Welcome to Deep Learning Online Bootcamp

Day 14 - Convolutional Neural Networks

dφ

Democratizing Data Science Learning

Learning Objectives

Spatial/ Translation Invariance

Convolutional Neural Network

Deep Learning on Limited Data - Transfer learning, Data Augmentation

Disadvantages of MLP

Comparison of CNN and ANN/MLP architecture

Batch Normalization

Until now you've worked with Multi Layer Perceptrons(MLP).

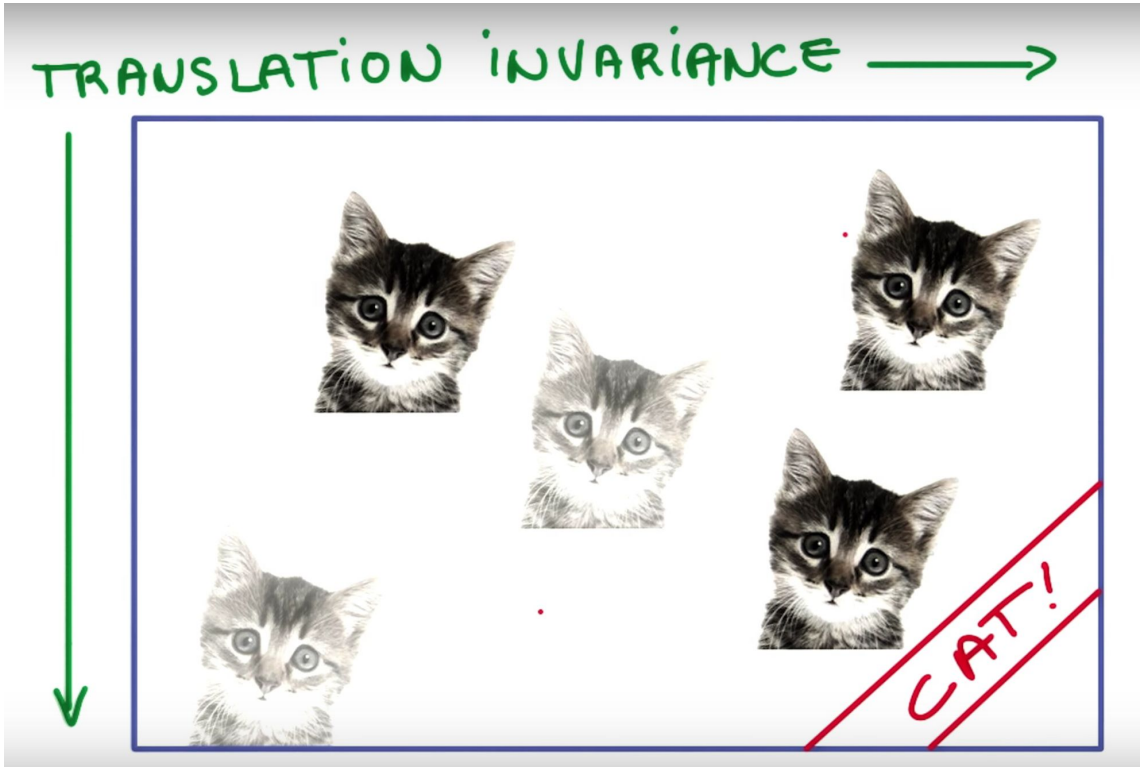
MLPs are great for MNIST - a simpler more straight forward dataset but lag behind CNNs when it comes to real world application in computer vision, specifically image classification.

The concepts that make CNN's so great are not complex but are very intuitive, logical and easy to understand.

Spatial/Translation Invariance

Spatial or Translation Invariance means that if an object occurs in any image it will be detected irrespective of its position in the image.

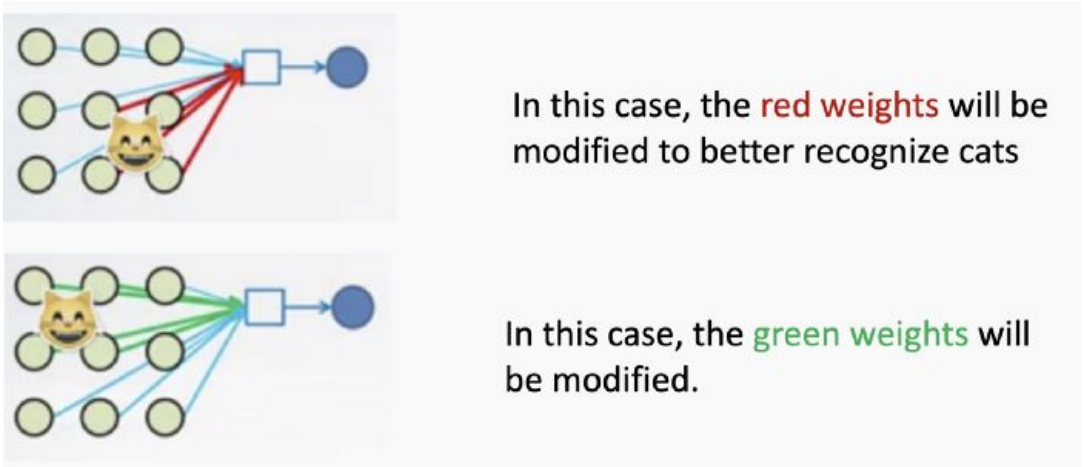
A cat is still a cat regardless of whether it appears in the top half or the bottom half of the image.



Disadvantages of MLP

- Need to **connect neuron in hidden layer to ALL the neurons in input layer**

- **No spatial information**: Since we reshape the image from 2D to 1D, it doesn't really understand the spatial structure of the image i.e it is **not spatially invariant**. For example, if a picture of a cat appears in the top left of the image in one picture and the bottom right of another picture, the MLP will try to correct itself and assume that a cat will always appear in this section of the image.



A cat detector using an MLP which changes as the position of the cat changes.

- And **many, many parameters** to be handled. MLPs use one perceptron for each input (e.g. pixel in an image, multiplied by 3 in RGB case). The amount of weights rapidly becomes unmanageable for large images. For a 224 x 224 pixel image with 3 color channels there are around 150,000 weights that must be trained!

Enters the Convolutional Neural Network

Convolutional Neural Networks are designed to be **spatially invariant**, that is - they are not sensitive to the position of object in the picture. The cat can be anywhere in the image and it'll still recognise it!

But what if the image doesn't just consist of a cat? What if there are more features? Will flipping the image confuse the CNN? No, a CNN will still work fine!

CNN's take advantage of the fact that **nearby pixels are more strongly related than distant ones**. So a pixels around sky will still be a sky and pixels near sunflower have more chances of being a sunflower.

Convolutional Neural Network/ ConvNet/ CNN

To teach an algorithm how to recognise objects in images, we use a specific type of Artificial Neural Network: a Convolutional Neural Network (CNN). Their name stems from one of the most important operations in the network: **convolution**.

