Image Coloring using Autoencoders

Transfer Learning

Agenda

- Image Colorization
- Autoencoders
- Dataset Overview
- Transfer Learning
- VGG-16

Image Colorization

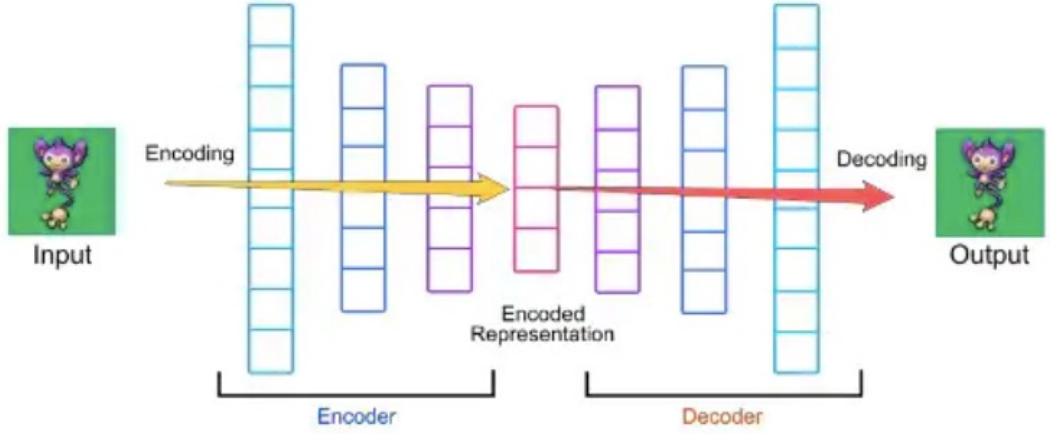
- **Image colorization** is the process of taking an input grayscale (black and white) image and then producing an output colorized image that represents the semantic colors and tones of the input.
- Image colorization assigns a color to each pixel of a target grayscale image
- Traditional colorization techniques requires significant user interaction.
- In this process, a fully automated data-driven approach “autoencoders” is used for colorization
- This method requires neither pre-processing nor post-processing.

Image Colorization Application

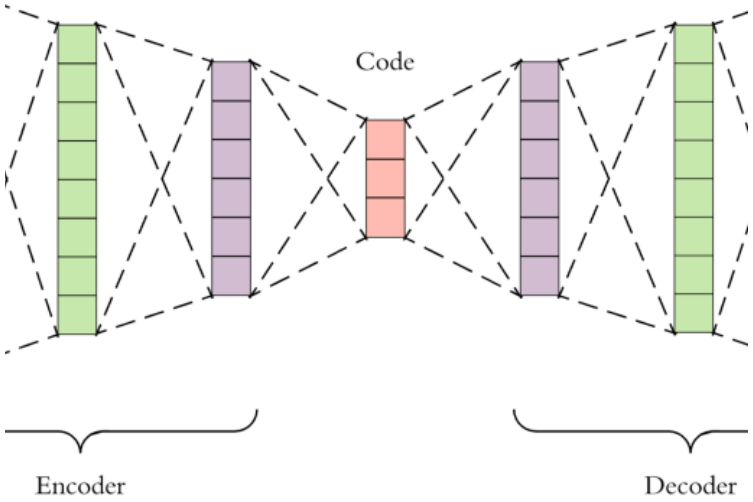
- Medical Microscope
- Medical Imagery
- Denoising and recreating old Images
- Night Vision Cameras
- Survillance

Autoencoders

An ***autoencoder*** neural network is an unsupervised Machine learning algorithm that applies backpropagation, setting the target values to be equal to the inputs.



Autoencoders



Autoencoders are a specific type of feedforward neural networks where the input is the same as the output.

An autoencoder neural network is an Unsupervised Machine learning algorithm that applies backpropagation, setting the target values to be equal to the inputs.

They compress the input into a lower-dimensional code(Encoding) and then reconstruct the output from this representation (Decoding).

