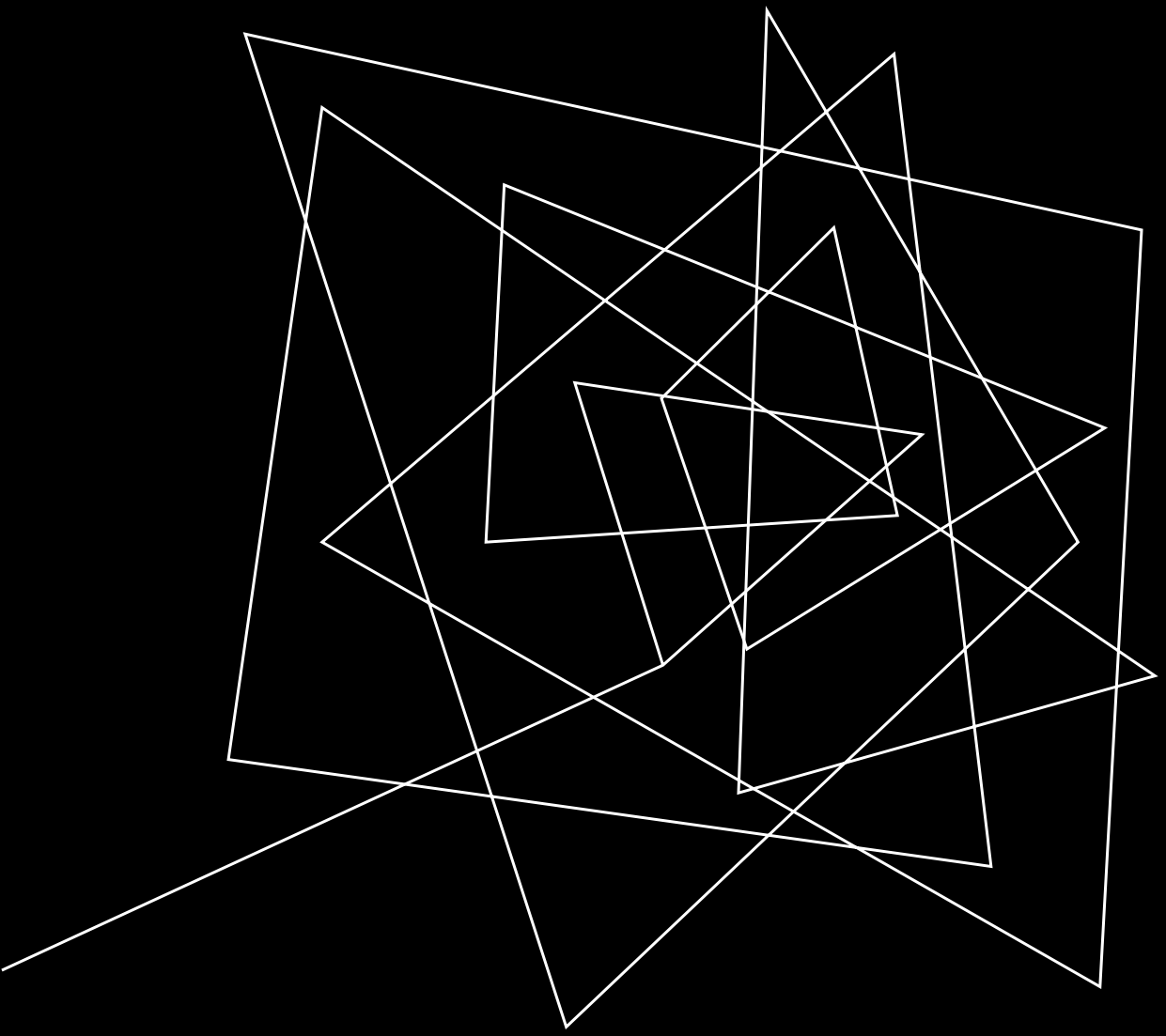
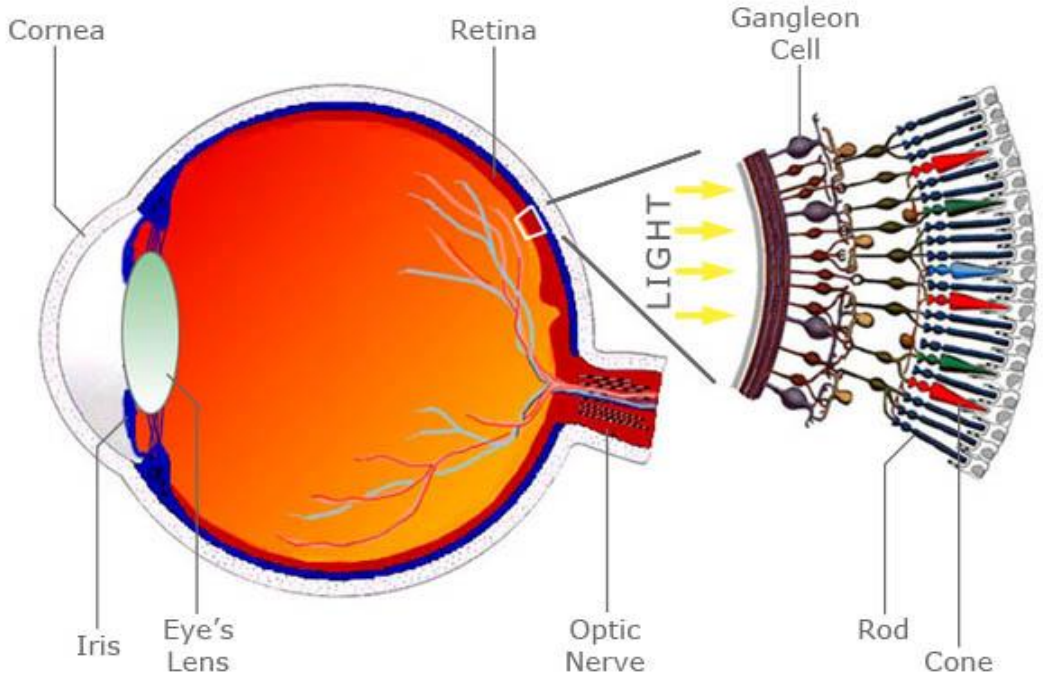


DEEP LEARNING FOR COMPUTER VISION

Convolution Neural Networks (CNNs), CNN Architectures, Transfer Learning, GANs, and Data Augmentation

Introduction: Computer Vision

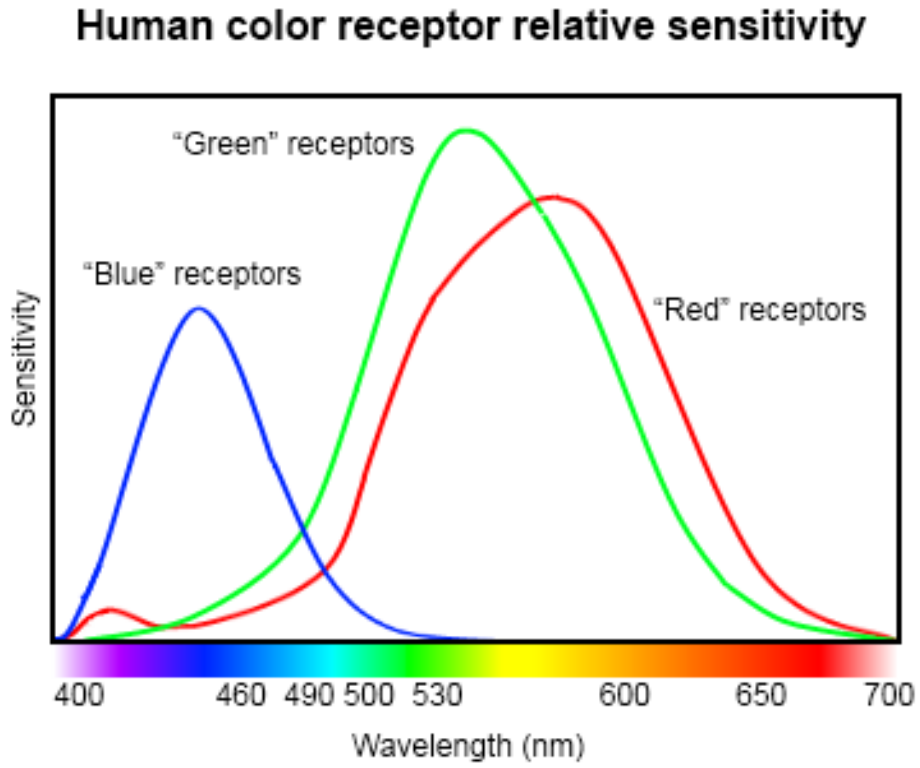
- The human eye (vision) is one of the most important parts that process the information visualized by the human (Information/Image Processing).
- **Idea:** In the human eye, there is what we call "**Cone Cell**", and these cells receive the light from the **Retina**
  - These "Cone" cells are sensitive to some **colors (electrical signals).**
  - These "**Cone**" cells are important because they contribute to the "Color perception/cognition"



Source: https://www.blueconemonochromacy.org/how-the-eye-functions/

Introduction: Computer Vision

- Generally, researchers found that the human being has a **Tri-color vision**.
- The cone cells are sensitive to **RED**, **GREEN**, and **BLUE**
- This can be seen as a projection on a 3-D space, each wavelength can be represented using 3 values (a ratio) **Red**, **Green**, and **Blue**



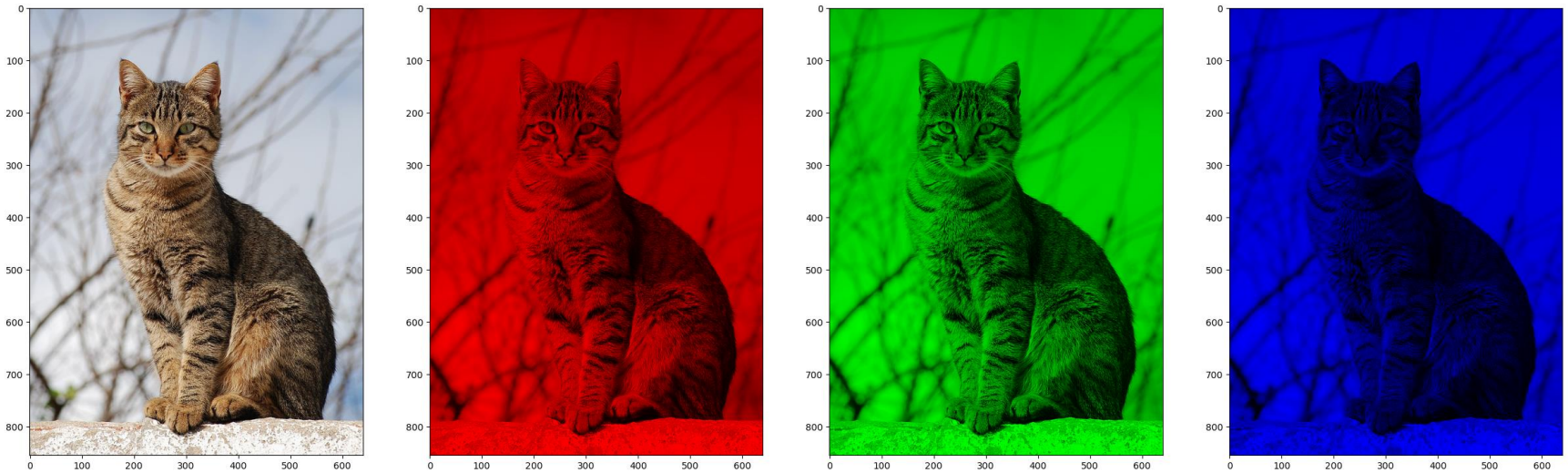
Introduction: Computer Vision

The computers use **RGB** representation to represent images. Each picture you see on your device is represented under a 3D matrix representing the values of **Red**, **Blue**, and **Green** for each pixel.

Source: https://fr.mathworks.com/help/images/image-types-in-the-toolbox.html
You can try it here: https://pixspy.com/

Introduction: Computer Vision

Question: How to extract **RGB** values from an image?



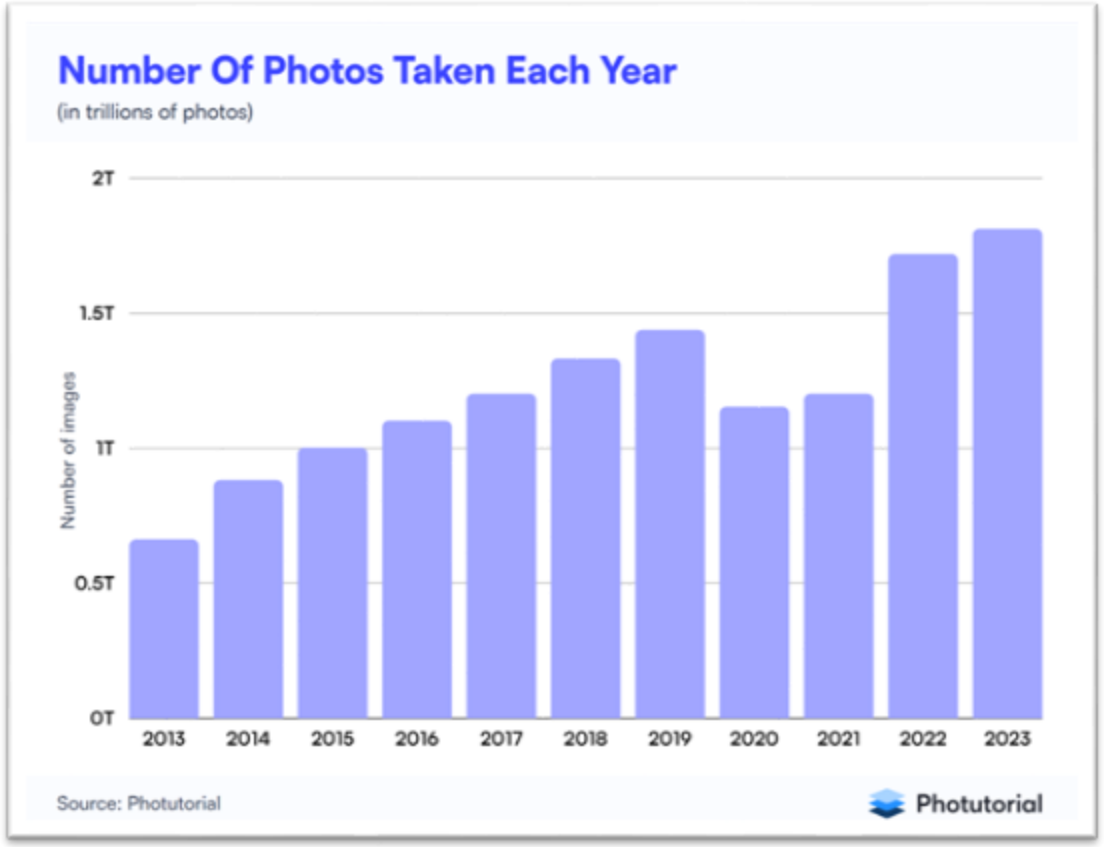
Demo: Session 3 – Computer Vision and CNN

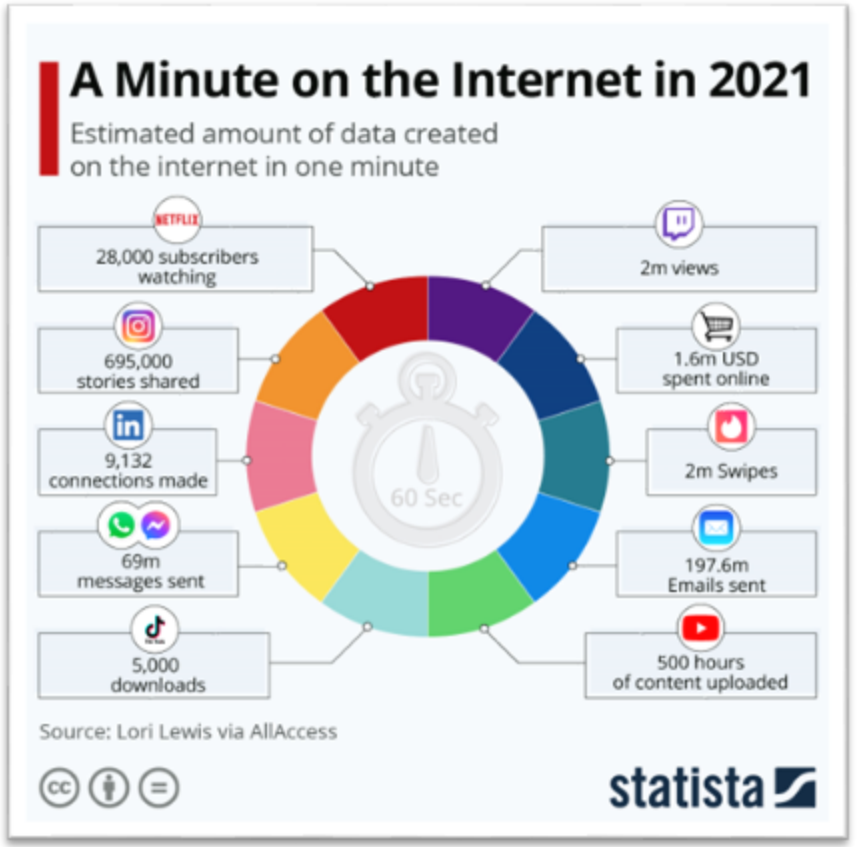
Introduction: Computer Vision
<DATA EXPLOSION>