The word convolution refers to the filtering process that happens in this type of network. In literal terms, it is a mathematical operation that makes filtering possible.

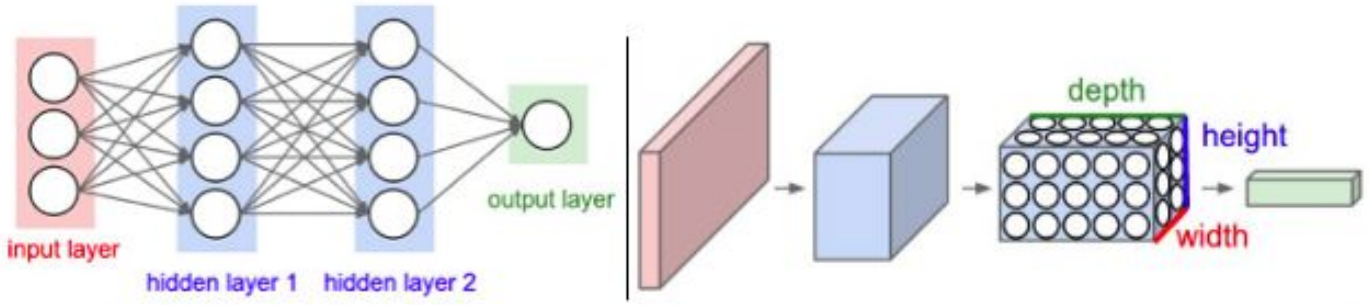
Comparison of Architectures - CNN and ANN/MLP

Regular Neural Networks:

- Transform an input by putting it through a series of hidden layers.
- Every layer is made up of a set of neurons, where each layer is fully connected to all neurons in the layer before.
- Finally, there is a last fully-connected layer — the output layer — that represent the predictions.

In CNNs :

- The layers are organised in 3 dimensions: width, height and depth.
- The neurons in one layer do not connect to all the neurons in the next layer but only to a small region of it.



Left: A regular 3-layer Neural Network. Right: A ConvNet arranges its neurons in three dimensions (width, height, depth), as visualized in one of the layers. Every layer of a ConvNet transforms the 3D input volume to a 3D output volume of neuron activations. In this example, the red input layer holds the image, so its width and height would be the dimensions of the image, and the depth would be 3 (Red, Green, Blue channels).

CNN Working

CNN Components

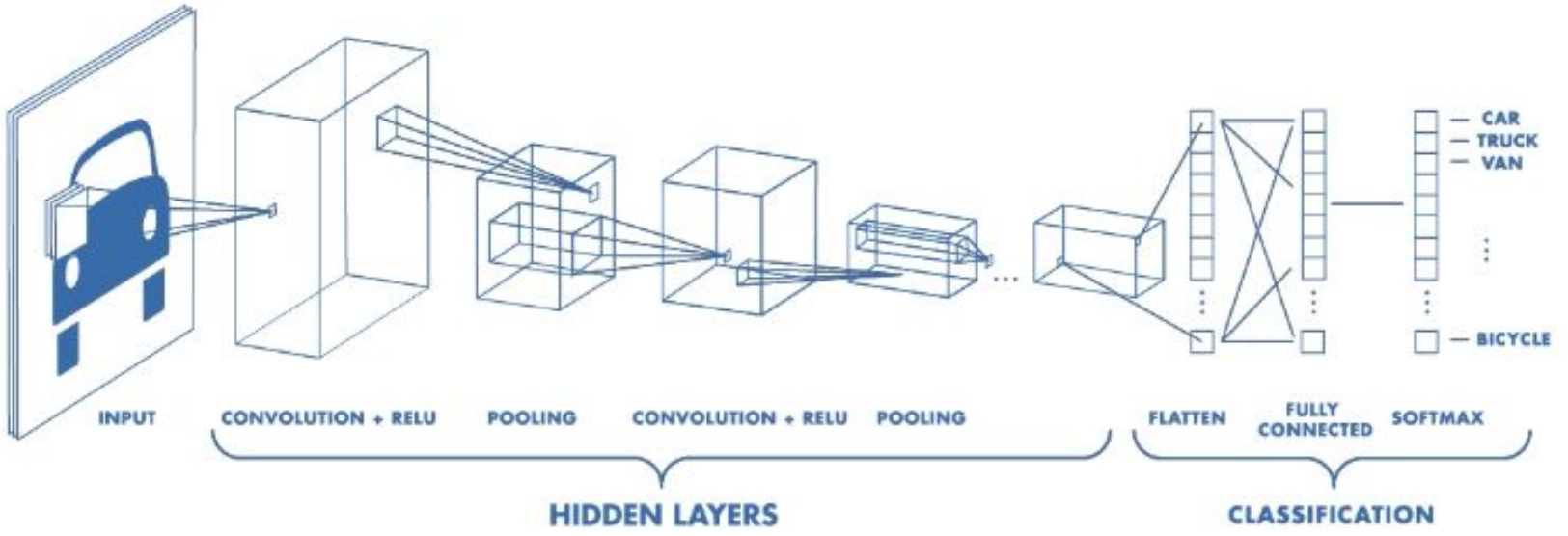
CNNs have two components:

1. **The Hidden layers/Feature extraction part**

In this part, the network will perform a series of operations during which the features are detected. If you had a picture of a zebra, this is the part where the network would recognise its stripes, two ears, and four legs.