To build an autoencoder we need 3 things: an encoding method, decoding method, and a loss function to compare the output with the target

Autoencoders

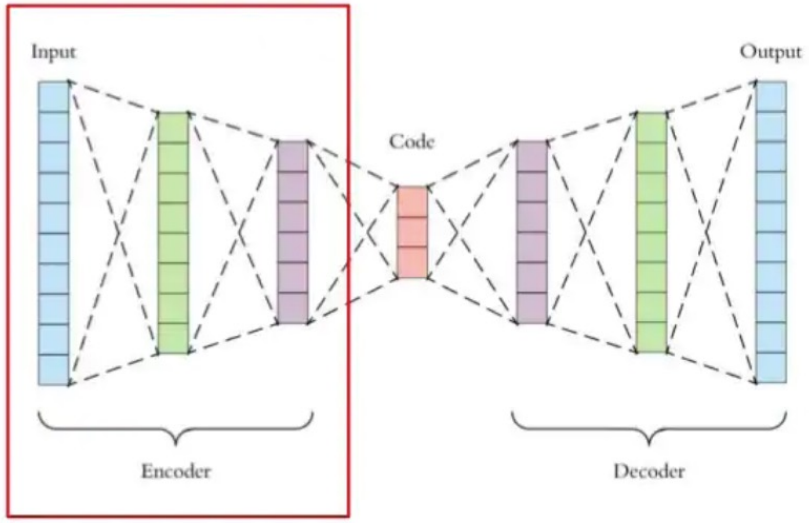
Encoder-Decoder as a very general framework/architecture design. In this design, We have some function that maps an input space to a different/latent space (the “encoder”). The decoder is simply the complementary function that creates a map from the (encoder’s) latent space to another target space (what is it we want to decode from the latent space).

However, **an autoencoder is a special case of an encoder-decoder architecture .**

In this, the target is the same as the input : X = model.predict(X)

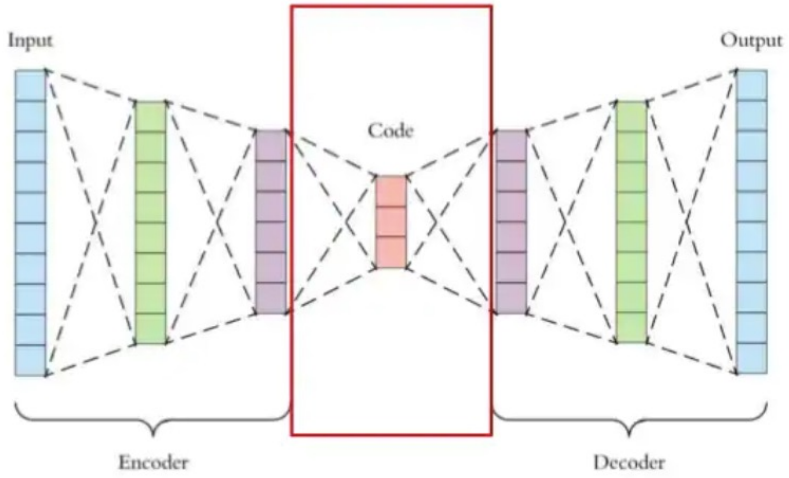
It is not predicting the future but simply reconstructing the present given a state/memory . Autoencoder is doing auto-association, which means that the model learns how to “recall” an input .