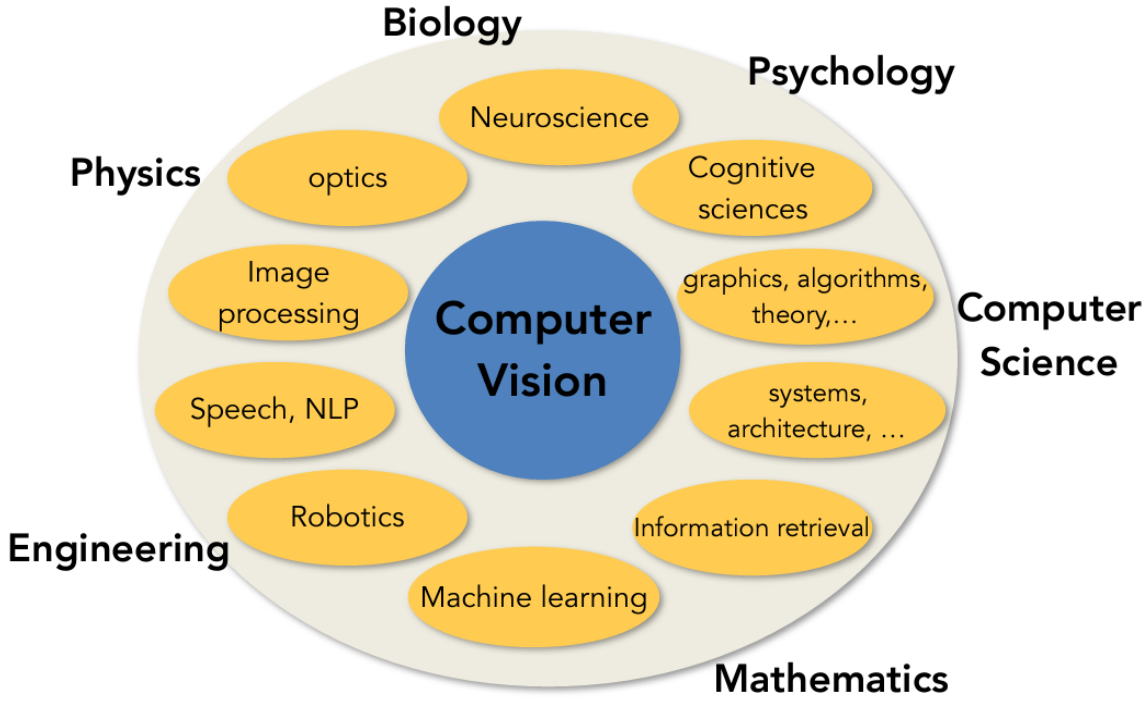



Introduction: Computer Vision
<DATA EXPLOSION>

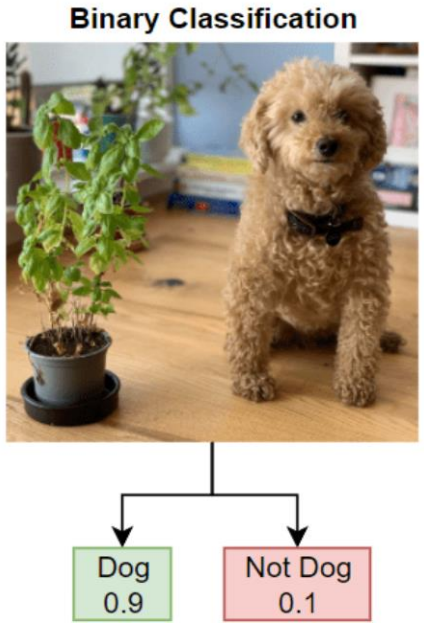


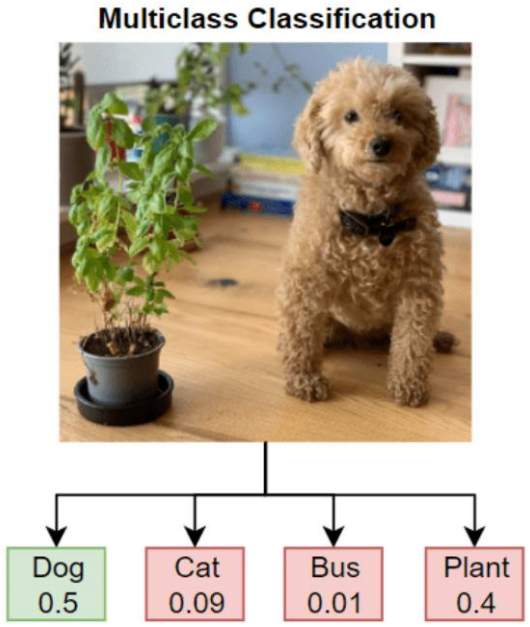


Introduction: Computer Vision <Interdisciplinary Field>



Computer Vision: Image Classification





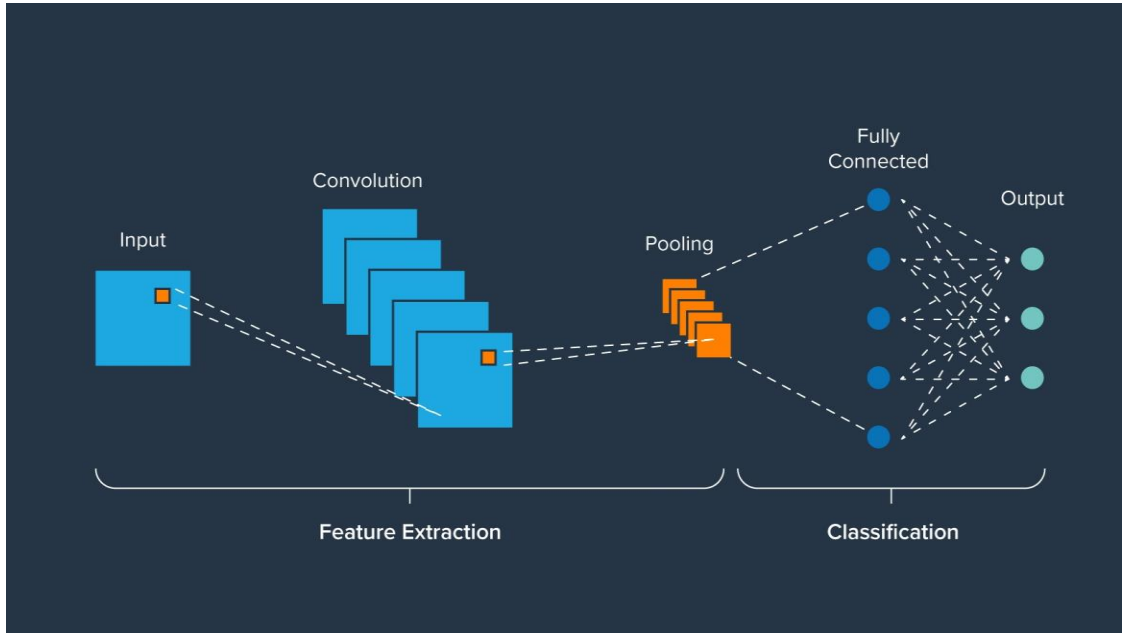


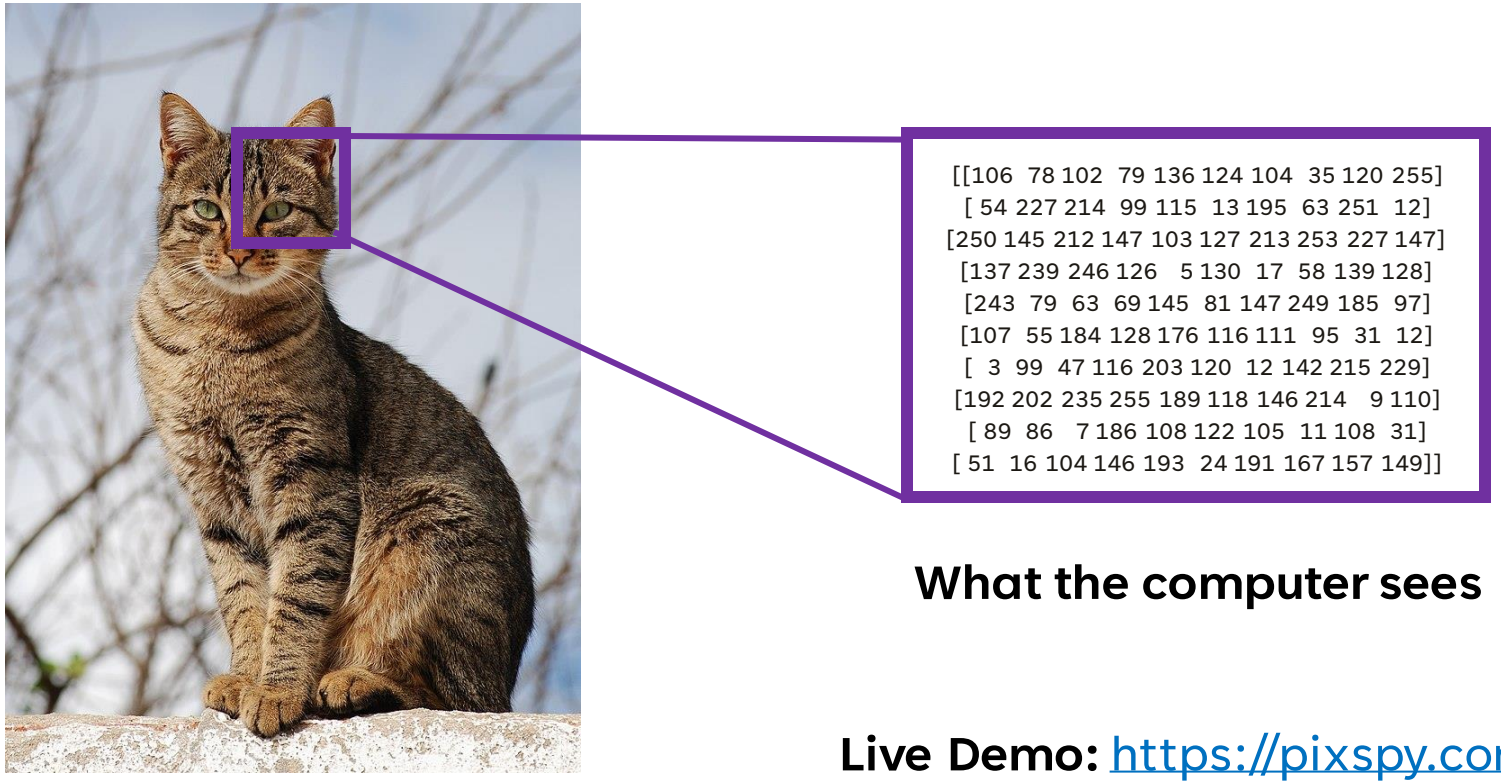
Image Classification: A Core Task in Computer Vision

Assume given set of discrete labels = {Dog, Cat, Truck, Plane, ...etc.}

→ **Cat**

Image Classification: A Core Task in Computer Vision

The Problem: Semantic Gap



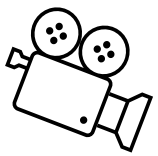
What the computer sees

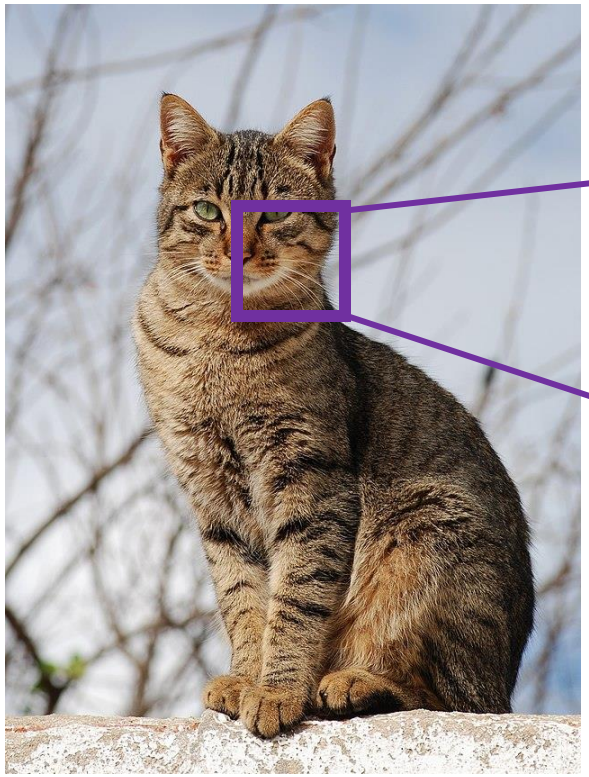
An image is just a big grid of numbers between [0, 255]: e.g., 800 * 600 * 3 (3 channels RGB)

Live Demo: https://pixspy.com/

Image Classification: A Core Task in Computer Vision

Challenges: Viewpoint variation





```
[[106  78 102  79 136 124 104  35 120 255]
 [ 54 227 214  99 115  13 195  63 251  12]
 [250 145 212 147 103 127 213 253 227 147]
 [137 239 246 126   5 130  17  58 139 128]
 [243  79  63  69 145  81 147 249 185  97]
 [107  55 184 128 176 116 111  95  31  12]
 [  3  99  47 116 203 120  12 142 215 229]
 [192 202 235 255 189 118 146 214   9 110]
 [ 89  86   7 186 108 122 105  11 108  31]
 [ 51  16 104 146 193  24 191 167 157 149]]
```

What the computer sees

All the pixels **change** when the **camera moves**. But somehow it is **still representing** the same object, "**Cat**". Our **algorithms** need to be **robust** and **adapted** to this.

Image Classification: A Core Task in Computer Vision

Other Challenges: Illumination and Deformation

Image Classification: A Core Task in Computer Vision

Other Challenges: Occlusion and Background clutter

Other Challenges: Interclass variation

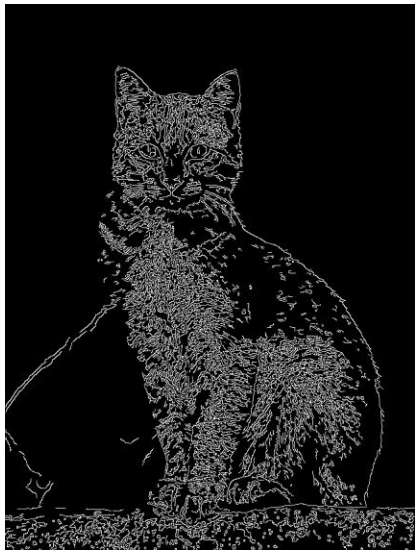
Image Classification: An Image Classifier?

```
def classify_image(image):
    # Some magic here?
    return class_label
```

In **traditional programming** techniques, there is **no obvious** way to hard code the **image classifier** algorithm.

Image Classification: Attempts have been made!