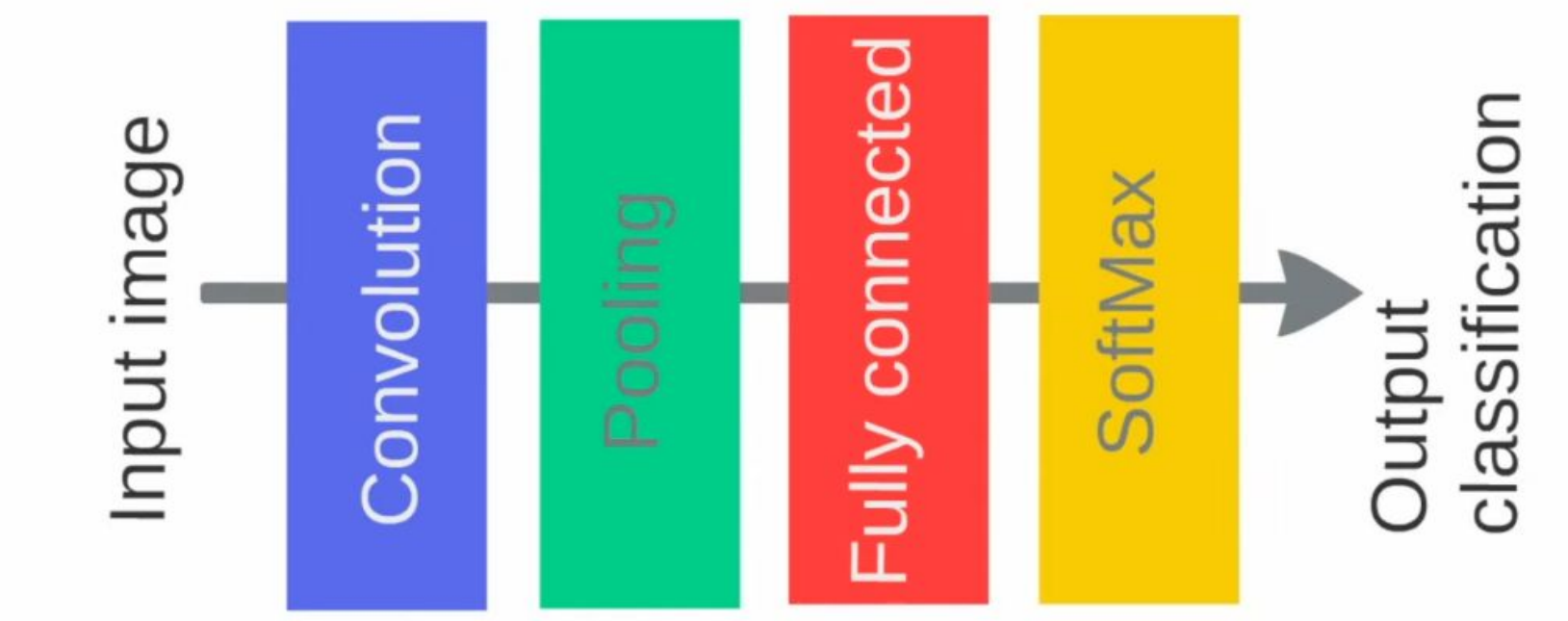
2. **The Classification part**

This part assigns a probability of the object(eg. The zebra) being in that image.



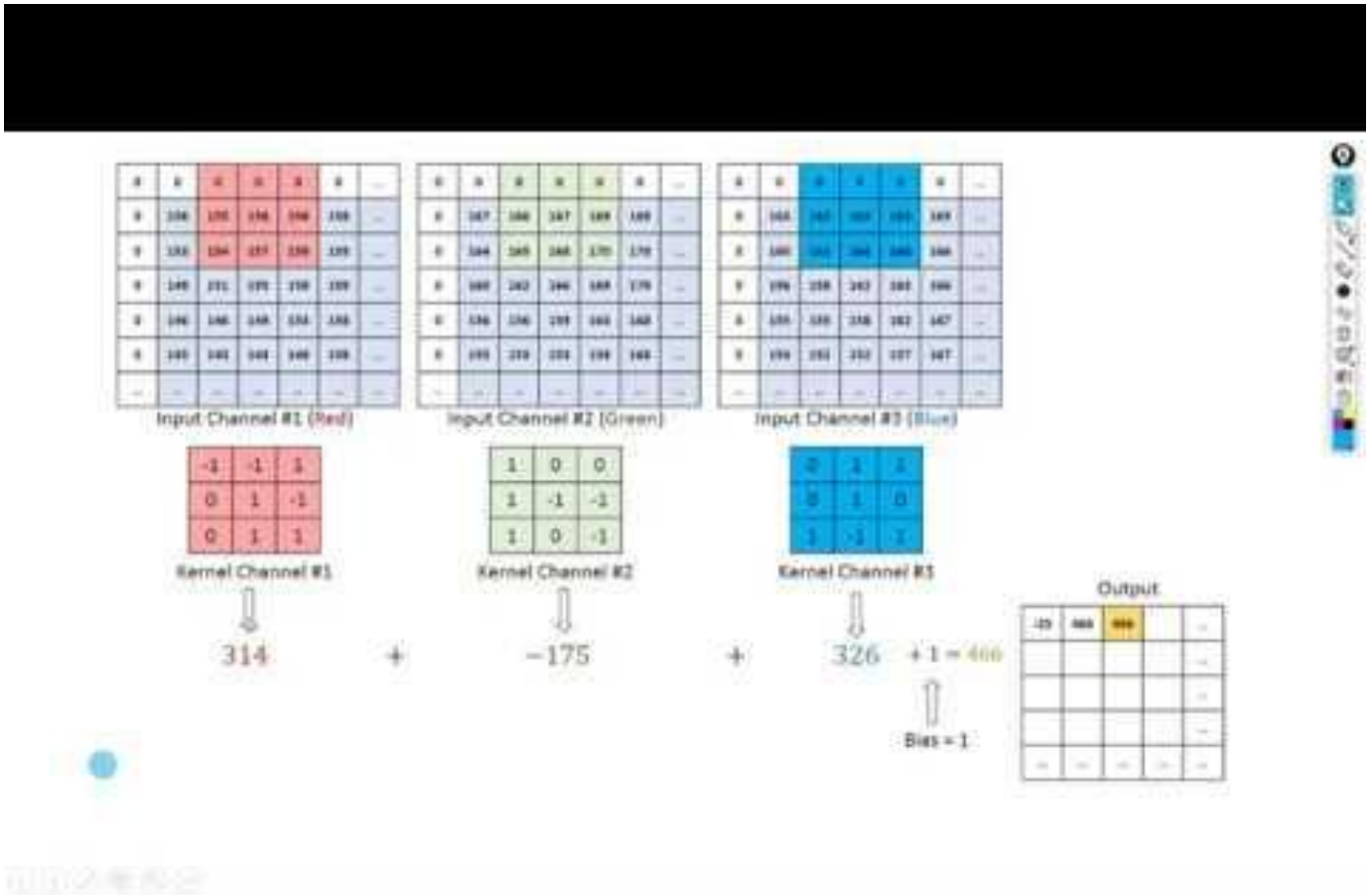
A basic CNN architecture

Below is the basic order of operations in a CNN. The Convolution and Pooling operations can occur multiple times in a CNN, the order will however remain the same. They'll always be succeeded by Fully Connected and Softmax/Sigmoid layer.