Autoencodes Arch’



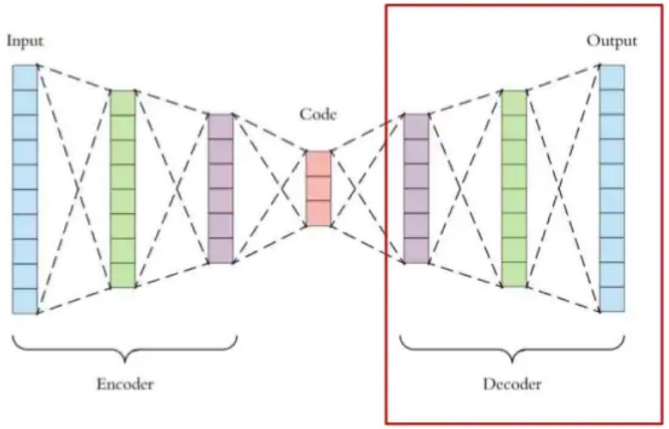
01 **Encoder**

This is the part of the network that compresses the input into a latent space representation.



02 **Code**

This is the part of the network represents the compressed input that is fed to the decoder



03 **Decoder**

This part aims to reconstruct the input from the latent space representation

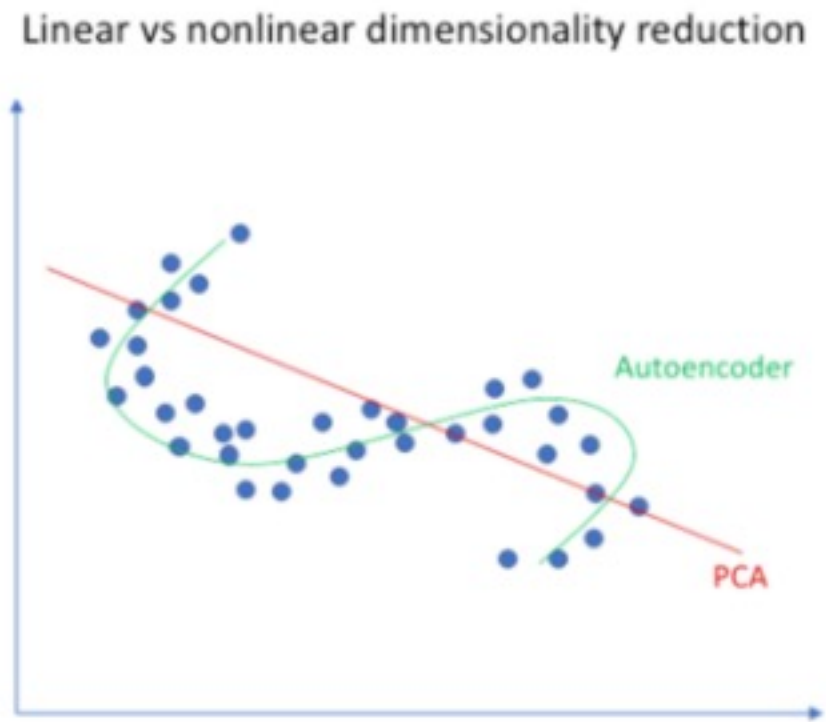
Features


01
Unsupervised
Autoencoders are considered an unsupervised learning technique since they don't need explicit labels to train on
02
Data-specific
Autoencoders are only able to meaningfully compress data similar to what they have been trained on
03
Lossy
The output of the autoencoder will not be exactly the same as the input, it will be a close but degraded representation