Find edges



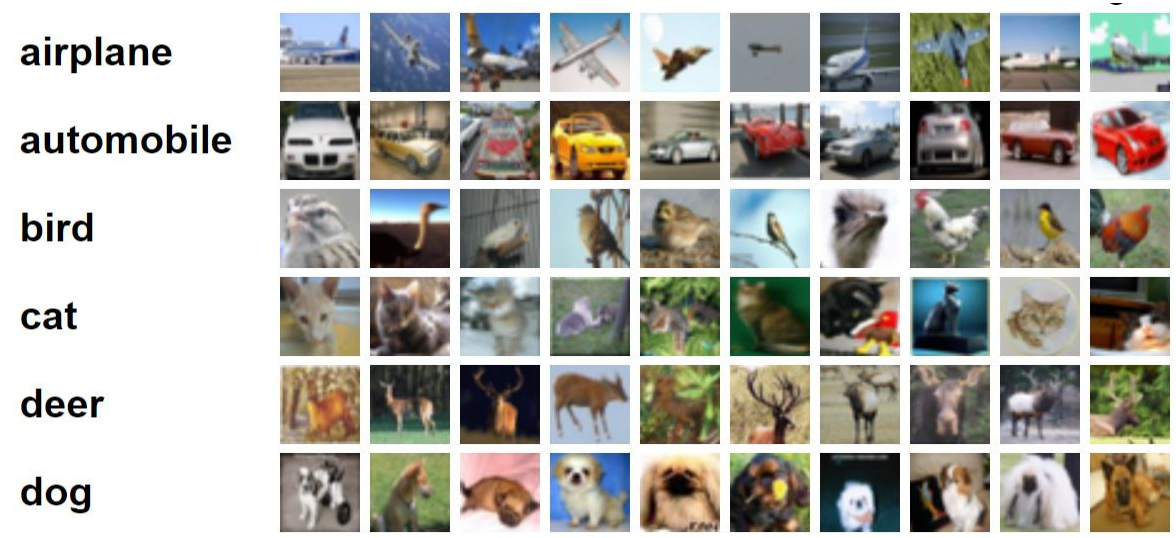
Find corners

Up, Down, Left, Right

John Canny, "A Computational Approach to Edge Detection", IEEE TPAMI 1986

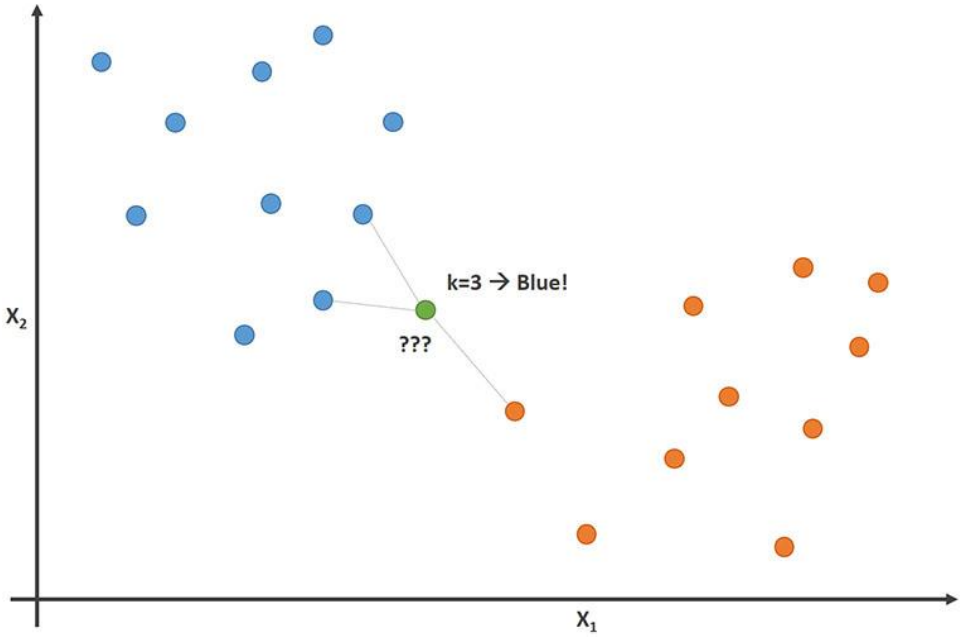
Image Classification: Data-Driven Approach

1. Collect a dataset of images and their corresponding labels
2. Use a machine learning algorithm to train a classifier
3. Test and evaluate the classifier on new (unseen) images



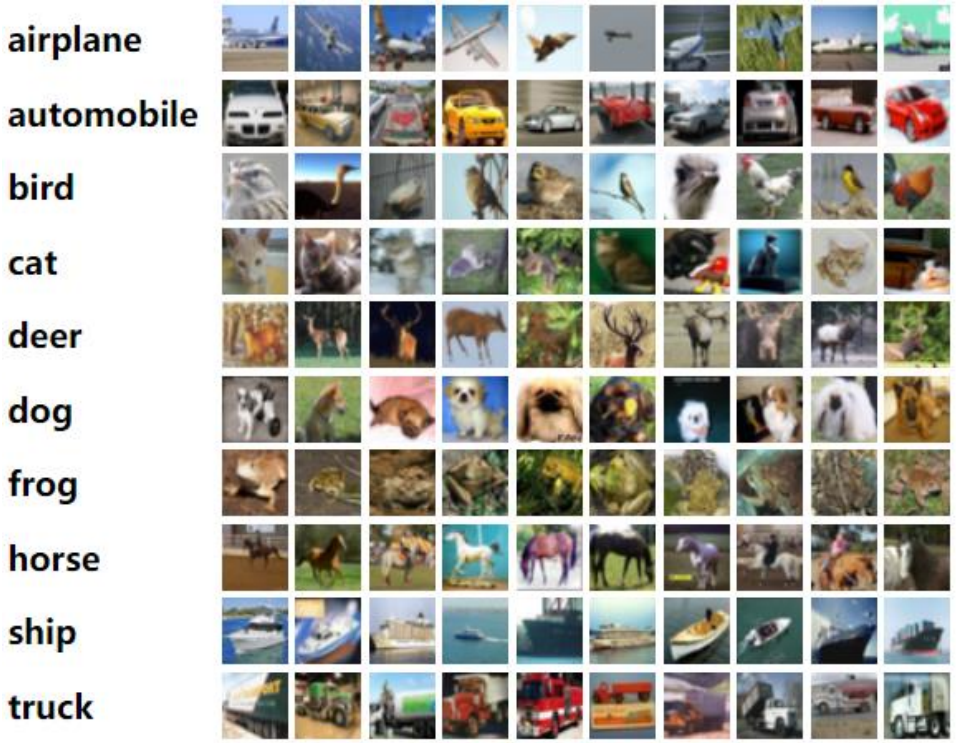
Example of training dataset

Image Classification: First Classifier <Nearest Neighbor>



The k-nearest neighbors' algorithm, also known as KNN or k-NN, is a non-parametric, supervised learning classifier, which uses proximity to make classifications or predictions about the grouping of an individual data point.

Image Classification: First Classifier <Nearest Neighbor>

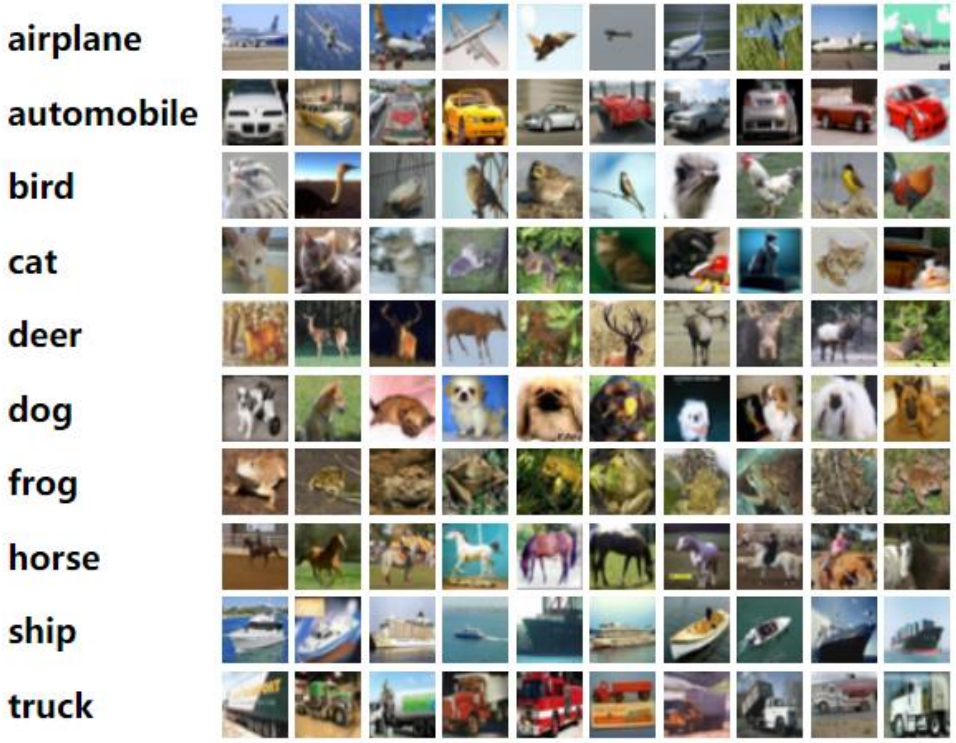


https://www.cs.toronto.edu/~kriz/cifar.html

Example of dataset: CIFAR-10

- The CIFAR-10 dataset (Canadian Institute For Advanced Research) is a collection of images that are commonly used to train machine learning and computer vision algorithms.
- 10 Classes
- 6000 images per class
- 50,000 Training images
- 10,000 Testing images
- 32x32 color images
- They were collected by Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton.

Image Classification: First Classifier <Nearest Neighbor>



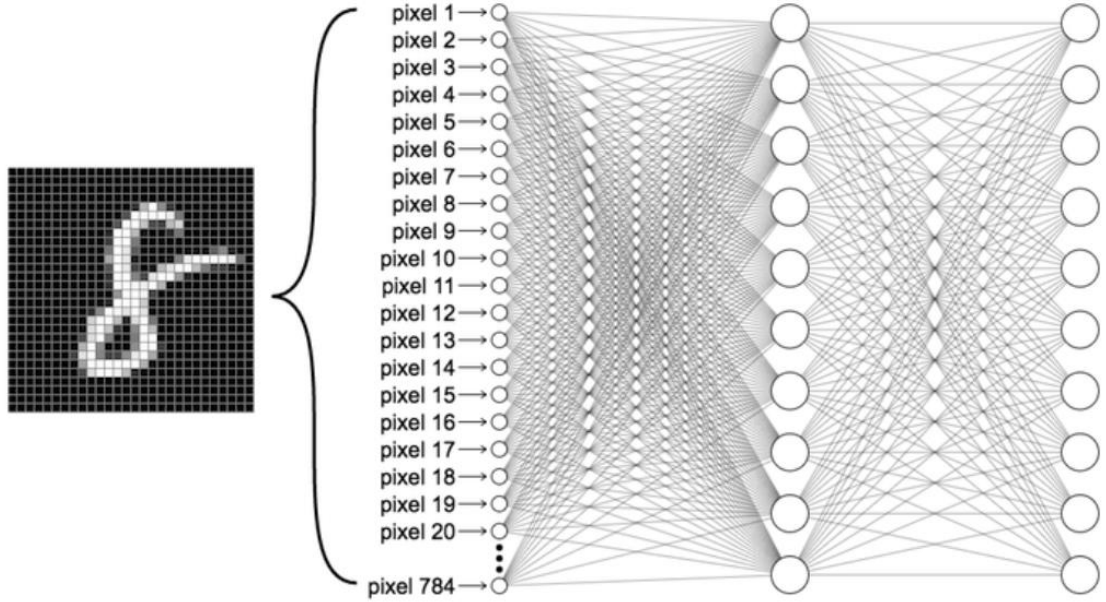
https://www.cs.toronto.edu/~kriz/cifar.html

Test images and nearest neighbors

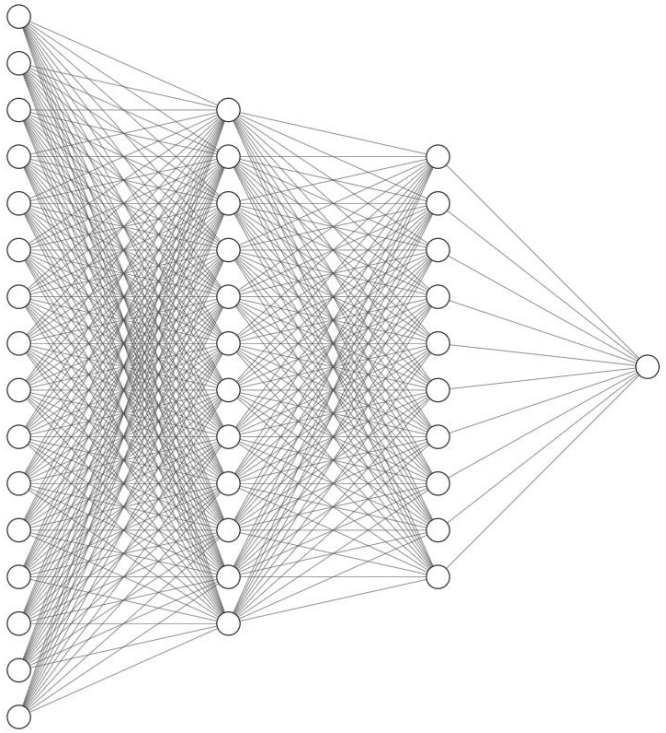
Computer Vision: Why The MLPs (or ANNs) are not Good for Images?

In the last session **(Session 2: Deep Learning)**, we used the MLP to classify the MNIST dataset, A reminder:

- Small images with 28*28 pixels
- The input layer has 784 input neurons
- Two hidden layers
- 10 output neurons
- **The total number of parameters: 235,146**



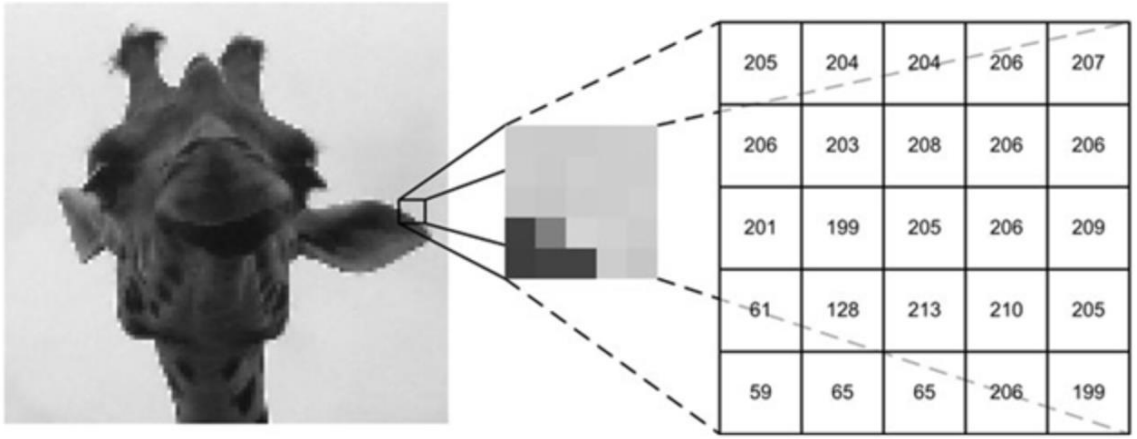
Computer Vision: Why The MLPs (or ANNs) are not Good for Images?

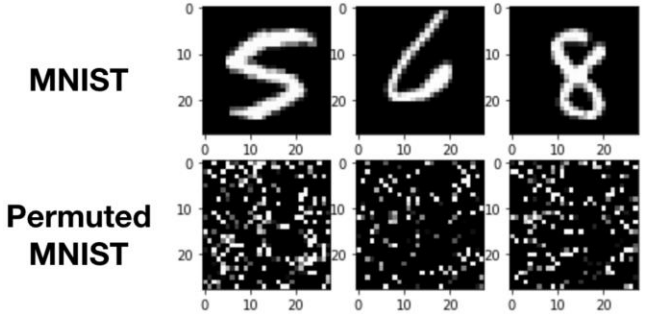


Now imagine, we want to build an MLP for RGB images.

- 855*640*3 Input layer
- 2 Hidden layers with 512 neurons each
- One output neuron
- **The total number of parameters: 840,762,881**

Computer Vision: Why The MLPs (or ANNs) are not Good for Images?



