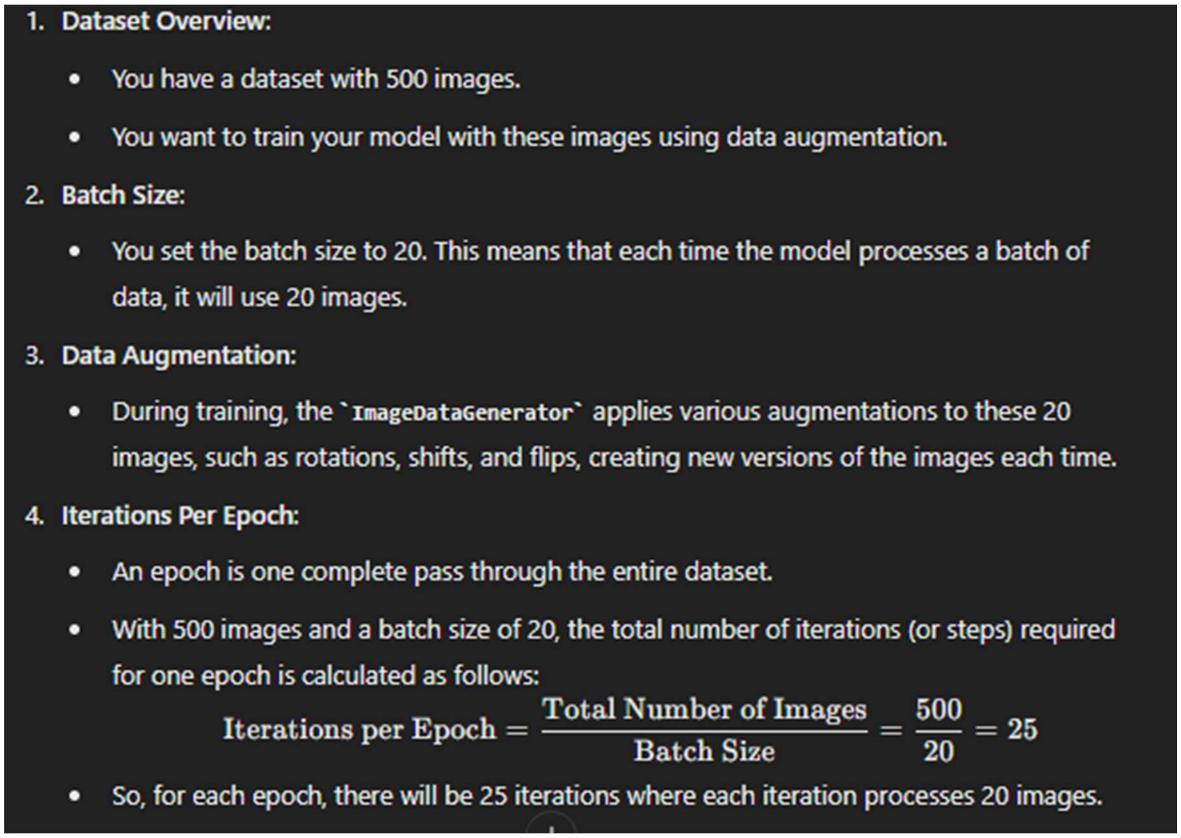
- Start with a pre-trained model.
- Freeze the earlier layers to retain the general features.
- Unfreeze some of the later layers (and optionally add new layers) and retrain them on the new dataset.

Use Case: Suitable when the new dataset is moderately sized or somewhat different from the original dataset.

How many images does data augmentation generate?

Then the "ImageDataGenerator" will produce 10 images in each iteration of the training. An iteration is defined as steps per epoch i.e. the total number of samples / batch_size. In above case, in each epoch of training there will be 100 iterations