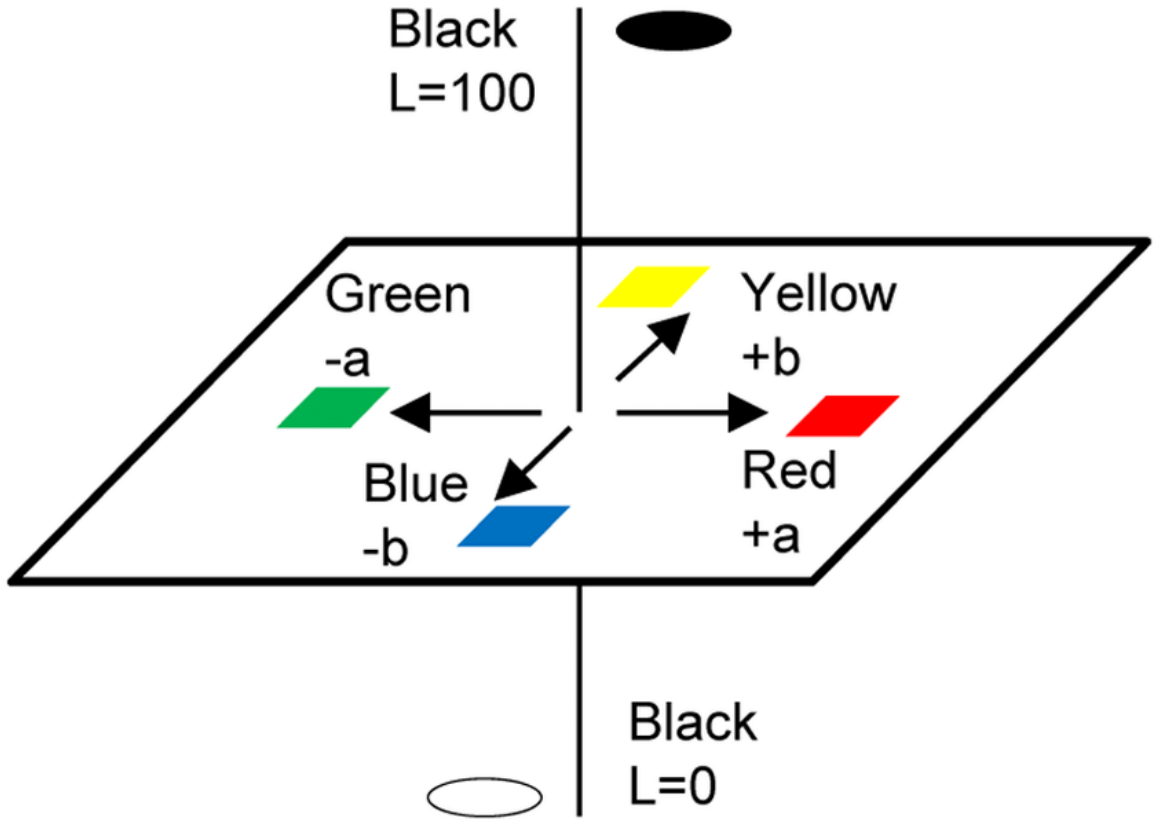
Autoencoders Tutorial: Its Emergence

Autoencoders are preferred over PCA because:

- An autoencoder can learn **non-linear transformations** with a **non-linear activation function** and multiple layers.
- It doesn't have to learn dense layers. It can use **convolutional layers** to learn which is better for video, image and series data.
- It is more efficient to learn several layers with an autoencoder rather than learn one huge transformation with PCA.
- An autoencoder provides a representation of each layer as the output.
- It can make use of **pre-trained layers** from another model to apply transfer learning to enhance the encoder/decoder.



The Data Set Overview



In this Model, Colored Input Images(Landscape Dataset) used.

These Images are decomposed using Lab model and used as an input feature (“L”) and classification labels (“a” and “b”).

For simplicity let’s split in two: “L” and “a+b” as shown in the block diagram.

Having the trained model , we can use it to colorize Black and White Test Images.

Transfer Learning

Humans have an inherent ability to transfer knowledge across tasks. What we acquire as knowledge while learning about one task, we utilize in the same way to solve related tasks. The more related the tasks, the easier it is for us to transfer, or cross-utilize our knowledge. Some simple examples would be,

- Know how to ride a motorbike Learn how to ride a car
- Know how to play classic piano Learn how to play jazz piano
- Know math and statistics Learn machine learning