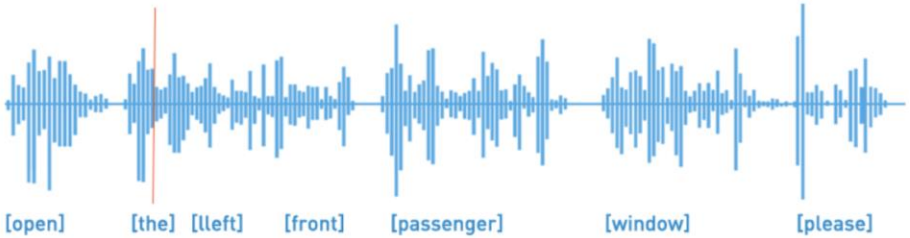


Another problem with MLPs for images, missing neighborhood information (Neighborhood and image structure)

Demo: Session 3 – Computer Vision and CNN

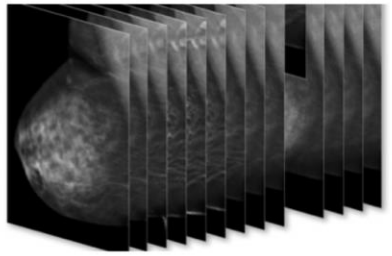
Computer Vision: The Importance of Signal Structure

1D

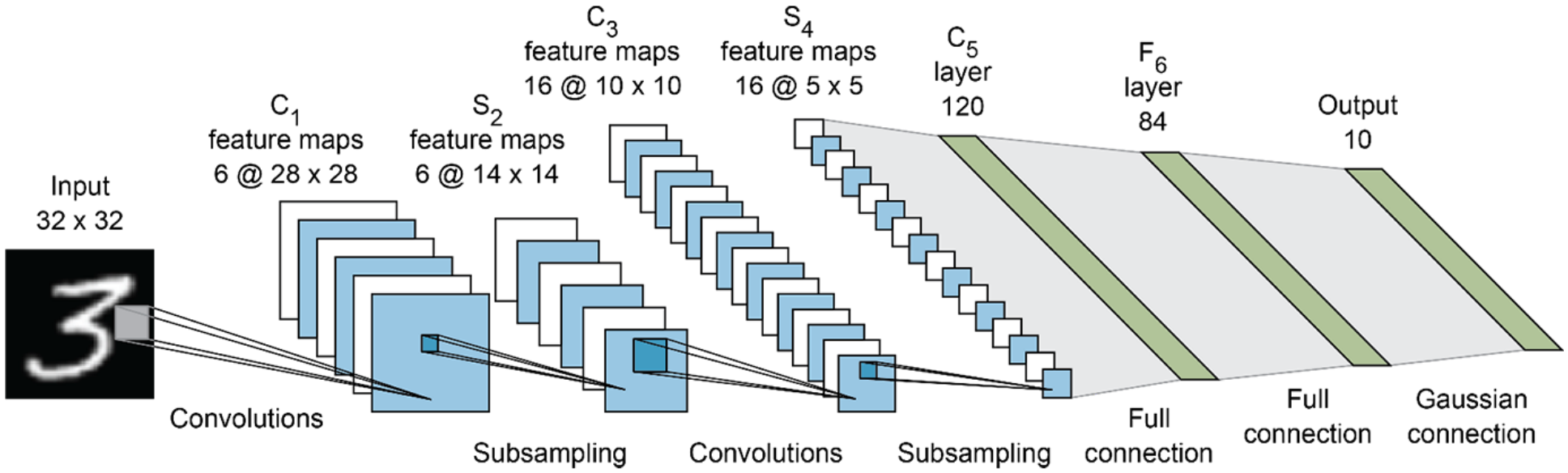


2D

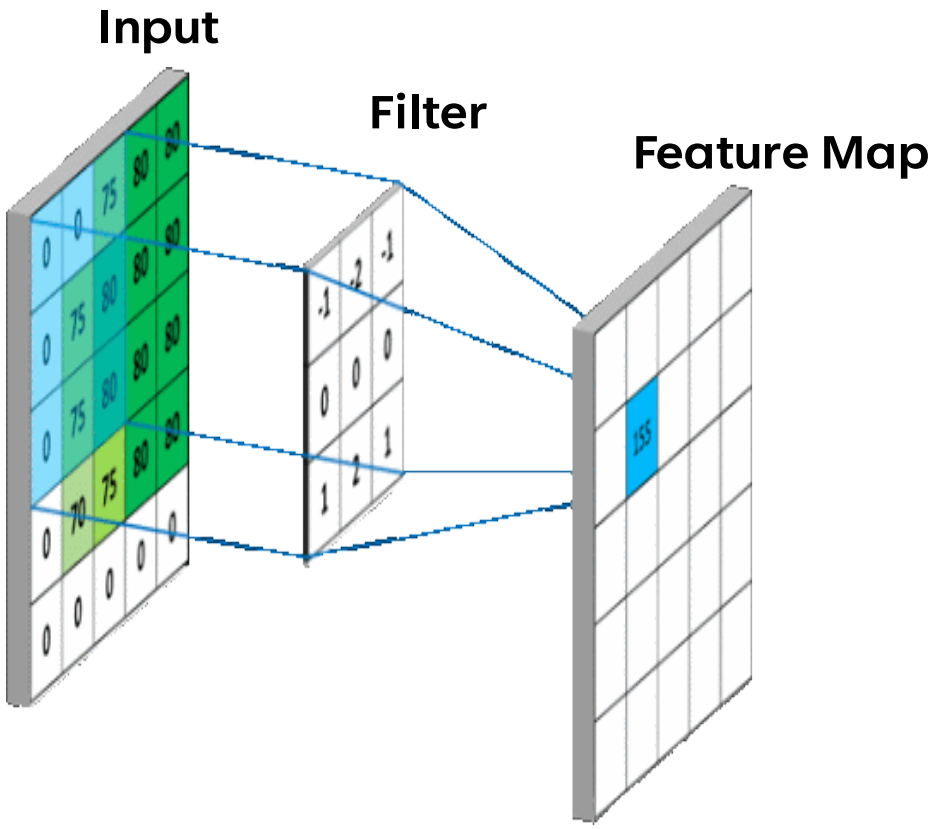
3D



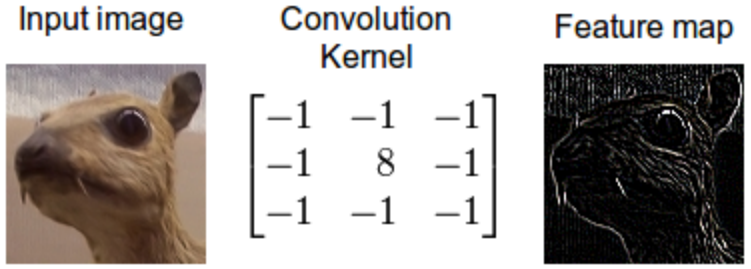
Convolutional Neural Networks (CNNs)



Convolutional Neural Networks (CNNs)



- The term convolution refers to **the mathematical combination of two functions to produce a third function**. It merges two sets of information. In the case of a CNN, the convolution is performed on the input data with the use of a filter or kernel (these terms are used interchangeably) to then produce a feature map.
- It is a very important technique to find patterns in images and image processing



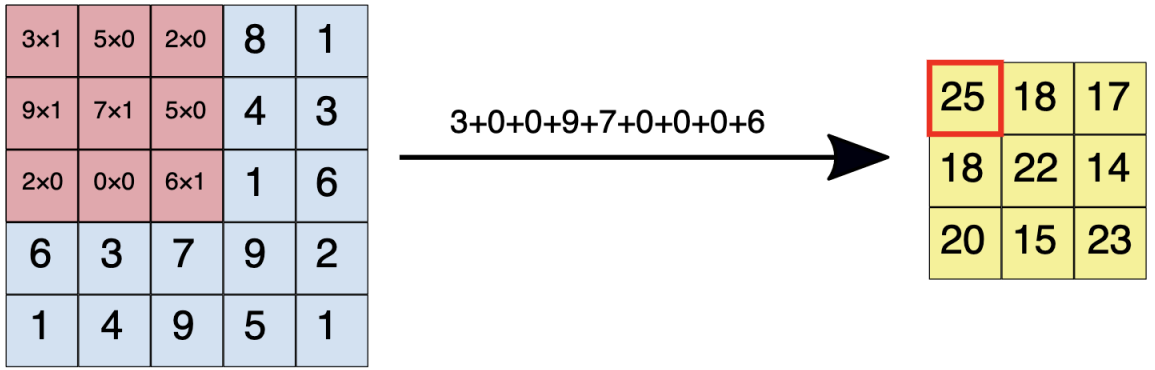
Convolutional Neural Networks (CNNs): Filter

Input

3	5	2	8	1
9	7	5	4	3
2	0	6	1	6
6	3	7	9	2
1	4	9	5	1

1	0	0
1	1	0
0	0	1

Filter/Kernel/Feature detector



Result: Feature map

Note: Feature map **output size** = [**size** of the **input** - **size** of the kernel/filter] + 1

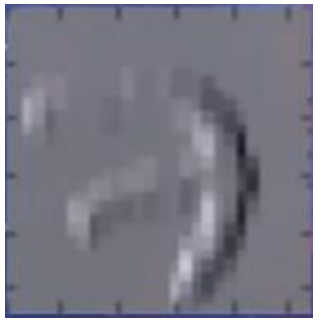
Convolutional Neural Networks (CNNs): Filter

Filter 1

-1	-1	-1
1	1	1
0	0	0

Filter 2

-1	1	0
-1	1	0
-1	1	0



Filter 3

0	0	0
1	1	1
-1	-1	-1

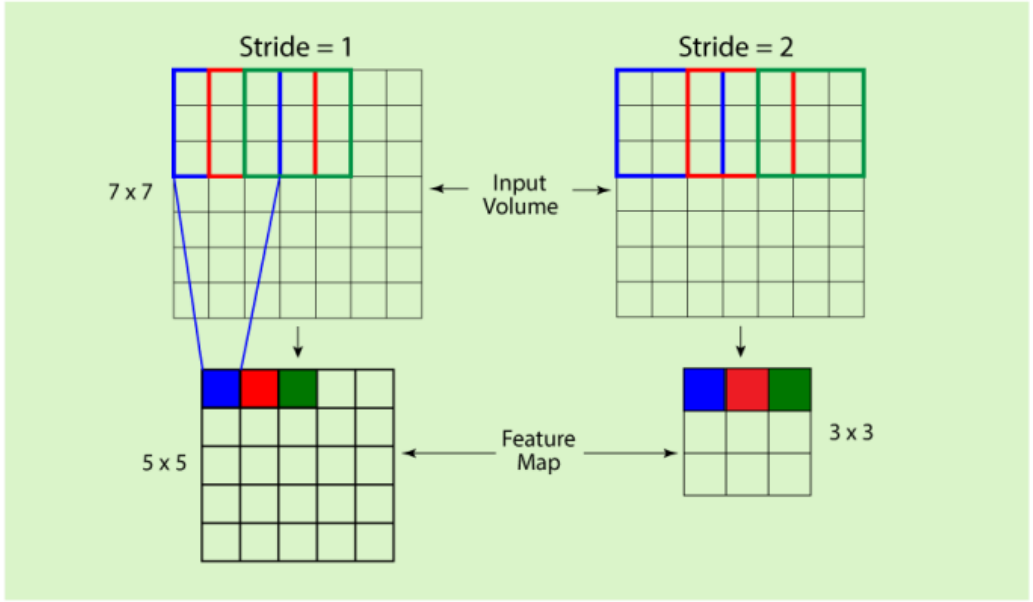
Filter 4

0	1	-1
0	1	-1
0	1	-1

Source: [Convolutional Neural Networks (CNNs) explained]

Convolutional Neural Networks (CNNs): Stride

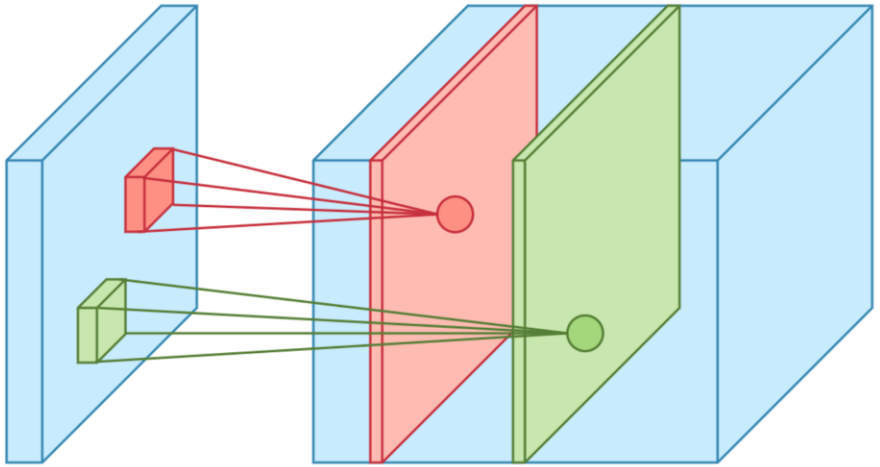
- The filter is moved across the image left to right, top to bottom, with a one-pixel column change on the horizontal movements, then a one-pixel row change on the vertical movements.
- The amount of movement between application of the filter to the input image is referred to as the **STRIDE.**



Note: Feature map **output size** = { [**size** of the **input** - **size** of the kernel/filter] / **STRIPE** } + 1

Convolutional Neural Networks (CNNs): ConvLayers

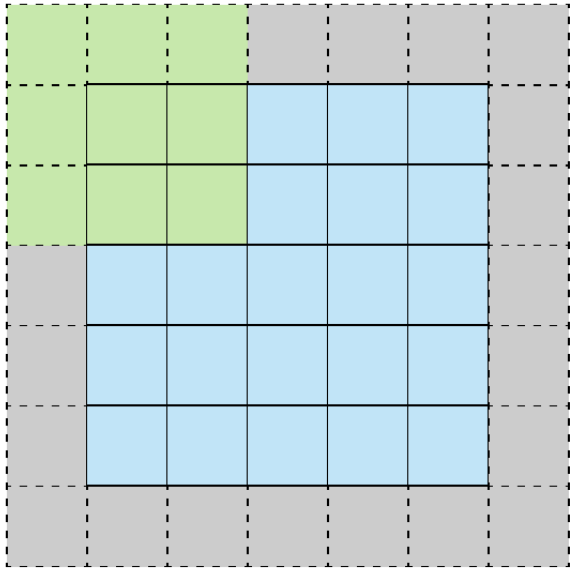
Conv Layer



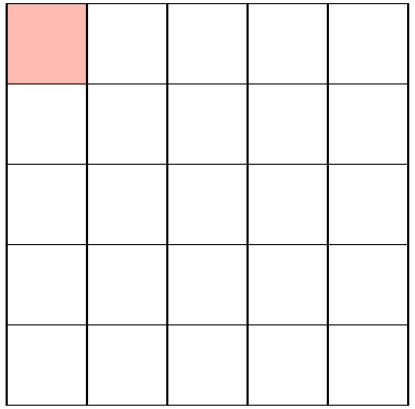
- Convolutional neural networks are a type of neural network in which we are trying to **find** the **filters** and **biases** that **minimize** the **loss function**.
- **Conv Layer** contains different filters/kernels with different lengths and widths.
- The filters are **computed** by the **CNN**, and that is what makes the **CNN** more **powerful**.

Convolutional Neural Networks (CNNs): Padding

Some employed techniques in **CNNs**: **Padding**



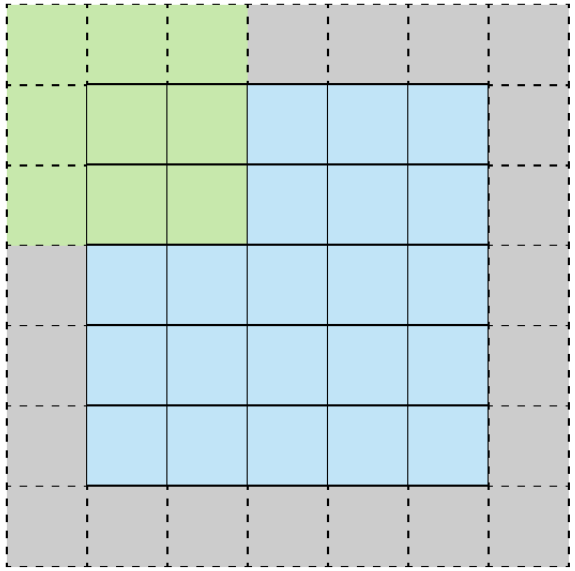
Stride 1 with Padding



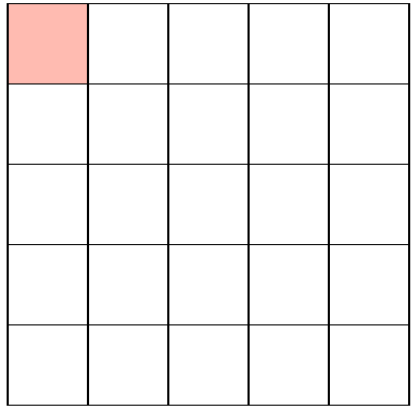
Feature Map

Convolutional Neural Networks (CNNs): Padding

- Padding is the best approach, where the number of pixels needed for the convolutional kernel to process the edge pixels are added onto the outside.
- Fixing the border effect problem with padding