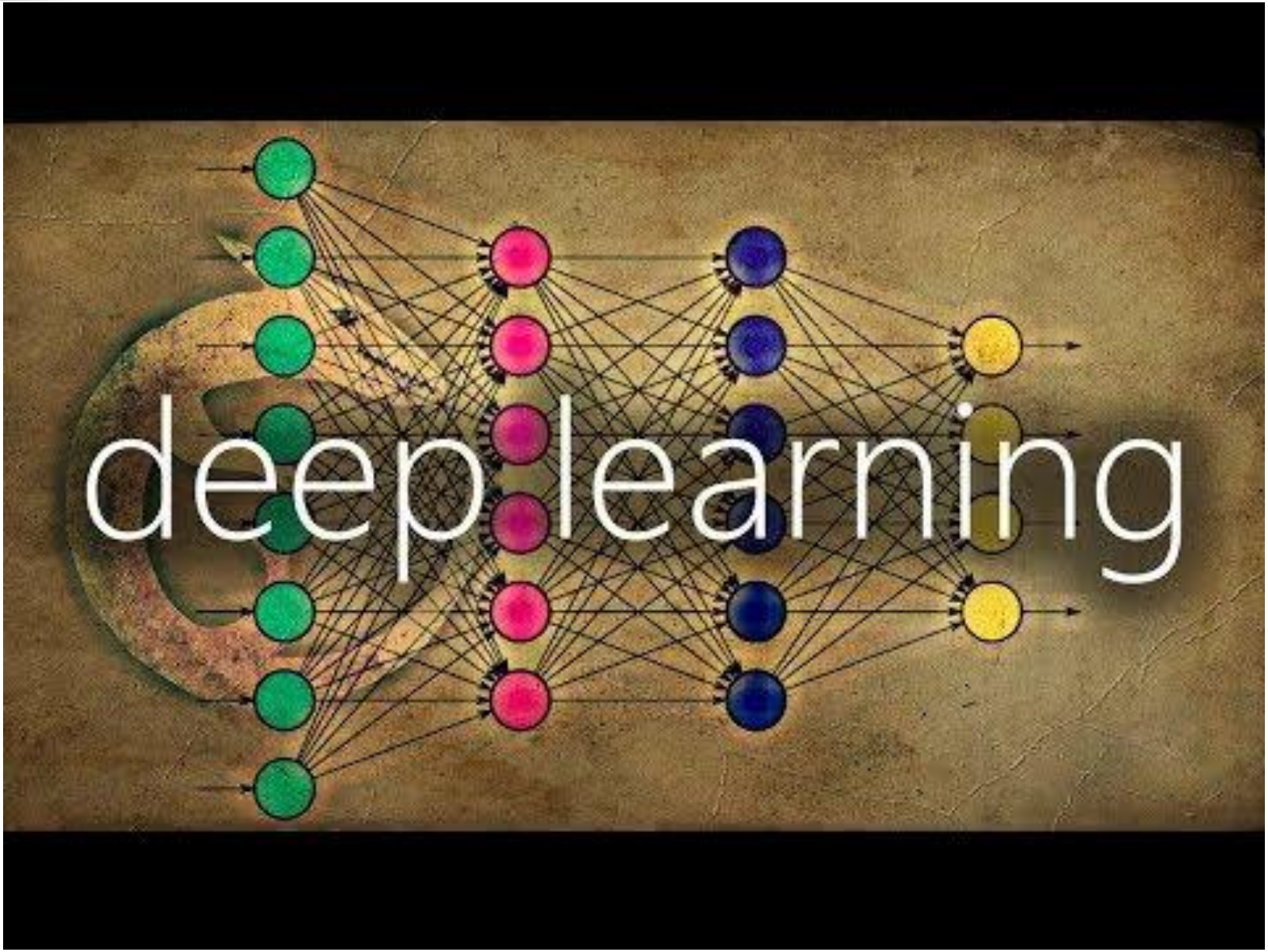


Basic Understanding of Filter, Stride & Convolution

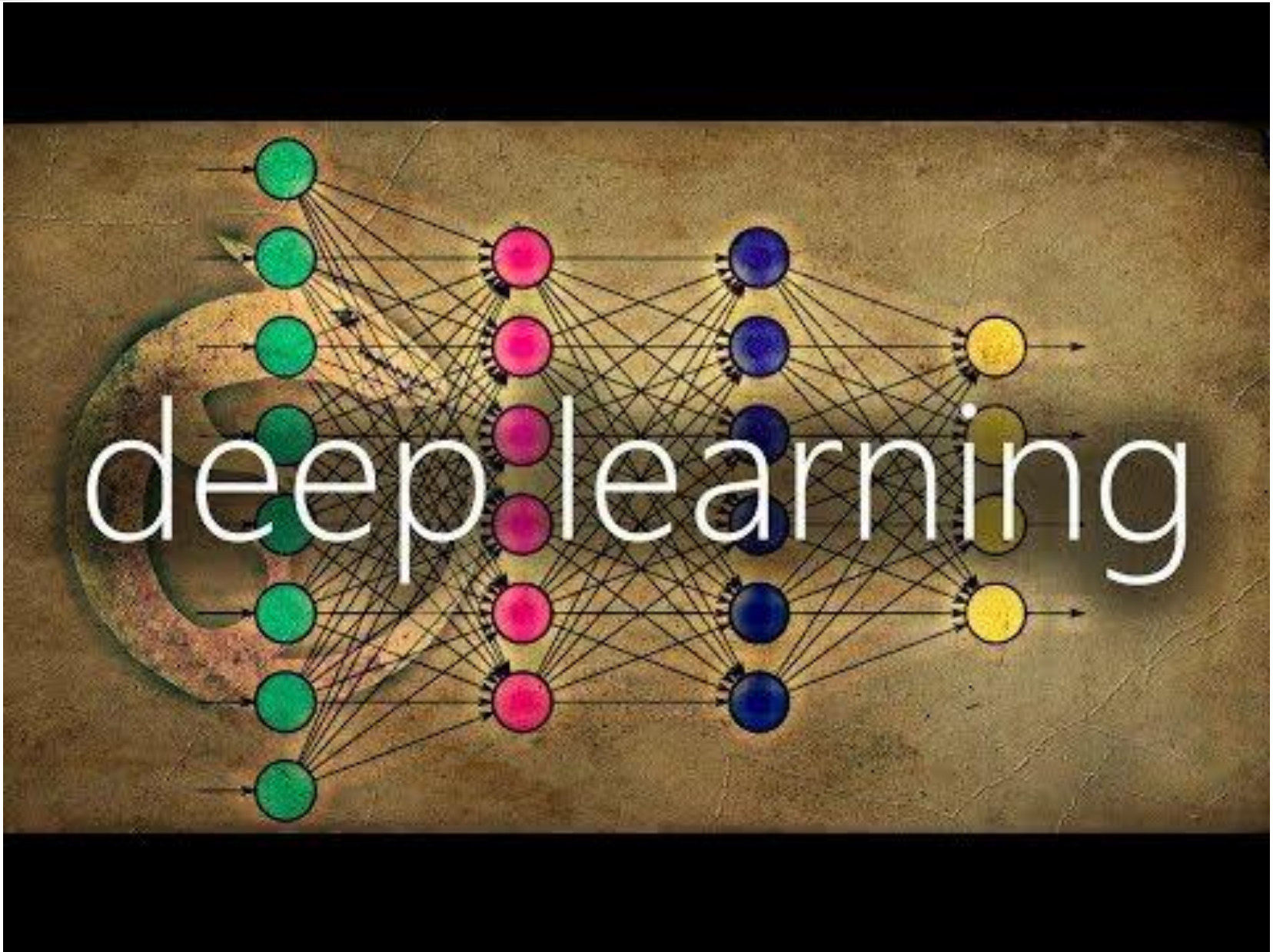
This is a brilliant explanation of how a kernel/filter makes convolution possible. For the people worried about how exactly does one get a particular size when applying a kernel on a matrix, this will solve your confusion.