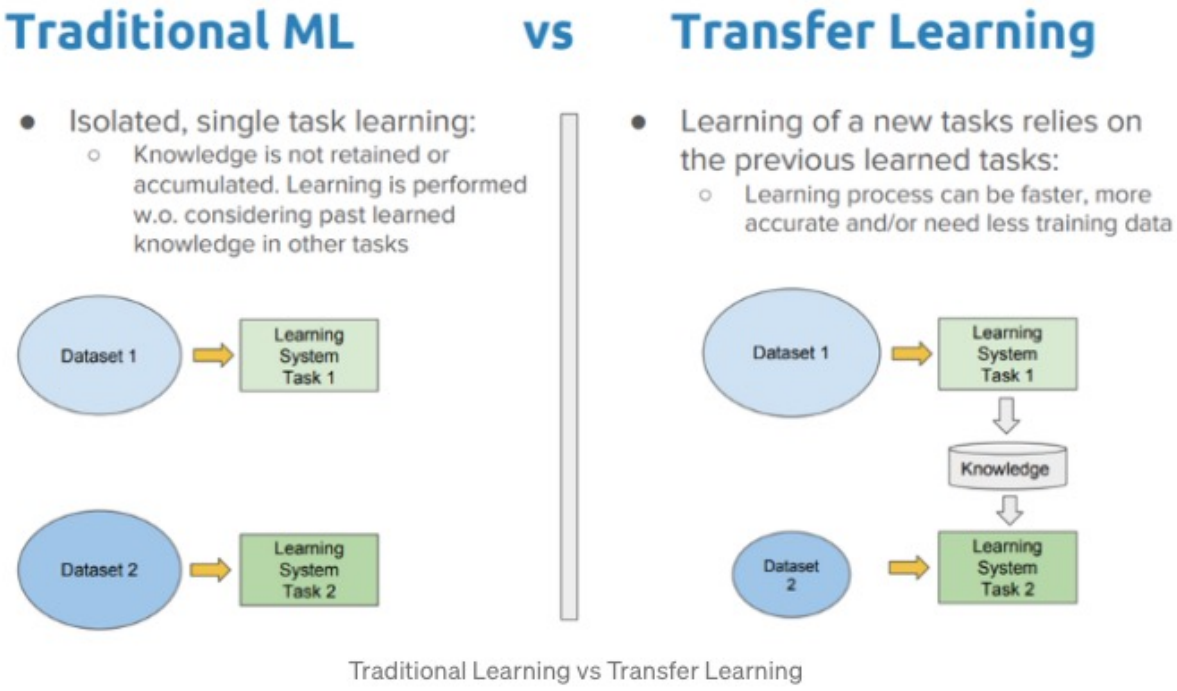
Transfer learning is a machine learning method where a model developed for a task is reused as the starting point for a model on a second task.

It is a popular approach in deep learning where pre-trained models are used as the starting point on computer vision and natural language processing tasks given the vast compute and time resources required to develop neural network models on these problems and from the huge jumps in skill that they provide on related problems.

Transfer Learning (Cont’)

Transfer learning, as we have seen so far, is having the ability to utilize existing knowledge from the source learner in the target task. During the process of transfer learning, the following three important questions must be answered:

- **What to transfer:** This is the first and the most important step in the whole process. We try to seek answers about which part of the knowledge can be transferred from the source to the target in order to improve the performance of the target task. When trying to answer this question, we try to identify which portion of knowledge is source-specific and what is common between the source and the target.
- **When to transfer:** There can be scenarios where transferring knowledge for the sake of it may make matters worse than improving anything (also known as negative transfer). We should aim at utilizing transfer learning to improve target task performance/results and not degrade them. We need to be careful about when to transfer and when not to.
- **How to transfer:** Once the *what* and *when* have been answered, we can proceed towards identifying ways of actually transferring the knowledge across domains/tasks. This involves changes to existing algorithms and different techniques, which we will cover in later sections of this article. Also, specific case studies are lined up in the end for a better understanding of how to transfer.



Traditional Learning vs Transfer Learning

Transfer Learning (Cont')

Transfer learning in DL

Myth: you can't do deep learning unless you have a million labeled examples for your problem.