Stride 1 with Padding

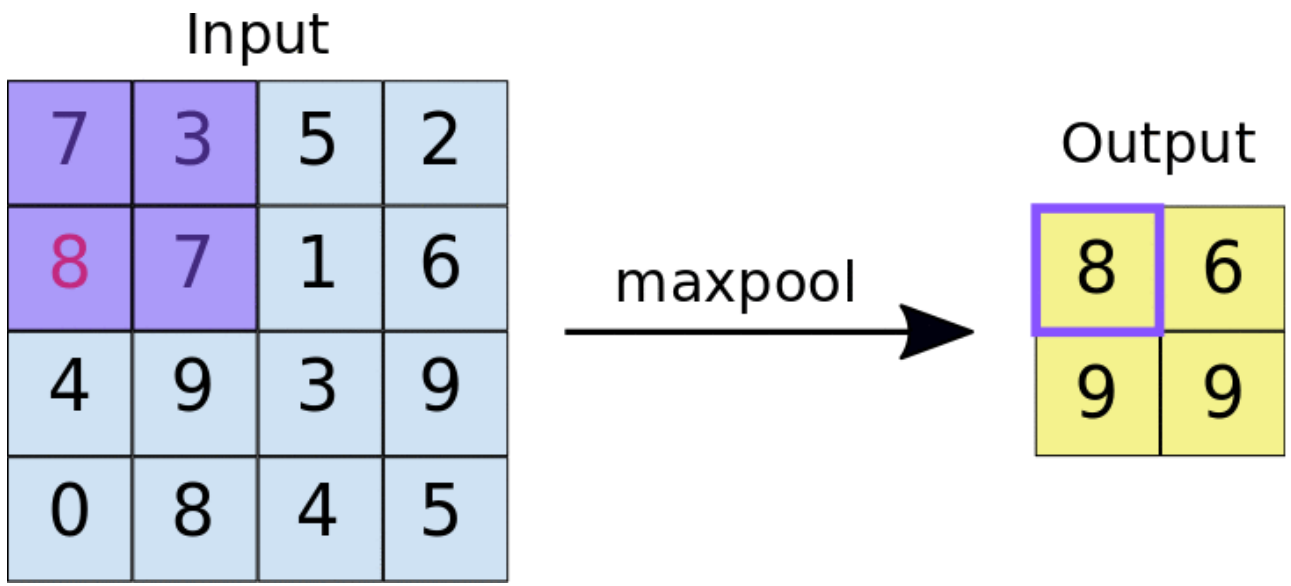


Feature Map

Convolutional Neural Networks (CNNs): Pooling

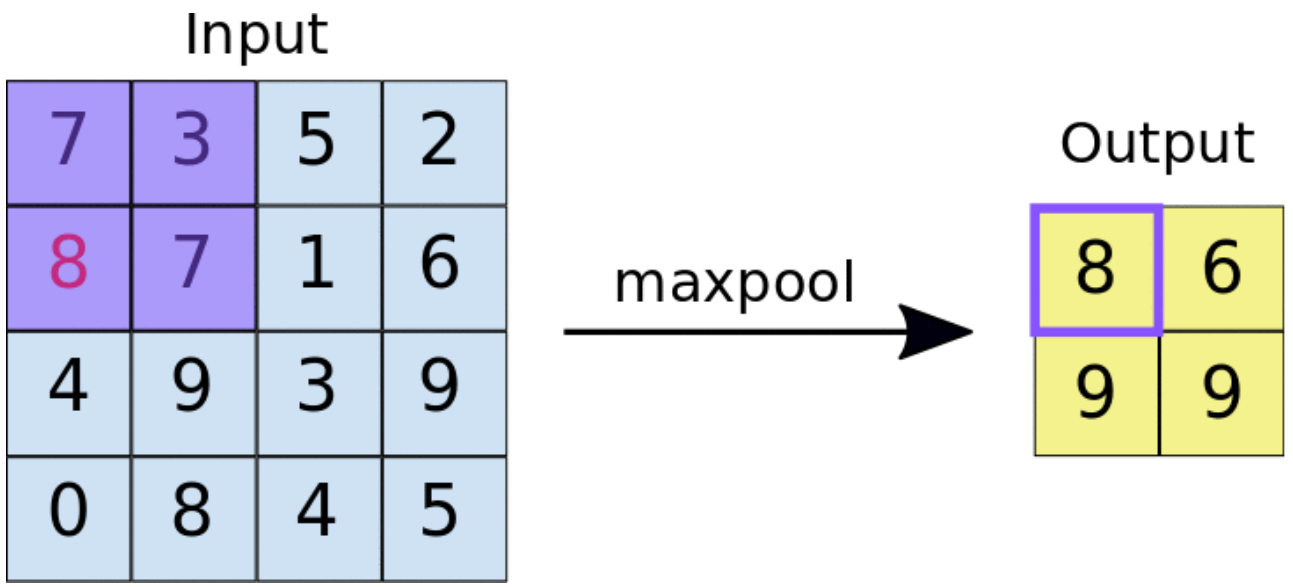
Some used techniques in **ConvNets**: **Pooling**

- **Pooling** is required to down sample the detection of features in feature maps.
- **Pooling layers** provide an approach to down sampling feature maps by summarizing the presence of features in patches of the feature map.
- Two common pooling methods are **average pooling** and **max pooling**

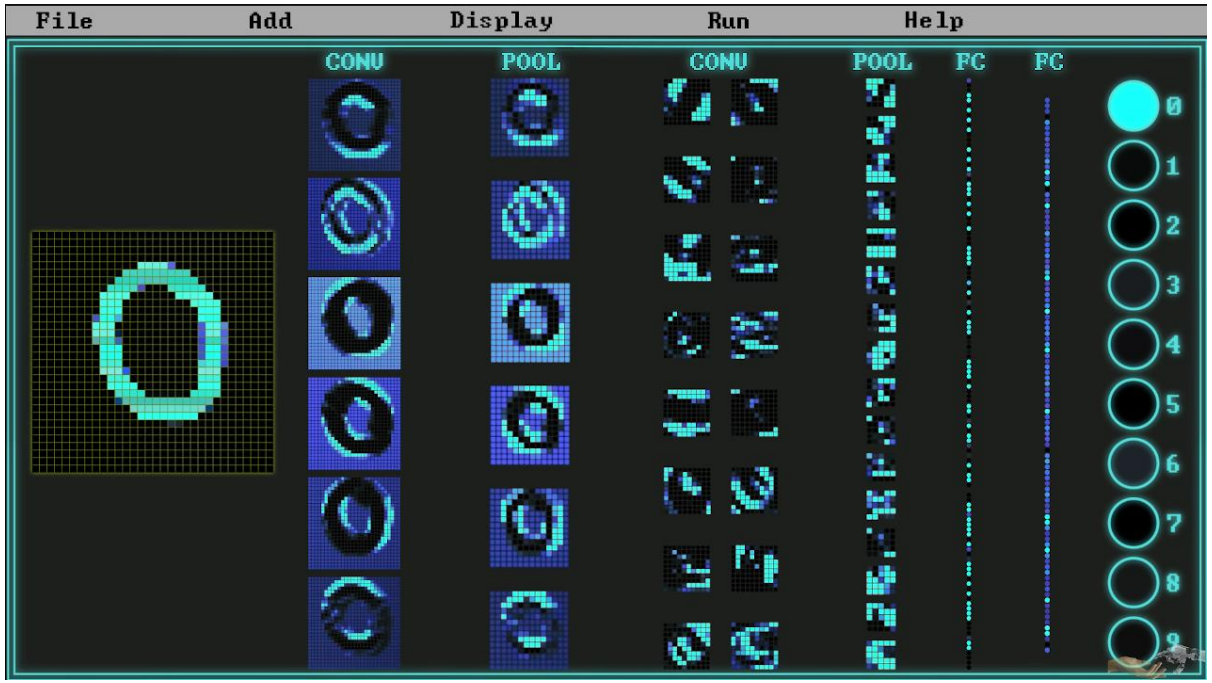


Convolutional Neural Networks (CNNs): MaxPooling

Some used techniques in **ConvNets**: **MaxPooling**

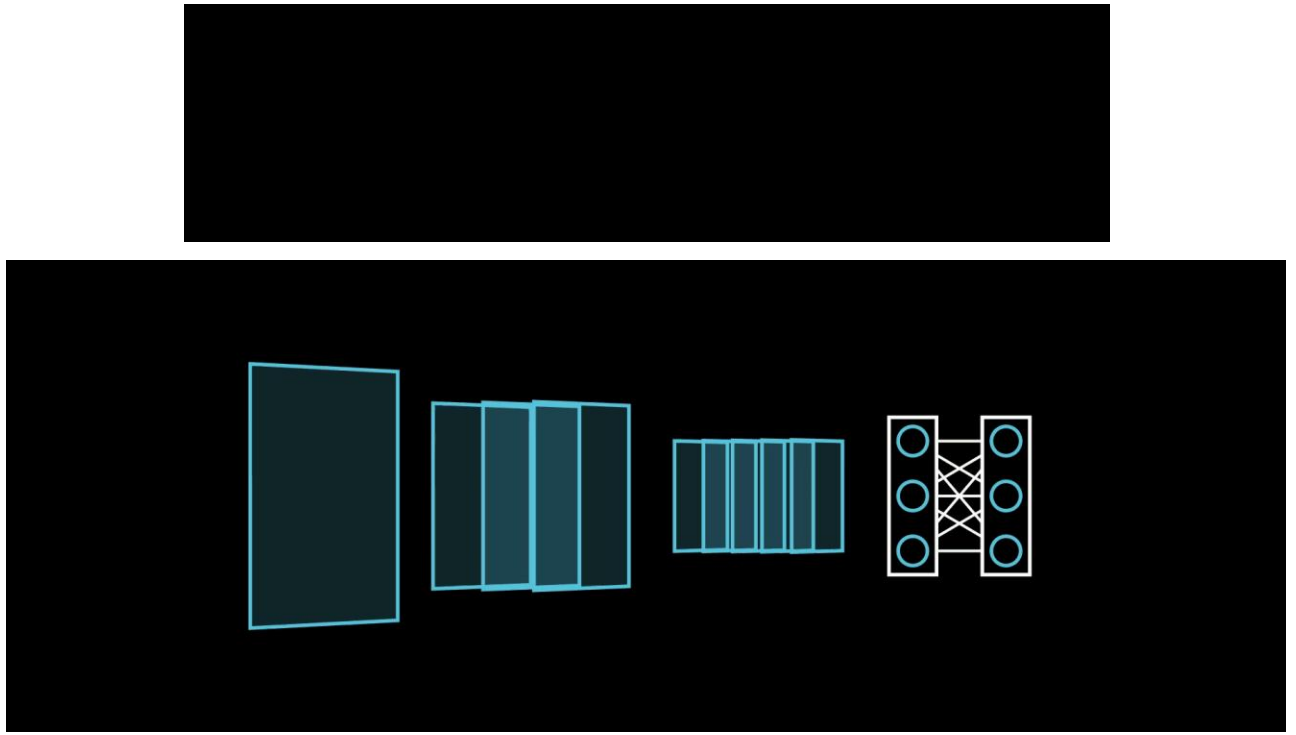


Convolutional Neural Networks (CNNs): CNN Visualized <DEMO>



[Convolutional Neural Networks Explained (CNN Visualized)]

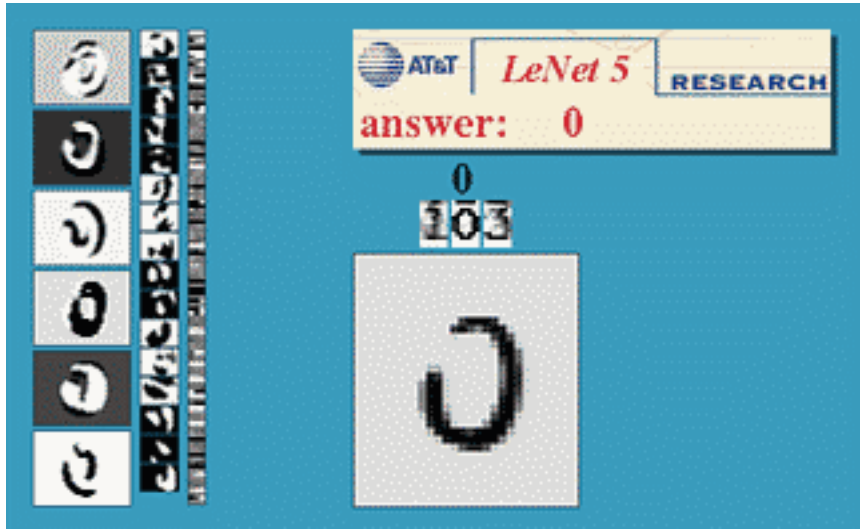
Convolutional Neural Networks (CNNs): CNN Visualization



ManimML Library Link: https://github.com/helblazer811/ManimML

Convolutional Neural Networks (CNNs): LeNet5

LeNet-5 is a convolutional neural network architecture designed for handwritten digit recognition. It was introduced by **Yann LeCun**, Léon Bottou, Yoshua Bengio, and Patrick Haffner in the paper titled "**Gradient-Based Learning Applied to Document Recognition**" published in **1998**.

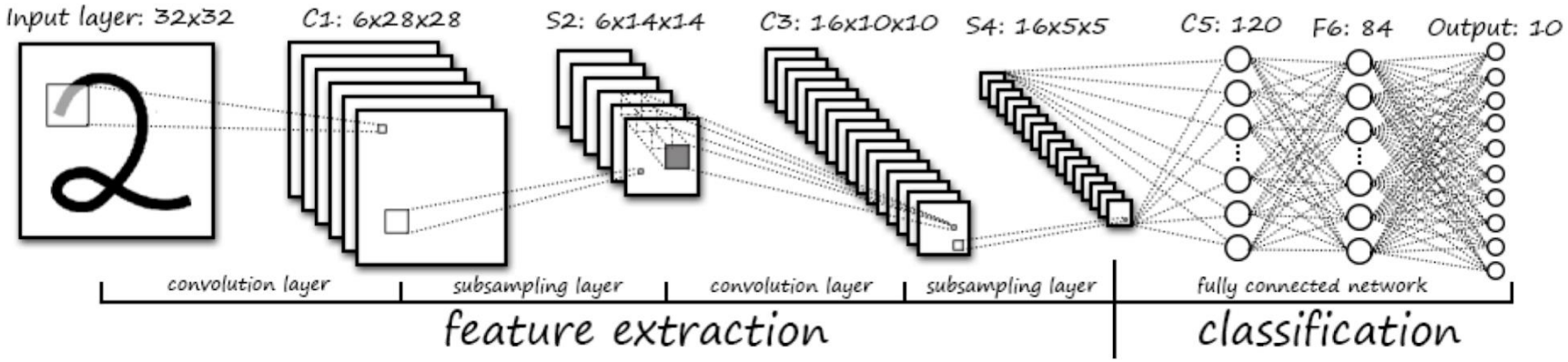


LeNet5 (1998)

Yann LeCun

Convolutional Neural Networks (CNNs): LeNet5

LeNet5 Architecture(1998)

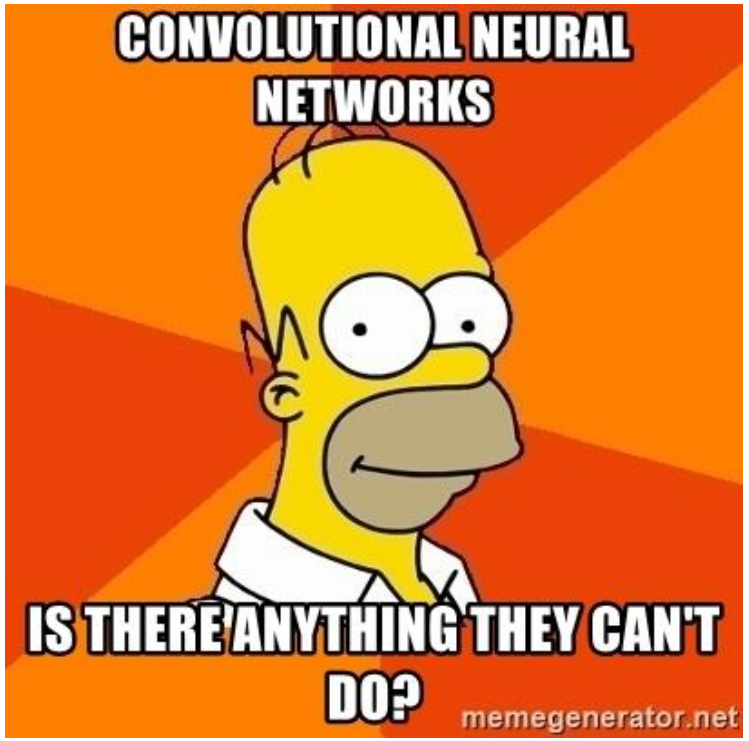


Convolutional Neural Networks (CNNs): LeNet5

LeNet5 Architecture(1998)

1. **Convolutional Layers**: LeNet-5 consists of seven layers, including two convolutional layers. The first convolutional layer applies six filters with a 5x5 receptive field, followed by a 2x2 max-pooling operation.
2. **Activation Functions**: In LeNet-5, the hyperbolic tangent function (tanh) is used as the activation function.
3. **Subsampling Layers**: Following the first convolutional layer, LeNet-5 has a subsampling layer (max-pooling) to reduce the spatial dimensions of the feature maps.
4. **Second Convolutional Layer**: The second convolutional layer applies sixteen 5x5 filters to the subsampled feature maps from the first convolutional layer. This is followed by another 2x2 max-pooling operation.
5. **Fully Connected Layers**: After the convolutional layers, there are three fully connected layers. The first fully connected layer has 120 units, the second has 84 units, and the final output layer has 10 units (corresponding to the 10 possible digits).
6. **Flatten Operation**: Between the convolutional and fully connected layers, there is a flatten operation to convert the 3D volume output from the convolutional layers into a 1D vector.
7. **Softmax Activation**: The output layer uses a softmax activation function, which turns the raw scores (logits) into class probabilities.