What is 0 Padding and why is it even required?

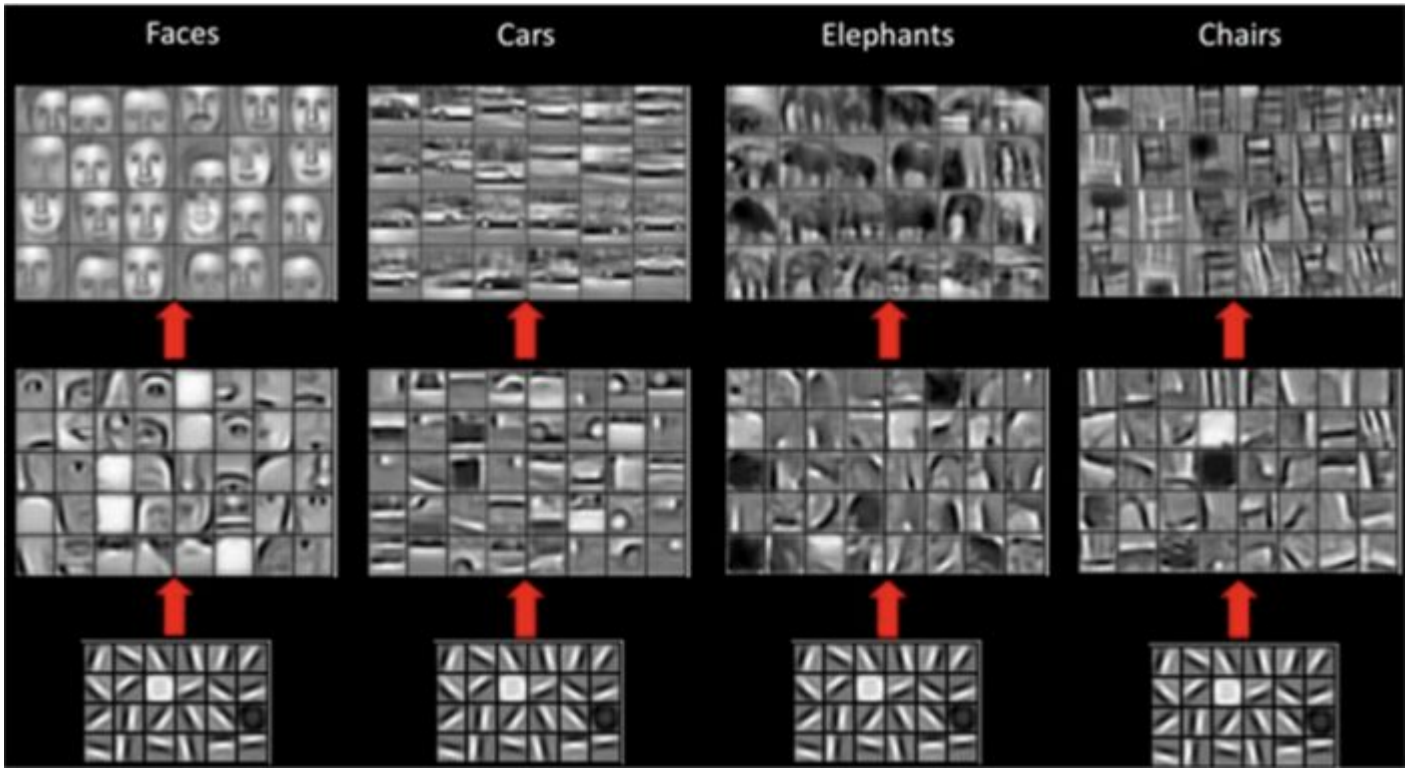


Max Pooling



What do CNN layers learn?

- Each CNN layer learns filters of increasing complexity.
- The first layers learn basic feature detection filters: edges, corners, etc
- The middle layers learn filters that detect parts of objects. For faces, they might learn to respond to eyes, noses, etc
- The last layers have higher representations: they learn to recognize full objects, in different shapes and positions.



What do CNN layers learn?

The earlier features of a ConvNet contain more generic features (e.g. edge detectors or color blob detectors), but later layers of the ConvNet becomes progressively more specific to the details of the classes contained in the original dataset.

How to use Deep Learning when you have Limited Data

The problem of Limited Data

"The analogy to deep learning is that the rocket engine is the deep learning models and the fuel is the huge amounts of data we can feed to these algorithms." — Andrew Ng

There has been a recent surge in popularity of Deep Learning, achieving state of the art performance in various tasks like Language Translation, playing Strategy Games and Self Driving Cars requiring millions of data points.

One common barrier for using deep learning to solve problems is the amount of data needed to train a model. The requirement of large data arises because of the large number of parameters in the model that machines have to learn.

So what do you do when while working on a problem, you're unable to find the required amount of data?

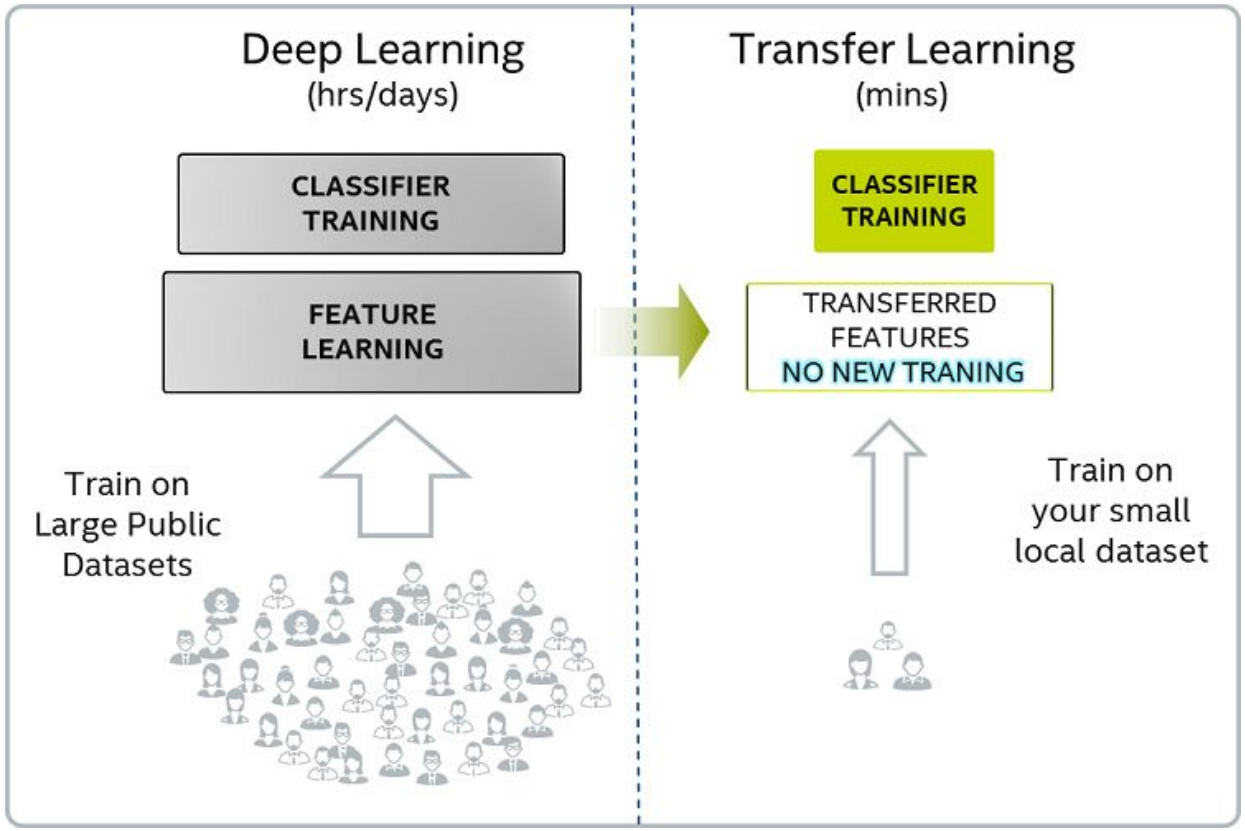
1. Transfer Learning to the Rescue!