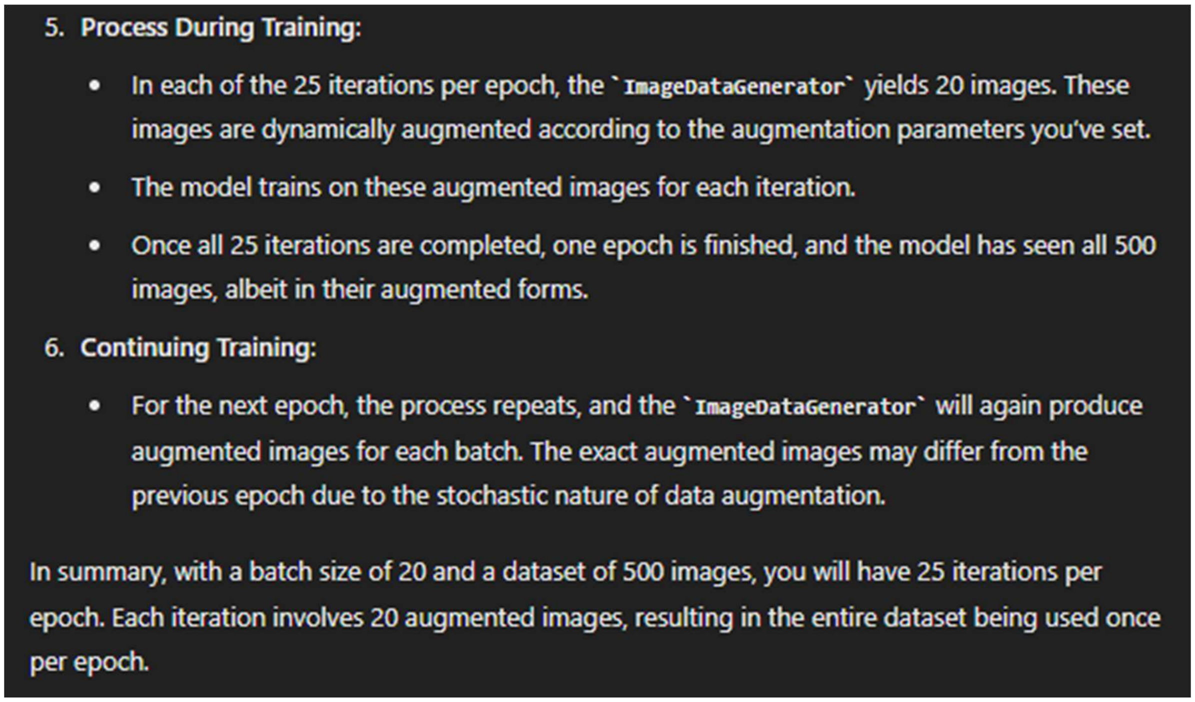
Example:



1. **Dataset Overview:**
   - You have a dataset with 500 images.
   - You want to train your model with these images using data augmentation.
2. **Batch Size:**
   - You set the batch size to 20. This means that each time the model processes a batch of data, it will use 20 images.
3. **Data Augmentation:**
   - During training, the `` `ImageDataGenerator` `` applies various augmentations to these 20 images, such as rotations, shifts, and flips, creating new versions of the images each time.
4. **Iterations Per Epoch:**
   - An epoch is one complete pass through the entire dataset.
   - With 500 images and a batch size of 20, the total number of iterations (or steps) required for one epoch is calculated as follows:

$$\text{Iterations per Epoch} = \frac{\text{Total Number of Images}}{\text{Batch Size}} = \frac{500}{20} = 25$$

   - So, for each epoch, there will be 25 iterations where each iteration processes 20 images.


5. **Process During Training:**
   - In each of the 25 iterations per epoch, the `ImageDataGenerator` yields 20 images. These images are dynamically augmented according to the augmentation parameters you've set.
   - The model trains on these augmented images for each iteration.
   - Once all 25 iterations are completed, one epoch is finished, and the model has seen all 500 images, albeit in their augmented forms.
6. **Continuing Training:**
   - For the next epoch, the process repeats, and the `ImageDataGenerator` will again produce augmented images for each batch. The exact augmented images may differ from the previous epoch due to the stochastic nature of data augmentation.

In summary, with a batch size of 20 and a dataset of 500 images, you will have 25 iterations per epoch. Each iteration involves 20 augmented images, resulting in the entire dataset being used once per epoch.

Note: Do not apply to much augmentation techniques it can also decrease model accuracy. The technique which we need to apply sometimes depends on model.

Extra:

An LSTM (Long Short-Term Memory) is a type of recurrent neural network (RNN) designed to learn and remember long-term dependencies in sequential data by using gating mechanisms to control the flow of information.