Convolutional Neural Networks (CNNs) Architectures: DEMO

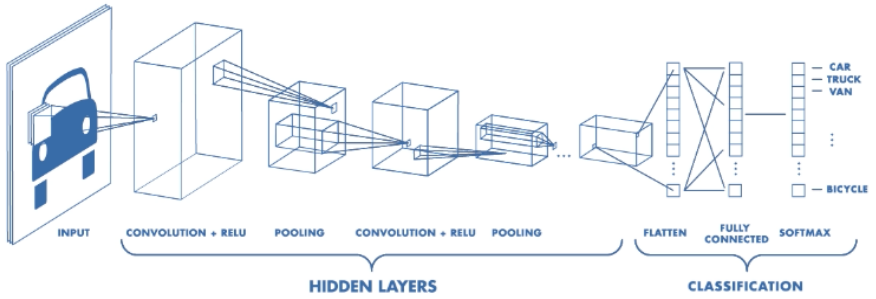


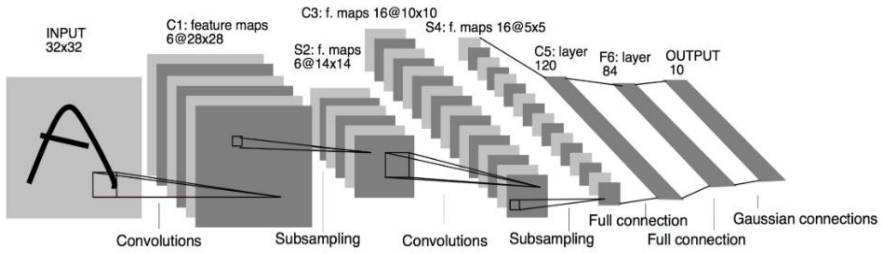
Demo: Session 3 – Computer Vision and CNN

Convolutional Neural Networks (CNNs) Architectures: Classic CNNs

Classic ConvNet Architecture:

- Input
- Conv blocks
  - Convolution + activation (ReLU)
  - Convolution + activation (ReLU)
  - ....
  - Maxpooling 2*2
- Output
  - Fully connected layers
  - Softmax



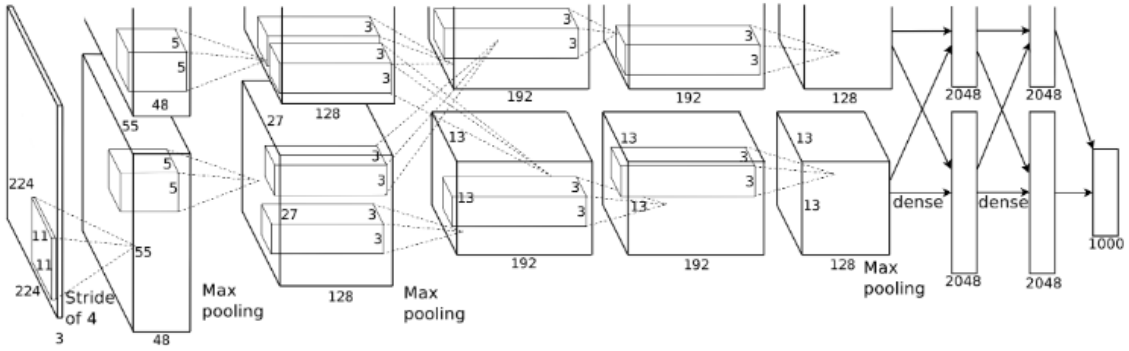


Convolutional Neural Networks (CNNs) Architectures: AlexNet

AlexNet is a deep convolutional neural network architecture designed for image classification tasks. It was developed by **Alex Krizhevsky**, **Ilya Sutskever**, and **Geoffrey Hinton** and **won** the **ImageNet** Large Scale Visual Recognition Challenge (ILSVRC) in **2012**.

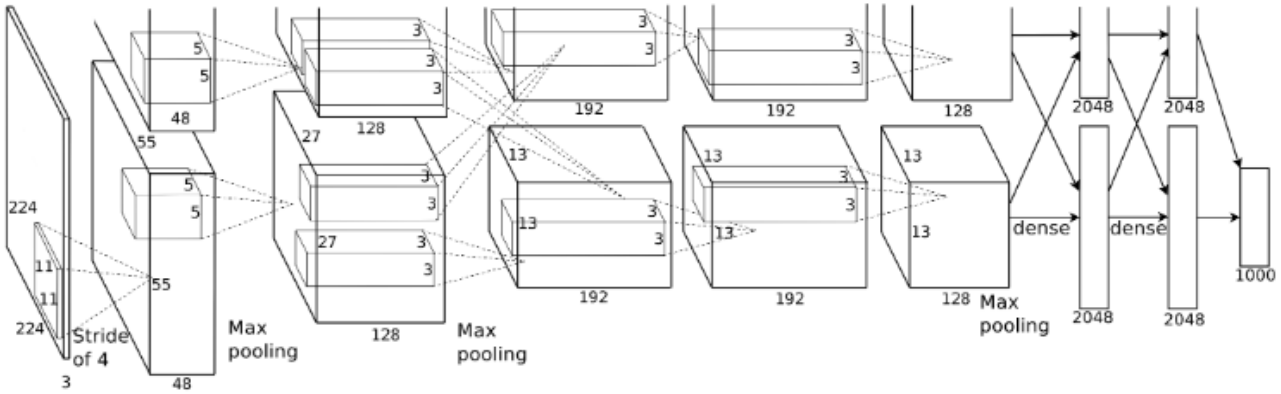
First conv layer: kernel 11x11x3x96 stride 4

- Kernel shape: (11,11,3,96)
- Output shape: (55,55,96)
- Number of parameters: 34,944
- Equivalent MLP parameters: 43.7 x 1e9



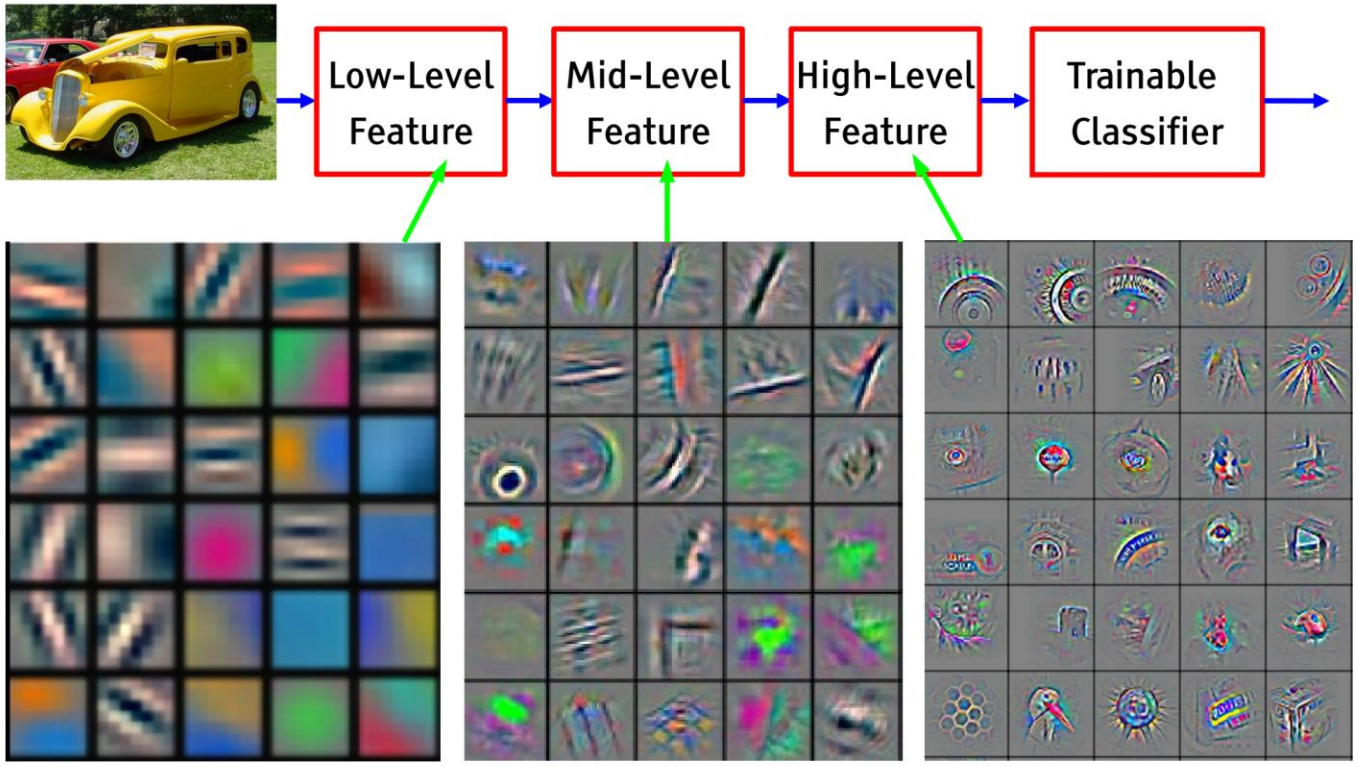
Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." *Advances in neural information processing systems* 25 (2012).

Convolutional Neural Networks (CNNs) Architectures: AlexNet




```
INPUT:     [227x227x3]
CONV1:     [55x55x96]   96 11x11 filters at stride 4, pad 0
MAX POOL1: [27x27x96]      3x3   filters at stride 2
CONV2:     [27x27x256] 256 5x5   filters at stride 1, pad 2
MAX POOL2: [13x13x256]     3x3   filters at stride 2
CONV3:     [13x13x384] 384 3x3   filters at stride 1, pad 1
CONV4:     [13x13x384] 384 3x3   filters at stride 1, pad 1
CONV5:     [13x13x256] 256 3x3   filters at stride 1, pad 1
MAX POOL3: [6x6x256]       3x3   filters at stride 2
FC6:       [4096]      4096 neurons
FC7:       [4096]      4096 neurons
FC8:       [1000]      1000 neurons (softmax logits)
```

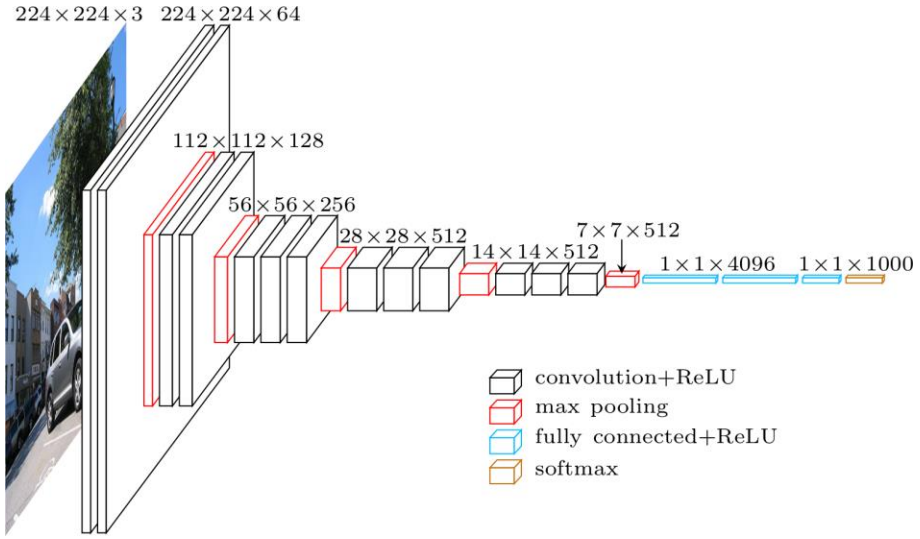
Convolutional Neural Networks (CNNs) Architectures: Hierarchical Representation



Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

Convolutional Neural Networks (CNNs) Architectures: VGG-16

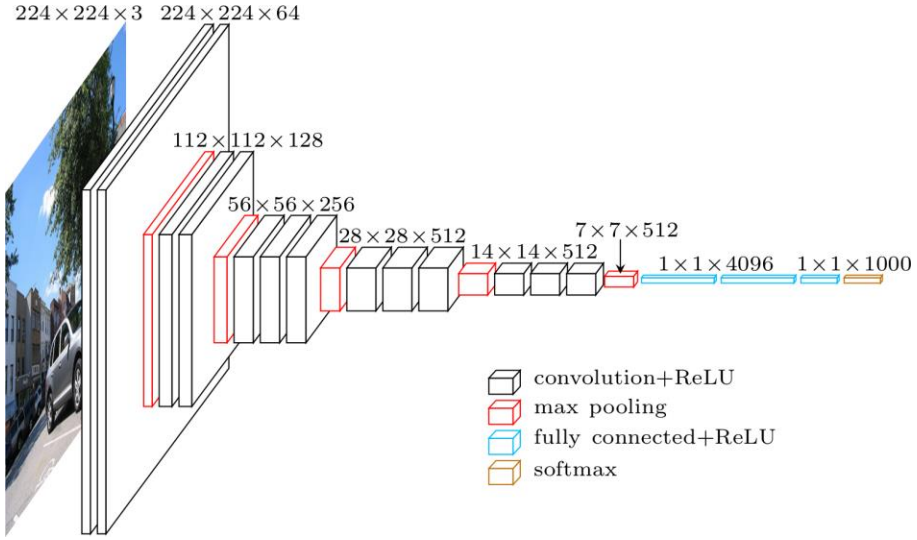
VGG-16 is a deep convolutional neural network architecture that was introduced by the **Visual Geometry Group (VGG)** at the University of Oxford. It was presented in the paper titled "**Very Deep Convolutional Networks for Large-Scale Image Recognition**" by **Karen Simonyan** and **Andrew Zisserman**. **VGG-16** was a runner-up in the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) in **2014**.



Simonyan, Karen, and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." arXiv preprint arXiv:1409.1556 (2014).

Convolutional Neural Networks (CNNs) Architectures: VGG-16

- **Architecture:** VGG-16 is characterized by its deep architecture. It has 16 weight layers, including 13 convolutional layers and 3 fully connected layers.
- **Convolutional Layers:** VGG-16 uses a series of small 3x3 convolutional filters with a stride of 1 and 'same' padding. This allows it to learn complex features by stacking multiple layers of small receptive fields.
- **Max-Pooling:** After every two convolutional layers, max-pooling with a 2x2 window and stride 2 is applied. This helps reduce the spatial dimensions of the feature maps while retaining important information.
- **ReLU Activation:** Like AlexNet, VGG-16 uses the Rectified Linear Unit (ReLU) activation function after each convolutional layer.
- **Fully Connected Layers:** The final layers of the network are fully connected. The architecture ends with two fully connected layers, each with 4,096 units, followed by an output layer with 1,000 units (corresponding to the 1,000 ImageNet classes).



Simonyan, Karen, and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." arXiv preprint arXiv:1409.1556 (2014).