Reality

- You can learn useful representations from **unlabeled data**
- You can train on a nearby **surrogate objective** for which it is easy to generate labels
- You can **transfer** learned representations from a related task

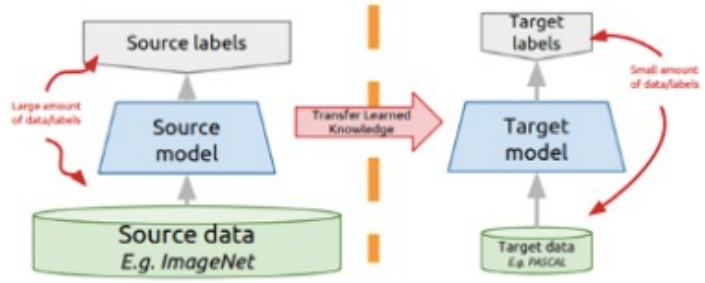
Transfer learning: idea

Instead of training a deep network from scratch for your task:

- Take a network trained on a different domain for a different **source task**
- Adapt it for your domain and your **target task**

Variations:

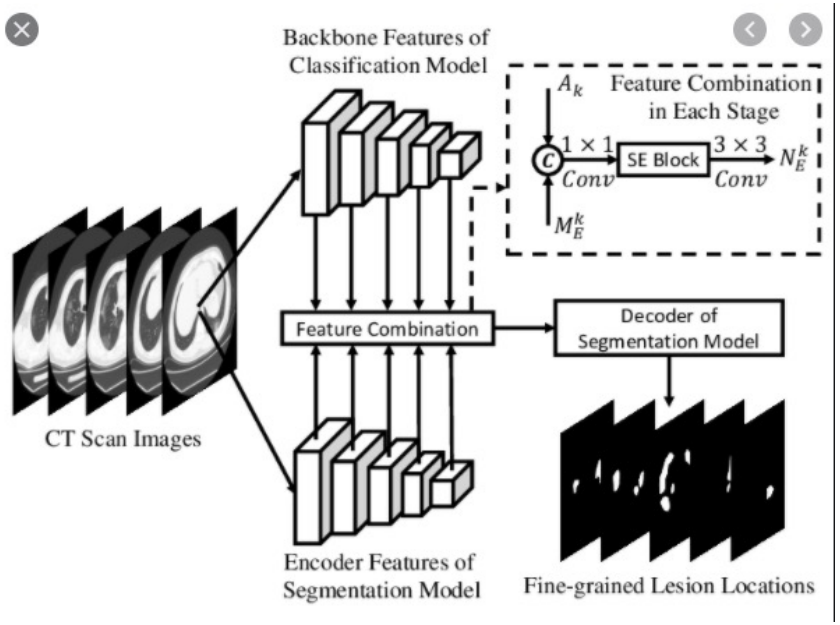
- Same domain, different task
- Different domain, same task