“If Deep Learning is the holy grail and data is the gatekeeper, transfer learning is the key.”

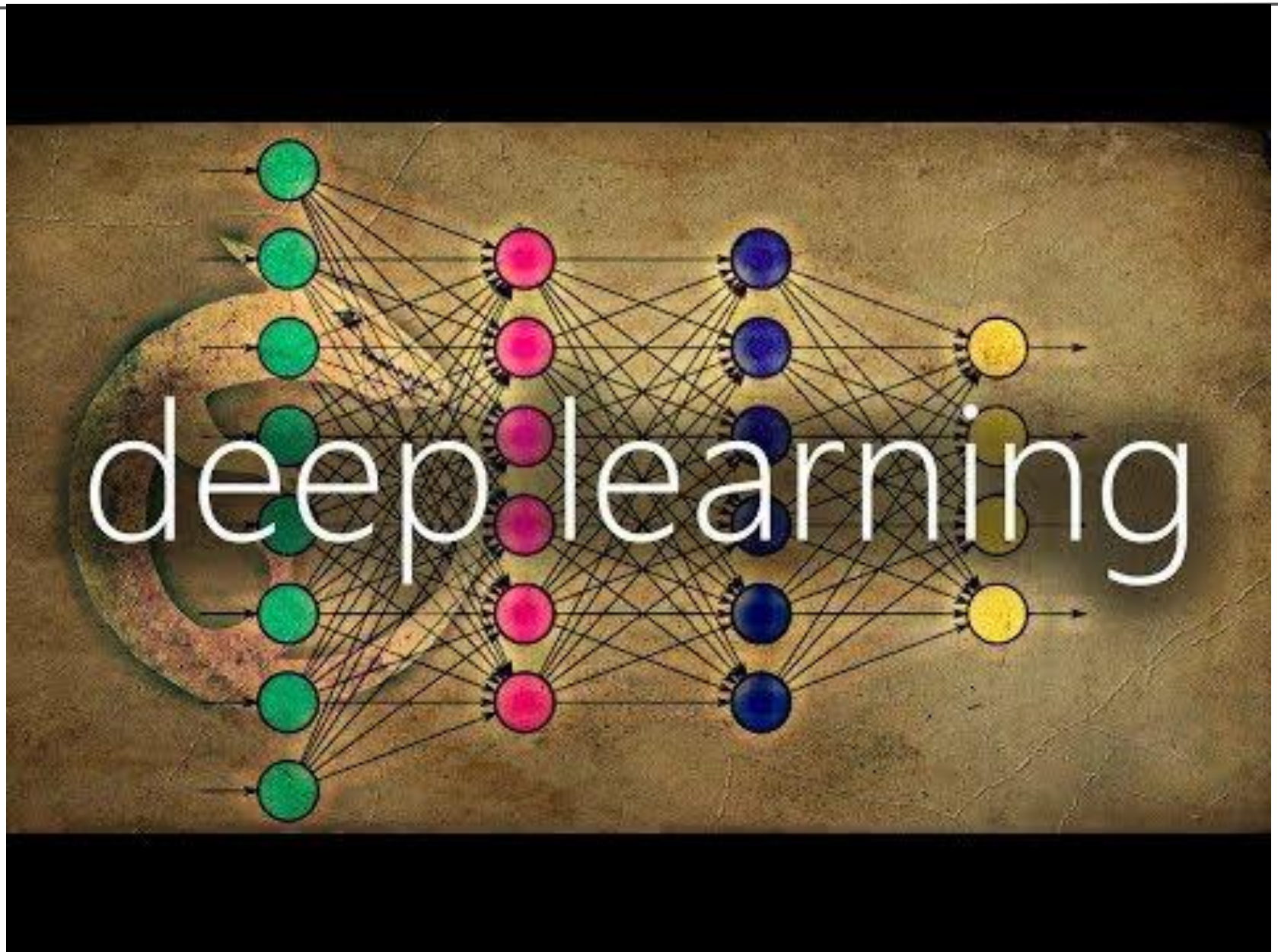
Transfer Learning

Transfer learning make use of the knowledge gained while solving one problem and applying it to a different but related problem.

For example, knowledge gained while learning to recognize cars can be used to some extent to recognize trucks.



Transfer Learning



Transfer Learning

Pre-training

When we train a network on a large dataset(for example: ImageNet) , we train all the parameters of the neural network and therefore the model is learned. It may take hours on your GPU to train on such a large dataset. This model is known as a **Pre-trained model**.

Fine Tuning

We can give the new dataset to fine tune the pre-trained CNN. Consider that the new dataset is almost similar to the original dataset used for pre-training. Since the new dataset is similar, the same weights can be used for extracting the features from the new dataset.

1. If the new dataset is very small, it's better to train only the final layers of the network to avoid overfitting, keeping all other layers fixed. So remove the final layers of the pre-trained network. Add new layers . **Retrain only the new layers.**

2. **If the new dataset is very much large, retrain the whole network** with initial weights from the pretrained model.

Transfer Learning

Another major advantage of using transfer learning is how well the model generalizes.

Larger models tend to overfit the data and don't work as well when you test it out on unseen data. Since transfer learning allows the model to see different types of data, it learns underlying rules of the world better.

Transfer Learning

You can have a look at this video if you're interested to learn how Transfer Learning is performed. It's a slightly advance topic so don't worry if you don't get it in the first go. Also, the instructor is using Keras but the same code will work with tf.Keras as well.

2. Data Augmentation

Data Augmentation

To get more data, we just need to make minor alterations to our existing dataset. Minor changes such as flips or translations or rotations. Our neural network would think these are distinct images anyway.

A convolutional neural network that can robustly classify objects even if its placed in different orientations is said to have the property called **invariance**. More specifically, a CNN can be invariant to **translation**, **viewpoint**, **size** or **illumination** (Or a combination of the above).

This essentially is the premise of **data augmentation**. In the real world scenario, we may have a dataset of images taken in a **limited set of conditions**. But, our **target application** may exist in a **variety of conditions**, such as different orientation, location, scale, brightness etc. We account for these situations by training our neural network with additional **synthetically modified data**.

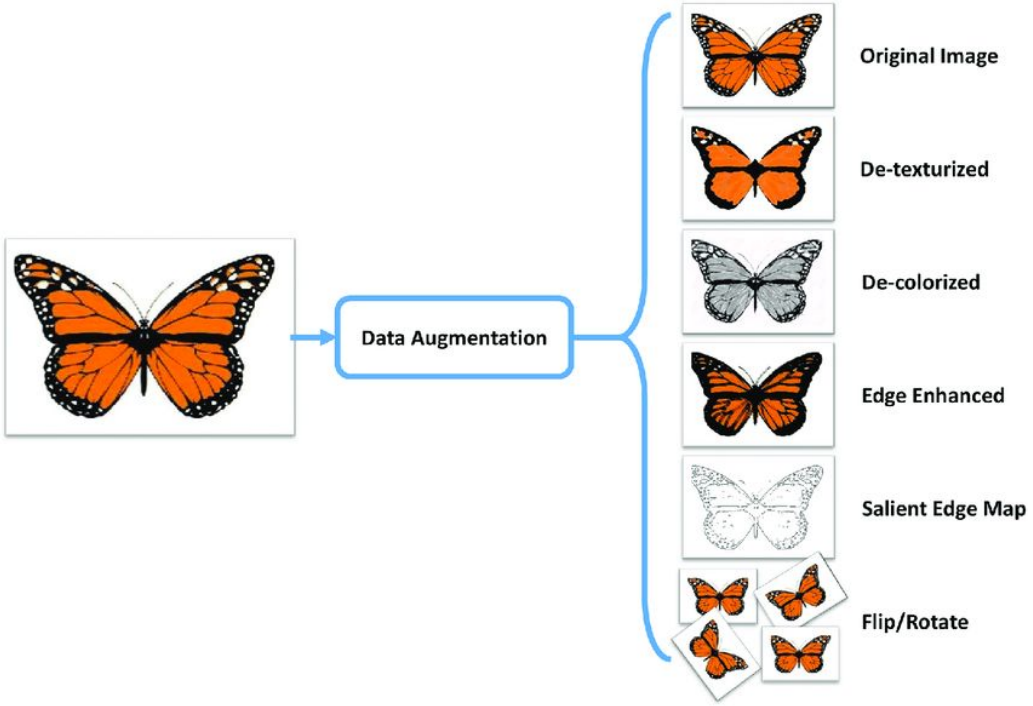
It is also one of the methods to **prevent overfitting**!

Data Augmentation

Your neural network is only as good as the data you feed it.

By performing augmentation, can prevent your neural network from learning irrelevant patterns, essentially boosting overall performance.

Your model for detecting a butterfly should be able to find a butterfly in the image even if it is flying sidewards, right?



Popular Augmentation Techniques on images

1. Flip - horizontally/vertically
2. Rotation
3. Scale
4. Crop
5. Translation
6. Varying Color
7. Adding noise
8. Adjusting brightness

Apart from the above, many more augmentation techniques exist.

Where to perform Data Augmentation?

Data Augmentation is usually performed on Train Set and often Validation Set as well. This is because we want to increase the size and variety of data our model learns from.

However, Data Augmentation can also be performed on Test Set. It is not to make the test data bigger/more accurate, but just to make the input data from the test set resemble that of the input data from the training set, so we can feed it into the same net (eg same dimensions).

For example, in image cropping, we'd need to crop the test images too, so they are of a similar size as the training images.

How to perform Data Augmentation?

A Tensorflow Keras function called `ImageDataGenerator` helps in performing Data Augmentation techniques.

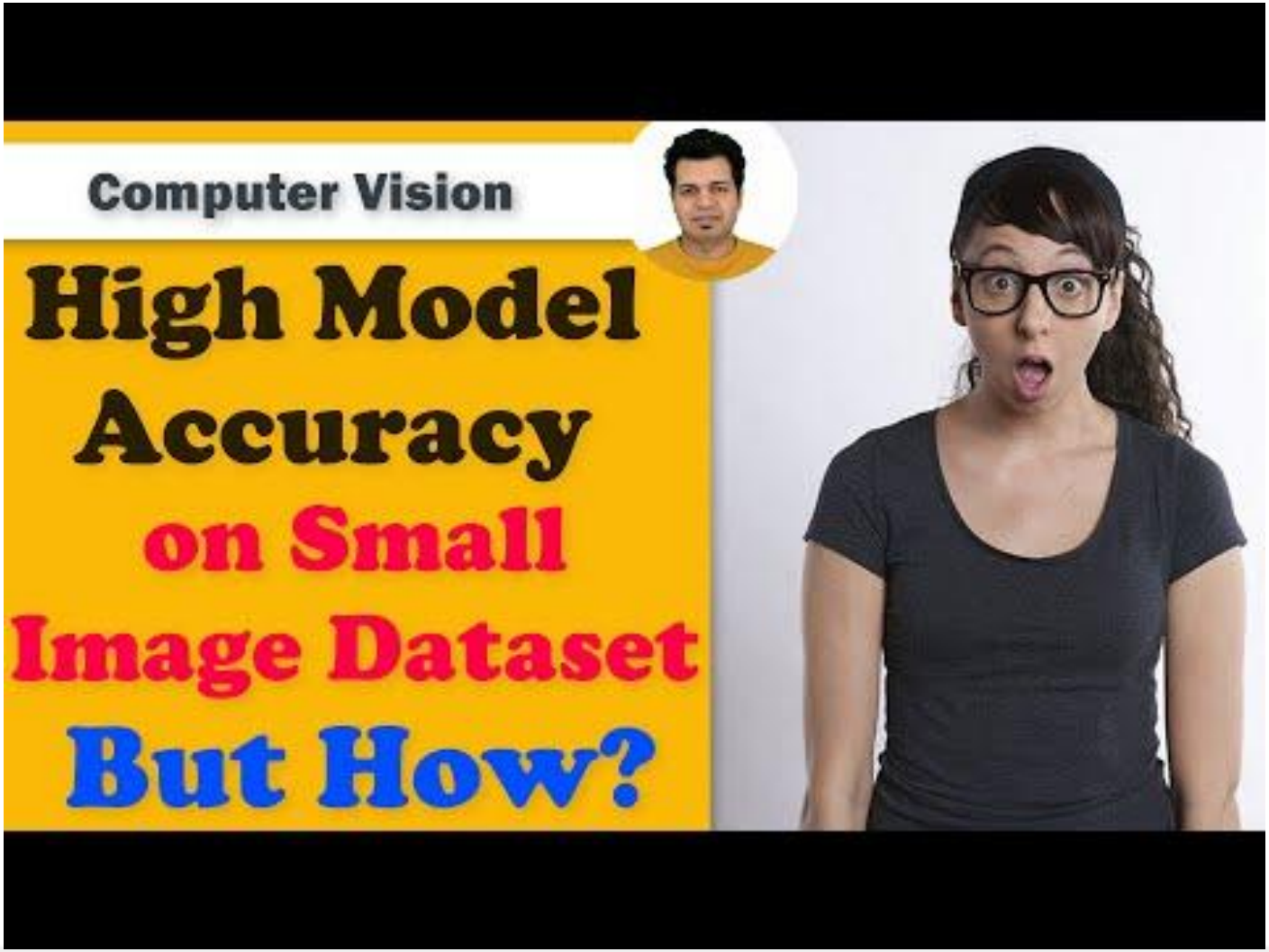
An ImageDataGenerator which performs augmentation might look like this:

```
train_datagen = ImageDataGenerator(
      rotation_range=20,
      width_shift_range=0.2,
      height_shift_range=0.2,
      shear_range=0.2,
      zoom_range=0.2,
      horizontal_flip=True,
      vertical_flip=True,
      fill_mode='nearest')
```