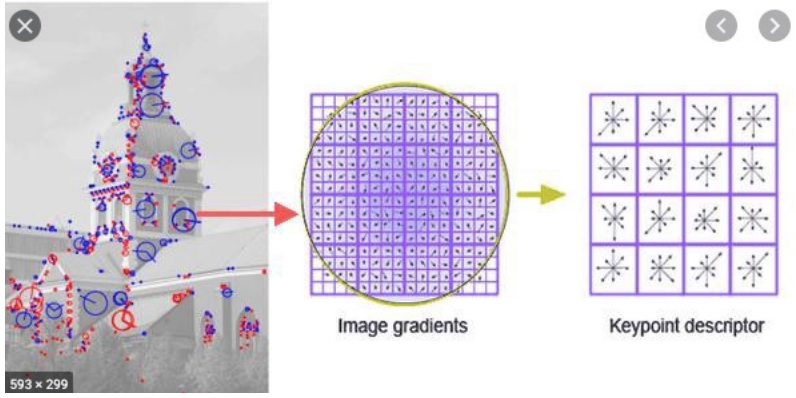


Ideas for deep transfer learning

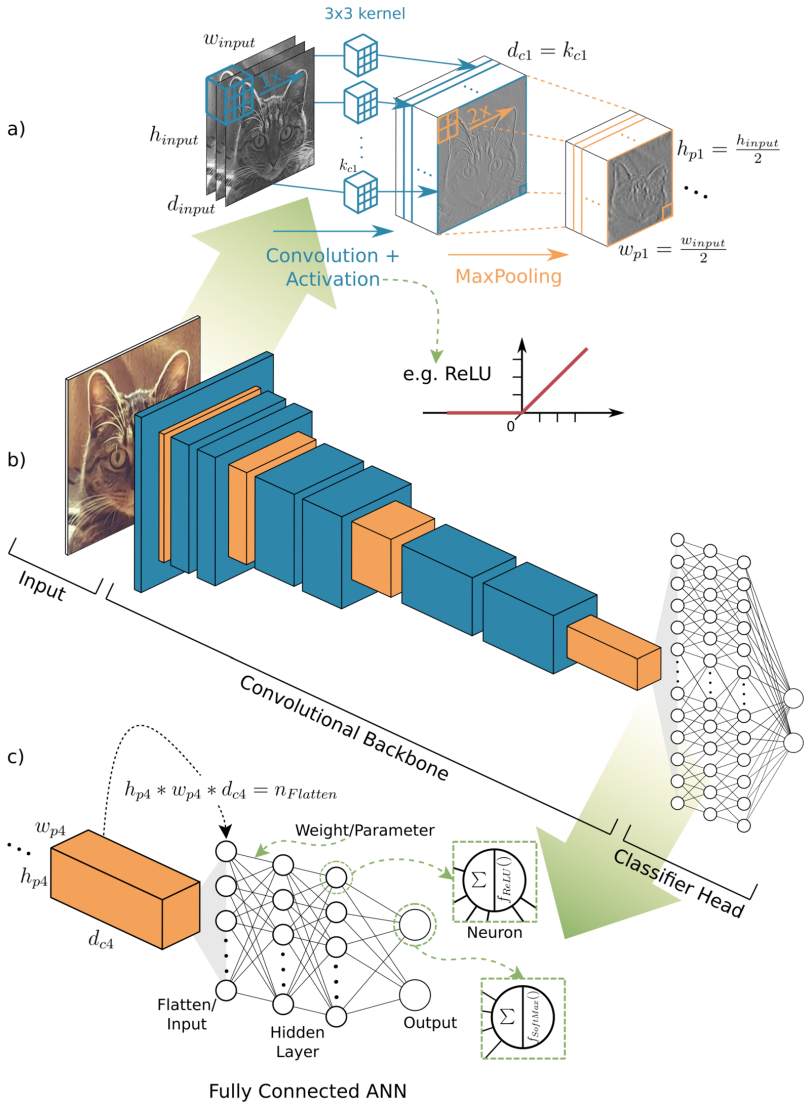
Backbone

- Backbone is a term used in models/papers to refer to the feature extractor network.
- This is a standard convolutional neural network (typically, VGG16 or VGG19) that serves as a feature extractor. The early layers detect low level features (edges and corners), and later layers successively detect higher level features (car, person, sky).
- Backbone gives you a feature map representation of input
- Multiple pre trained model used as a backbone for a variety of tasks
- **VGG16**, is a classic neural network used as a **backbone** for many computer vision tasks

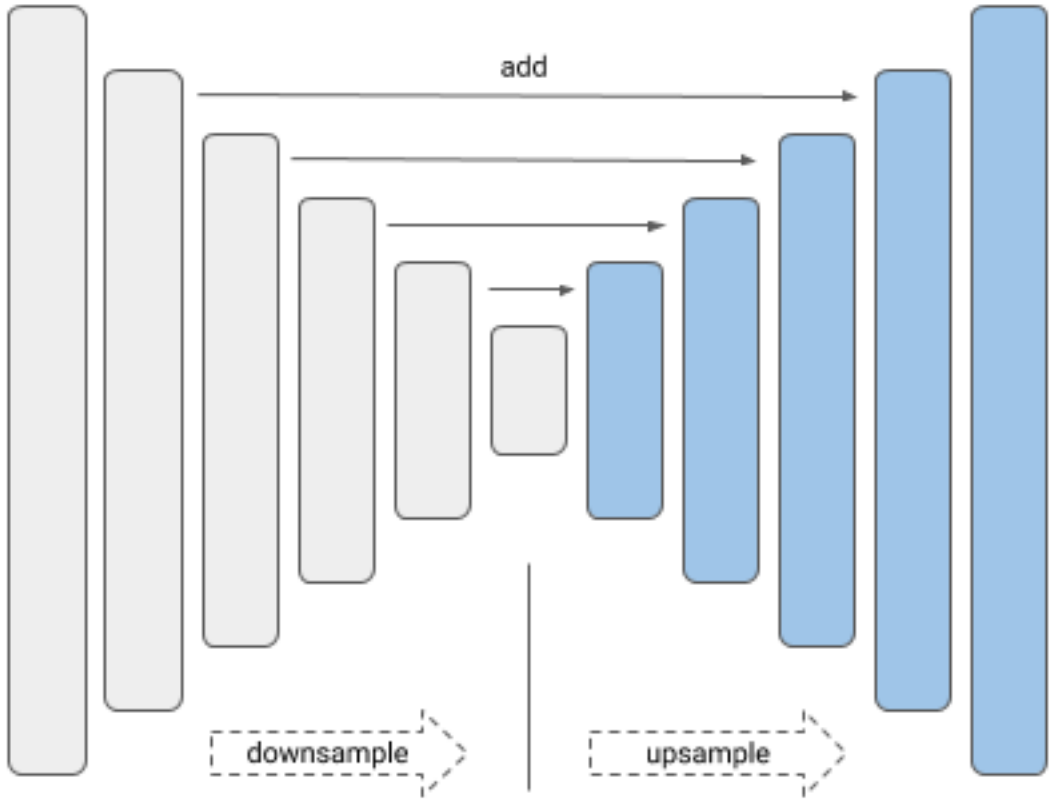




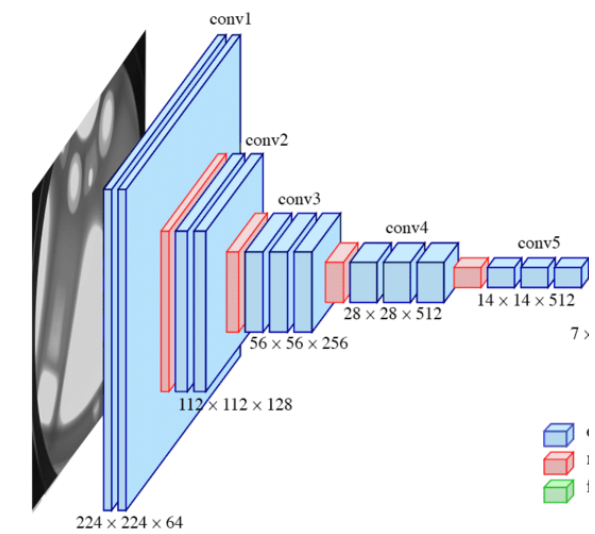
Autoencoders with Backbone



Autoencoders with VGG16



VGG-16 Model Backbone



VGG16 is a convolution neural net (CNN) architecture Visual Geometry Group from Oxford, who developed it.

It follows this arrangement of convolution and max pool layers consistently throughout the whole architecture. In the end it has 2 FC(fully connected layers) followed by a softmax for output.

The network has 41 layers. There are 16 layers with learnable weights: 13 convolutional layers, and 3 fully connected layers.

The 16 in VGG16 refers to it has 16 layers that have weights. This network is a pretty large network, and it has about 138 million (approx) parameters.