Look at the variety of augmentation operations available!

Note: Apart from ImageDataGenerator, there exist multiple other options of performing Data Augmentation as well.

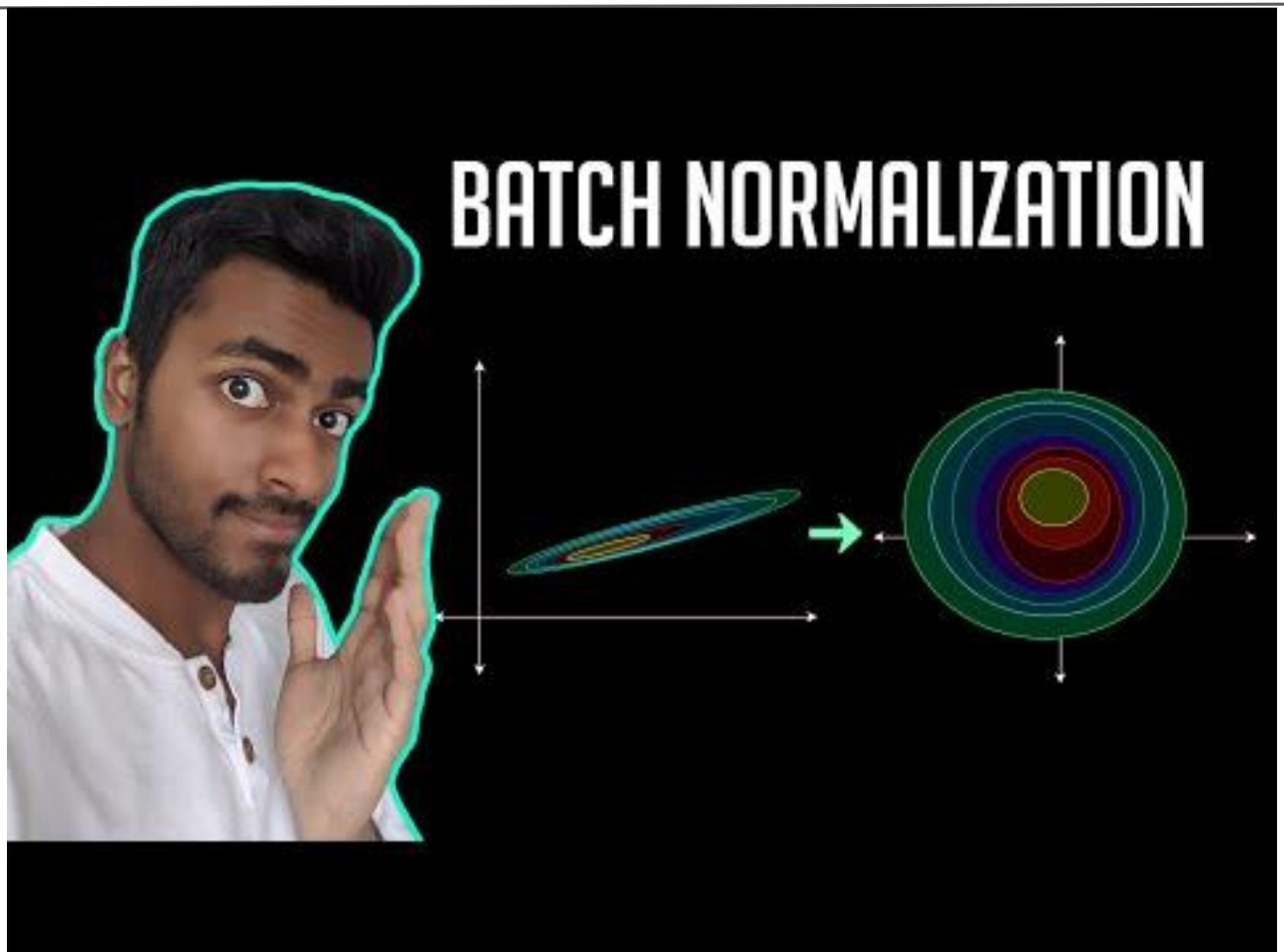
How to perform Data Augmentation?



Batch Normalization

- Making models faster and more stable

Batch Normalization: What and Why?



Batch Normalization: How?

Applying Batch Normalization is as simple as adding a layer called BatchNormalization() after the convolution layer. It can be added to any Neural Network.

```
# Create the model
model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=input_shape)
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(BatchNormalization())
model.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(BatchNormalization())
model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(BatchNormalization())
model.add(Dense(no_classes, activation='softmax'))
```

Additional Material

A nice article throwing light on the variety of Data Augmentation techniques in practice:

https://towardsdatascience.com/exploring-image-data-augmentation-with-keras-and-tensorflow-a8162d89b844

Slide Download Link

- You can download the slides here: https://docs.google.com/presentation/d/1gay6jH7mfZIRjAPtt9gDqKoAlx5wSaDOEbDtmQToUAY/edit?usp=sharing

References

- https://towardsdatascience.com/what-is-transfer-learning-8b1a0fa42b4
- https://medium.com/nanonets/nanonets-how-to-use-deep-learning-when-you-have-limited-data-f68c0b512cab
- https://nanonets.com/blog/data-augmentation-how-to-use-deep-learning-when-you-have-limited-data-part-2/

That’s it for the day. Thank you!

Feel free to post any queries on Discuss
or in the #help channel on Slack