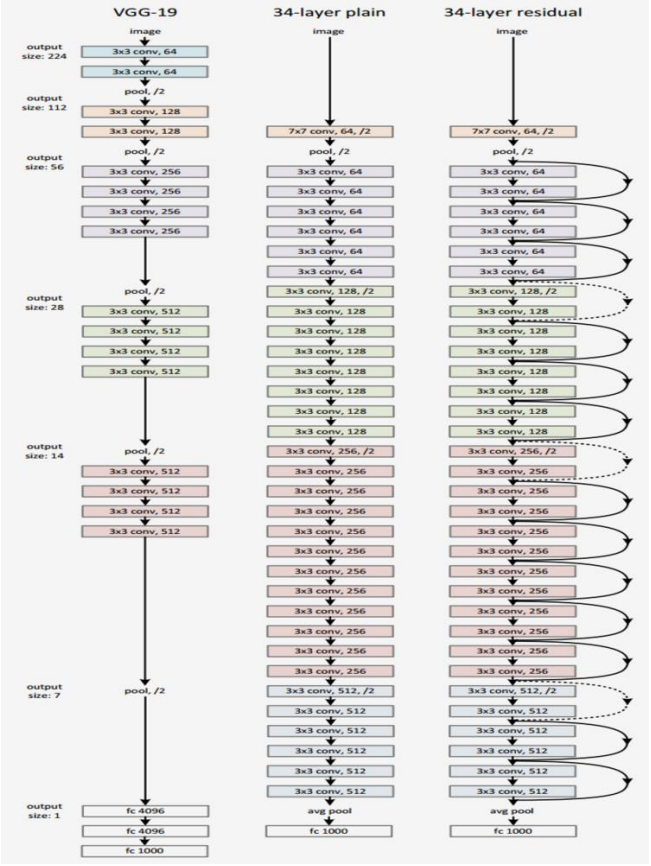
Convolutional Neural Networks (CNNs) Architectures: VGG-16

VGG-16 in Keras

```
model.add(Convolution2D(64, 3, 3, activation='relu',input_shape=(3,224,224)))
model.add(Convolution2D(64, 3, 3, activation='relu'))
model.add(MaxPooling2D((2,2), strides=(2,2)))

model.add(Convolution2D(128, 3, 3, activation='relu'))
model.add(Convolution2D(128, 3, 3, activation='relu'))
model.add(MaxPooling2D((2,2), strides=(2,2)))

model.add(Convolution2D(256, 3, 3, activation='relu'))
model.add(Convolution2D(256, 3, 3, activation='relu'))
model.add(Convolution2D(256, 3, 3, activation='relu'))
model.add(MaxPooling2D((2,2), strides=(2,2)))

model.add(Convolution2D(512, 3, 3, activation='relu'))
model.add(Convolution2D(512, 3, 3, activation='relu'))
model.add(Convolution2D(512, 3, 3, activation='relu'))
model.add(MaxPooling2D((2,2), strides=(2,2)))

model.add(Convolution2D(512, 3, 3, activation='relu'))
model.add(Convolution2D(512, 3, 3, activation='relu'))
model.add(Convolution2D(512, 3, 3, activation='relu'))
model.add(MaxPooling2D((2,2), strides=(2,2)))

model.add(Flatten())
model.add(Dense(4096, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(4096, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(1000, activation='softmax'))
```

Convolutional Neural Networks (CNNs) Architectures: ResNet

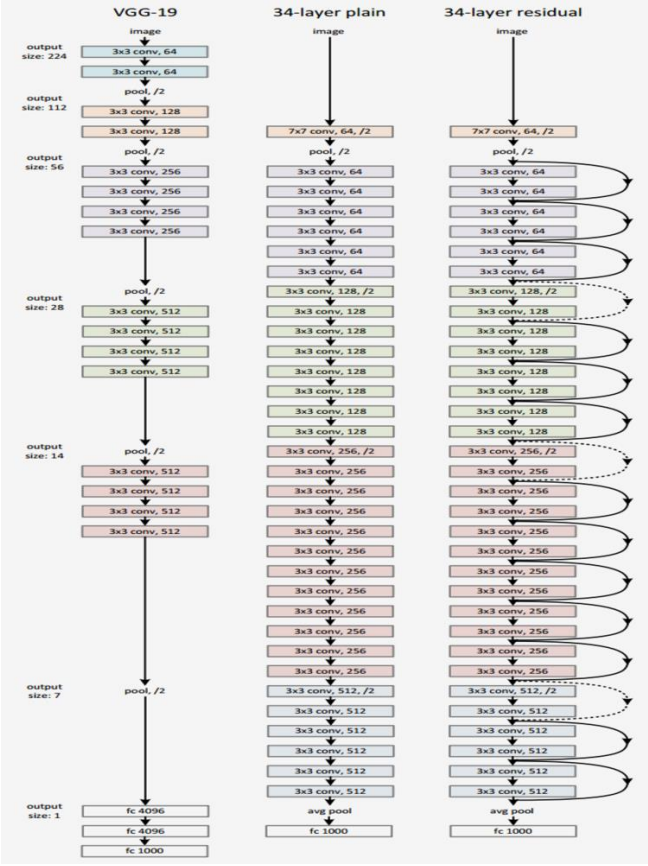
- ResNet, short for Residual Network, is a deep neural network architecture that was introduced in the paper titled "Deep Residual Learning for Image Recognition" by Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, published in 2015. ResNet is known for its ability to train very deep networks effectively, overcoming the problem of vanishing gradients.



He, Kaiming, et al. "Deep residual learning for image recognition." Proceedings of the IEEE conference on computer vision and pattern recognition. 2016.

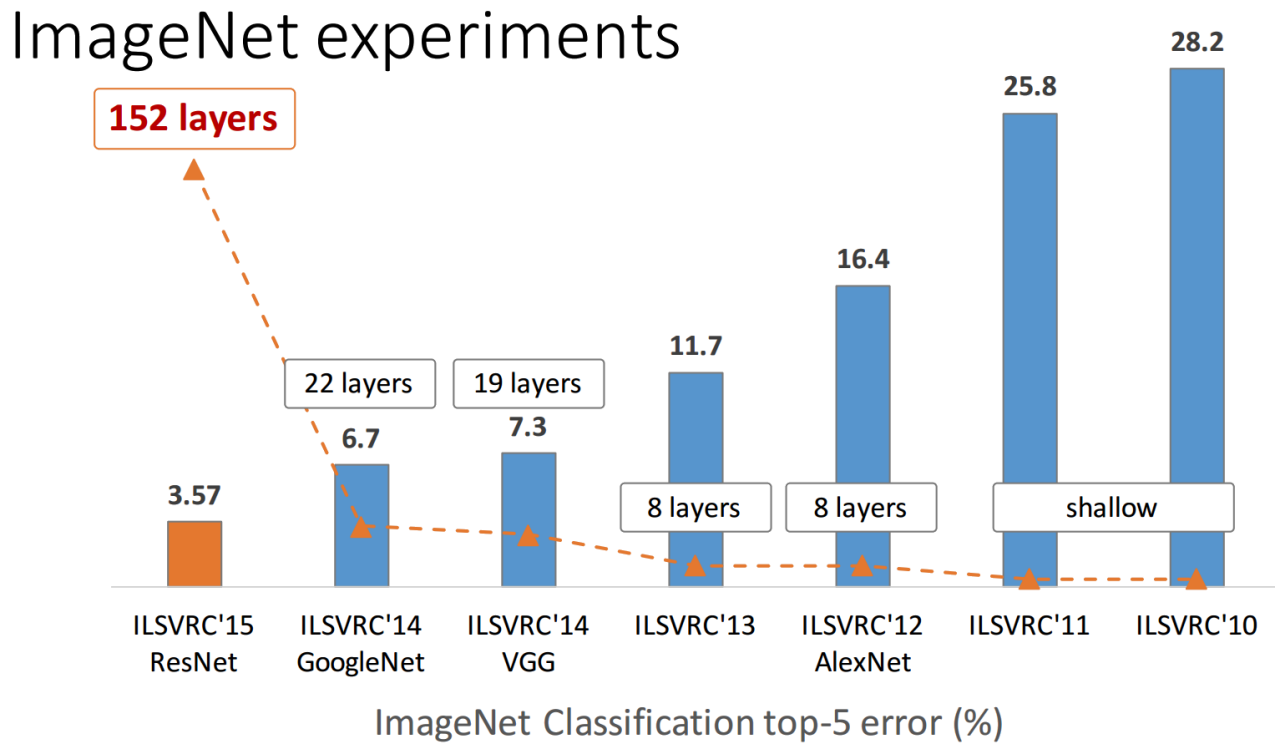
Convolutional Neural Networks (CNNs) Architectures: ResNet

- **ResNet50** Compared to **VGG**:
  - Superior accuracy in all vision tasks
    5.25% top-5 error vs 7.1%
  - Less parameters
    25M vs 138M
  - Computational complexity
    3.8B Flops vs 15.3B Flops
  - Fully Convolutional until the last layer



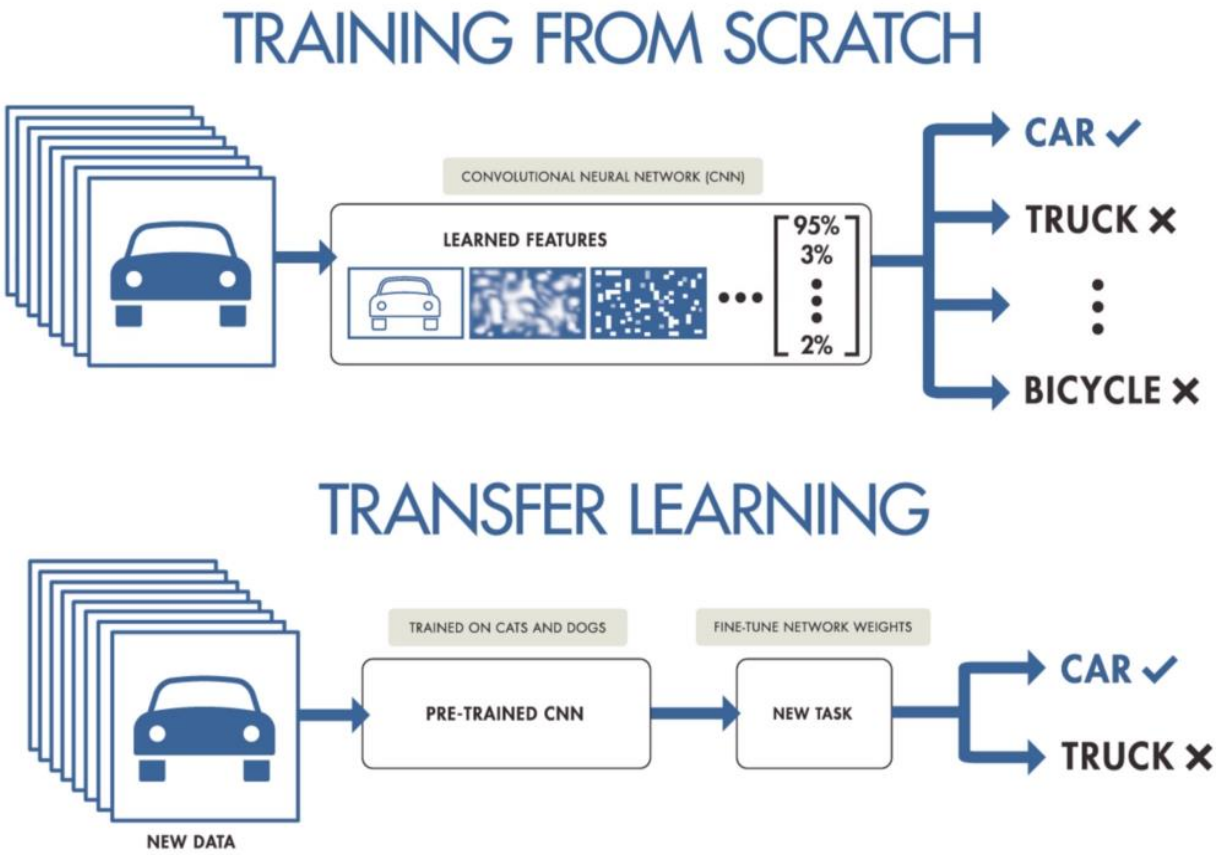
He, Kaiming, et al. "Deep residual learning for image recognition." Proceedings of the IEEE conference on computer vision and pattern recognition. 2016.

Convolutional Neural Networks (CNNs) Architectures: Deeper is Better



from Kaiming He slides "Deep residual learning for image recognition." ICML. 2016.

Transfer Learning and Pre-Trained Models

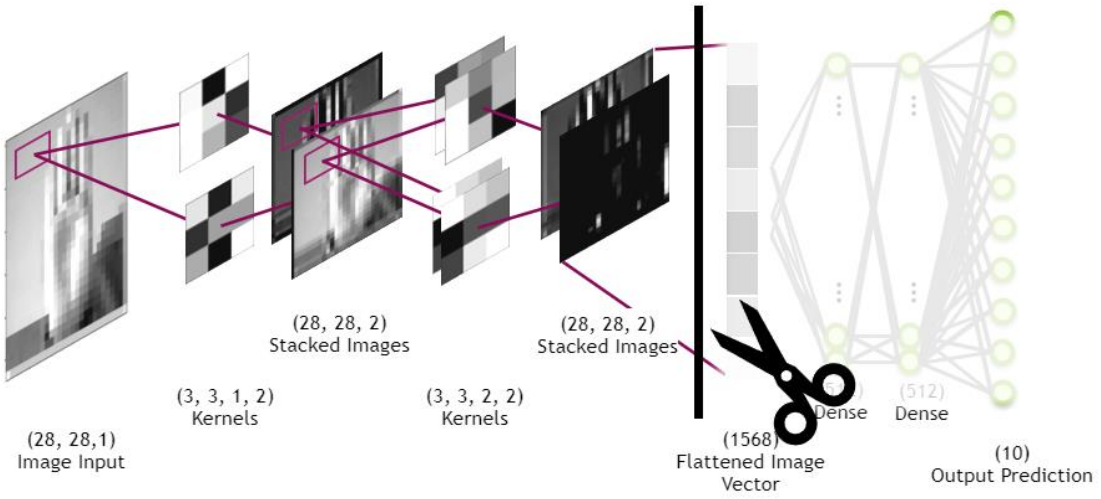


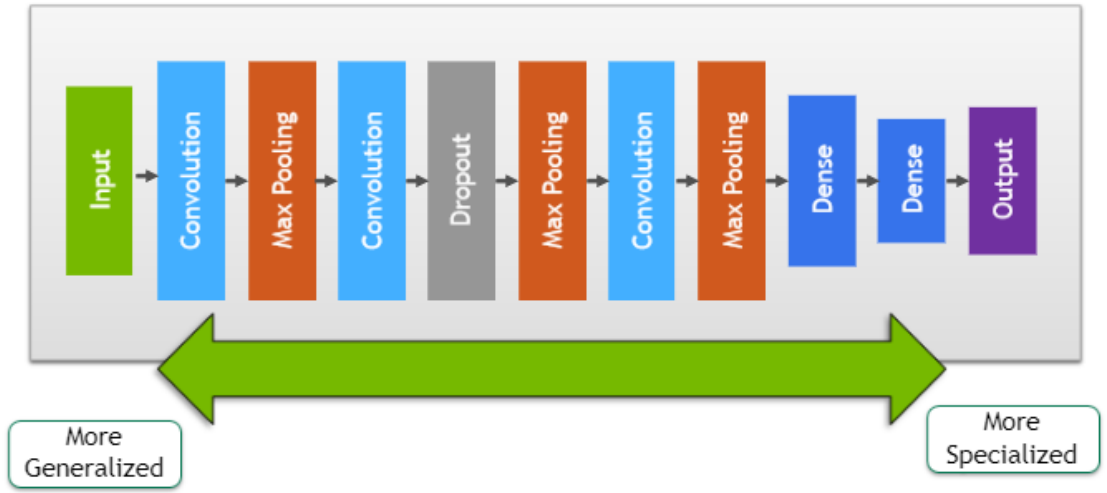
Transfer Learning and Pre-Trained Models

Problem: Training a model on **ImageNet** from scratch takes **days** or **weeks**.
Hint: Many models trained on ImageNet and their weights are publicly available!
Solution: Transfer learning

- Use pre-trained weights, remove last layers to compute representations of images
- Train a classification model from these features on a new classification task
- The network is used as a generic feature extractor
- Better than handcrafted feature extraction on natural images

Transfer Learning and Pre-Trained Models

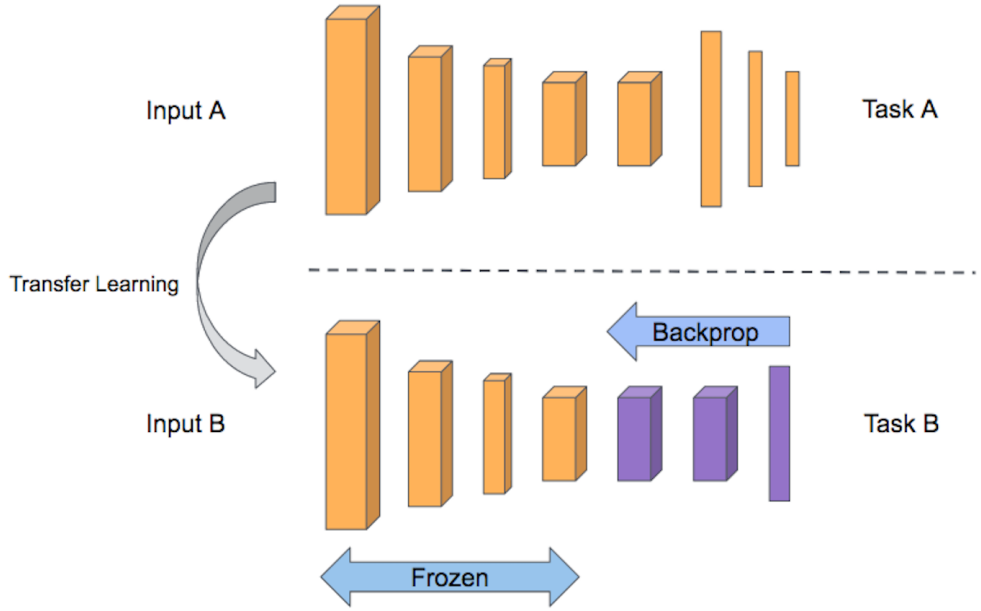




Transfer Learning and Pre-Trained Models

Transfer Learning (TL) from task A to task B can be very useful if:

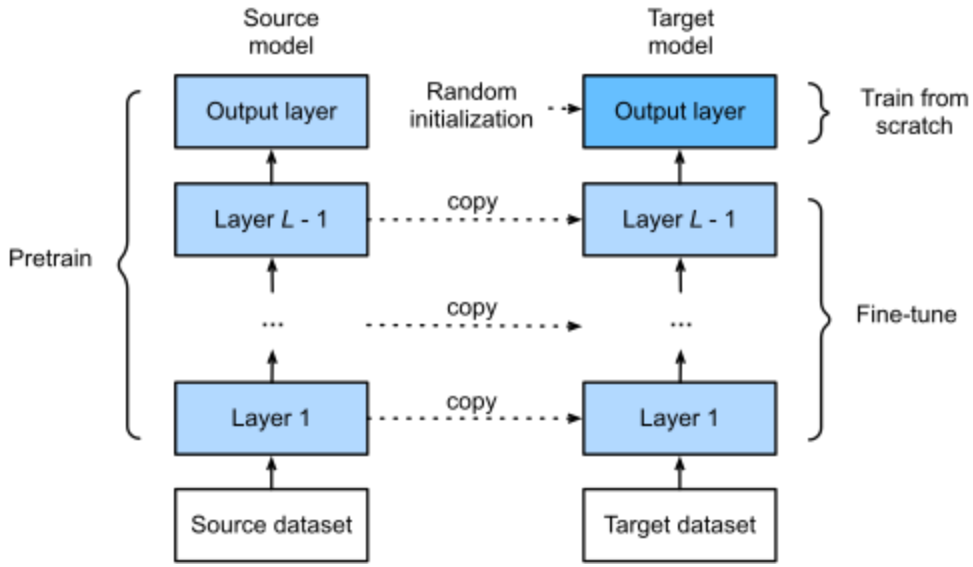
- Task A and B have the same input X.
- There is a lot more data for task A than task B.
- Low level features from task A could be very helpful for learning task B



Transfer Learning and Pre-Trained Models: Fine-Tuning

Fine-tuning:

- Retraining the (some) parameters of the network (given enough data)
- Truncate the last layer(s) of the pre-trained network
- Freeze the remaining layers weights
- Add a (linear) classifier on top and train it for a few epochs
- Then fine-tune the whole network or the few deepest layers
- Use a smaller learning rate when fine tuning



Source: http://d2l.ai/chapter_computer-vision/fine-tuning.html

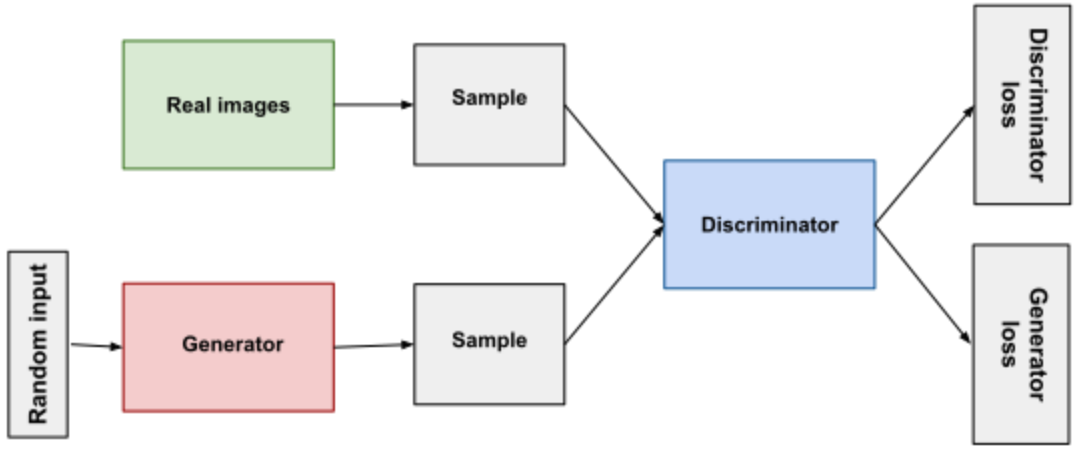
Transfer Learning and Pre-Trained Models: DEMO

Project: Session 3 – Computer Vision and CNN

Generative Adversarial Network (GAN)

Generative Adversarial Networks (GANs) are a class of machine learning models (Generative models) introduced by **Ian Goodfellow** and his colleagues in **2014**. **GANs** are a type of generative model, which means they are designed to generate new data that is similar to some existing dataset.
A GAN consists of two neural networks:

- **Generator**: This network generates data. It takes random noise as input and transforms it into data samples that are intended to be similar to real data. For example, in a GAN that generates images, the generator takes random noise as input and tries to produce images that look like real photographs.
- **Discriminator**: This network tries to distinguish between real data (from the actual dataset) and fake data generated by the generator. It is trained to assign high probabilities to real data and low probabilities to fake data.

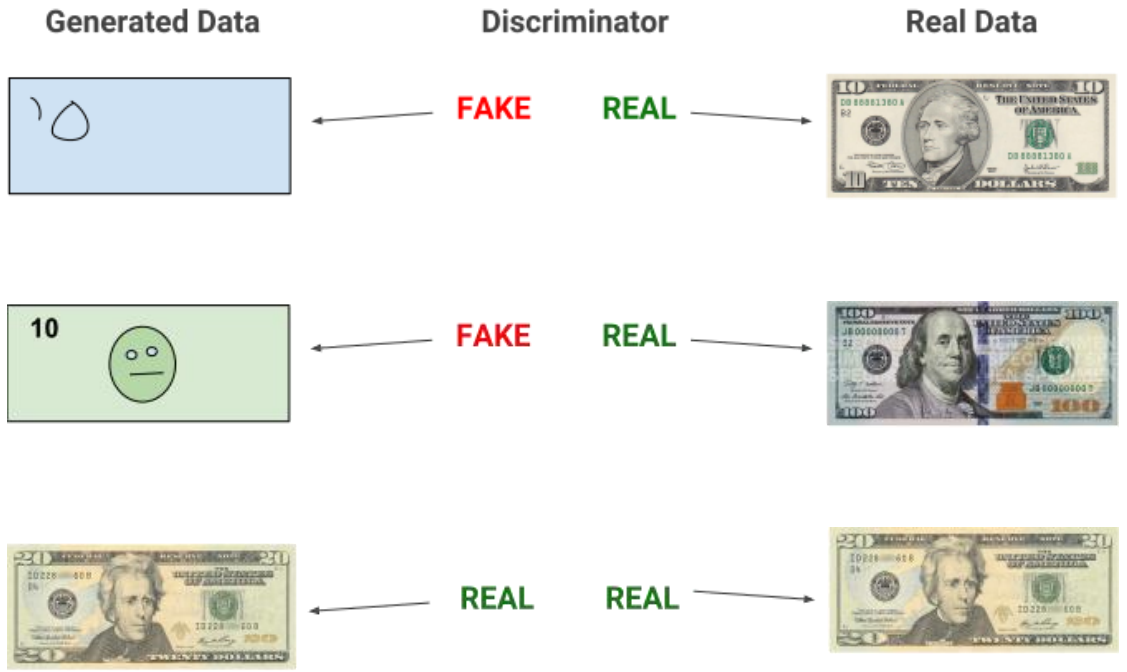


Source: https://developers.google.com/machine-learning/gan/gan_structure

Generative Adversarial Network (GAN): Overview of GAN Structure

A generative adversarial network (**GAN**) has **two parts**:

- The **generator** learns to generate plausible data. The generated instances become negative training examples for the discriminator.
- The **discriminator** learns to distinguish the **generator's** fake data from real data. The **discriminator** penalizes the generator for producing implausible results.



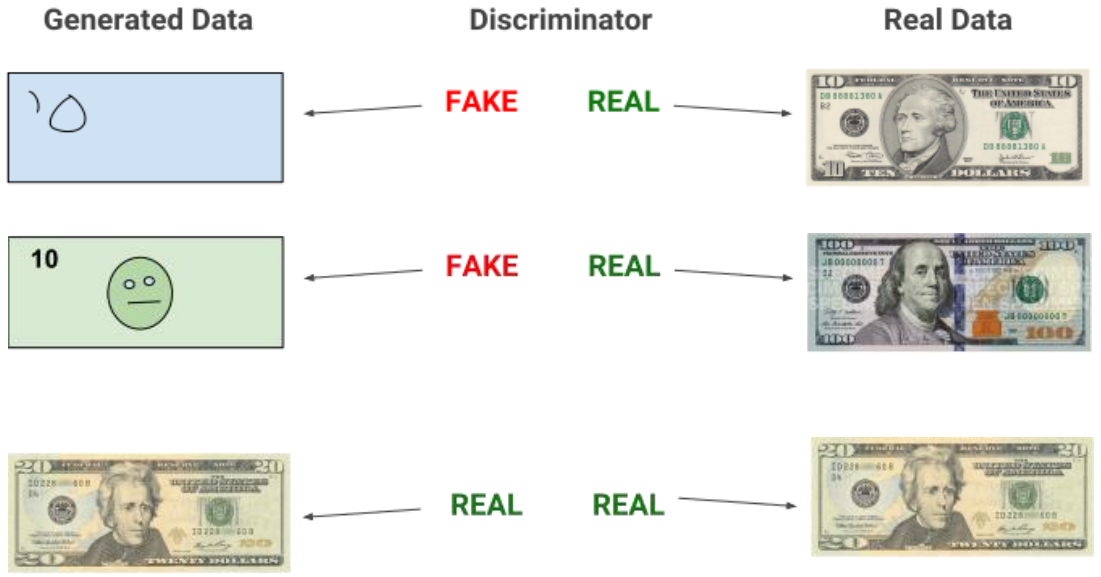
Source: https://developers.google.com/machine-learning/gan/gan_structure

Generative Adversarial Network (GAN): Overview of GAN Structure

The training process of a GAN involves a kind of adversarial game between these two networks:

1. Initially, the generator produces random samples, which are usually of low quality and unlikely to resemble the real data.
2. The discriminator is trained on a mixture of real and fake data, learning to distinguish between them.
3. The generator then tries to produce better fake samples to fool the discriminator.
4. The process continues iteratively, with both networks improving over time. The generator learns to generate more convincing data, while the discriminator becomes better at distinguishing real from fake.

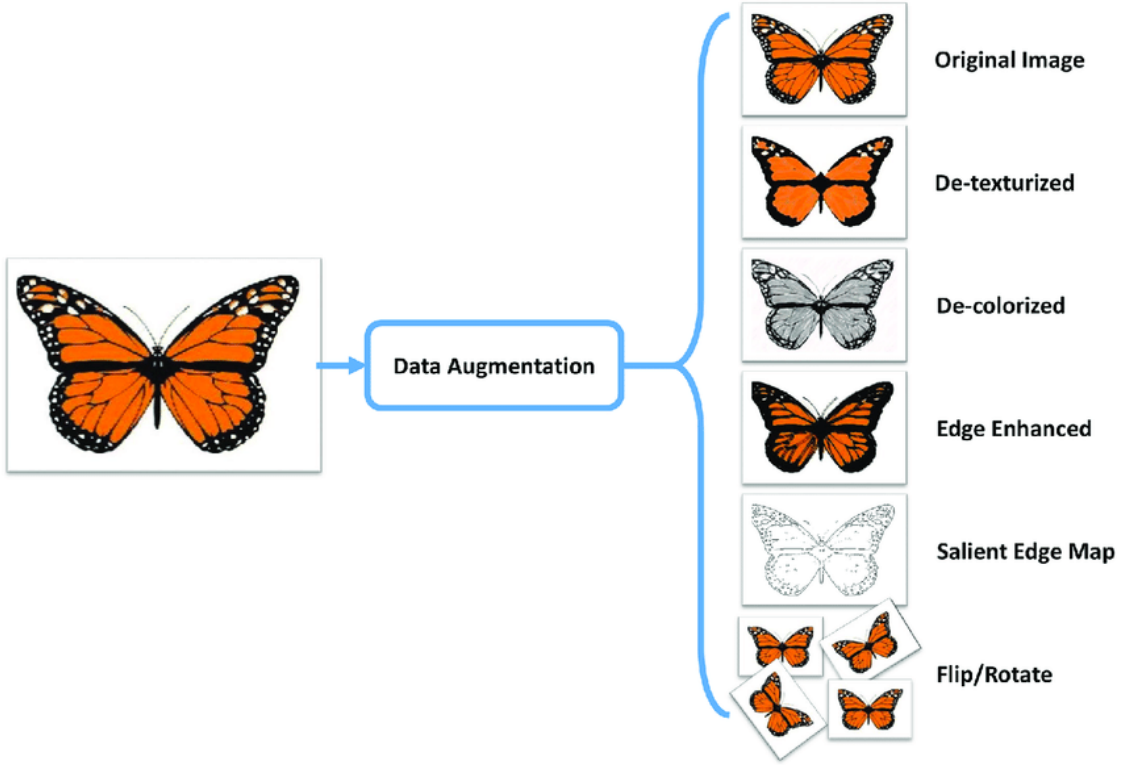
Ideally, this process leads to a point where the generator produces data that is indistinguishable from real data to the discriminator. At this point, the GAN has reached a stable state, and the generator is said to have learned the underlying distribution of the real data.



Source: https://developers.google.com/machine-learning/gan/gan_structure

Data Augmentation for Images

Definition: Data augmentation is a technique of artificially increasing the training set by creating modified copies of a dataset using existing data. It includes making minor changes to the dataset or using deep learning to generate new data points.



Source: https://docs.smartly.ai/

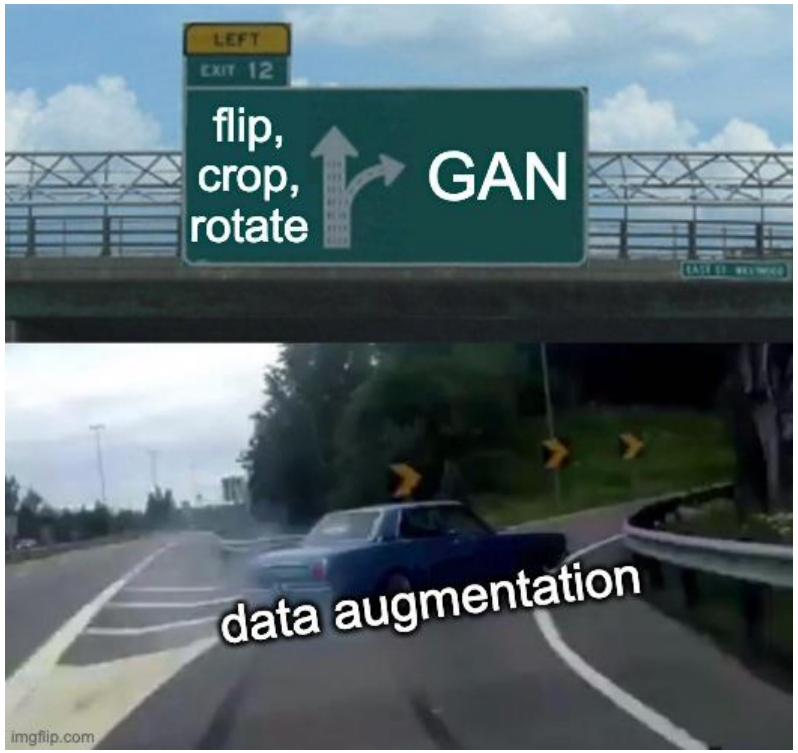
Data Augmentation: Augmented vs. Synthetic Data

Augmented data is driven from original data with some minor changes. In the case of image augmentation, we make geometric and color space transformations (flipping, resizing, cropping, brightness, contrast) to increase the size and diversity of the training set.

Synthetic data is generated artificially without using the original dataset. It often uses DNNs (Deep Neural Networks) and GANs (Generative Adversarial Networks) to generate synthetic data.

Note: the augmentation techniques are not limited to images. You can augment audio, video, text, and other types of data too.

Data Augmentation: Augmented vs. Synthetic Data



Source: https://cs236g.stanford.edu/memes

Data Augmentation: When to Use Data Augmentation?

- To prevent models from overfitting.
- The initial training set is too small.
- To improve the model accuracy.
- To Reduce the operational cost of labeling and cleaning the raw dataset.

Data Augmentation Techniques for Images

- **Geometric transformations:** randomly flip, crop, rotate, stretch, and zoom images. You need to be careful about applying multiple transformations on the same images, as this can reduce model performance.
- **Color space transformations:** randomly change RGB color channels, contrast, and brightness.
- **Kernel filters:** randomly change the sharpness or blurring of the image.
- **Random erasing:** delete some part of the initial image.
- **Mixing images:** blending and mixing multiple images.

Advanced Data Augmentation Techniques for Images

- **Generative adversarial networks (GANs):** used to generate new data points or images. It does not require existing data to generate synthetic data.
- **Neural Style Transfer:** a series of convolutional layers trained to deconstruct images and separate context and style.

Examples of Photorealistic GAN-Generated Faces.Taken from Progressive Growing of GANs for Improved Quality, Stability, and Variation, 2017.

Example of Neural Style Transfer

Data Augmentation: Applications in Healthcare

- **Acquiring** and **labeling** medical imaging datasets is **time-consuming** and **expensive**. You also need a subject matter expert to validate the dataset before performing data analysis. Using geometric and other transformations can help you train robust and accurate machine-learning models.
- For example, in the case of Pneumonia Classification, you can use random cropping, zooming, stretching, and color space transformation to improve the model performance. However, you need to be careful about certain augmentations as they can result in opposite results. For example, random rotation and reflection along the x-axis are not recommended for the X-ray imaging dataset.

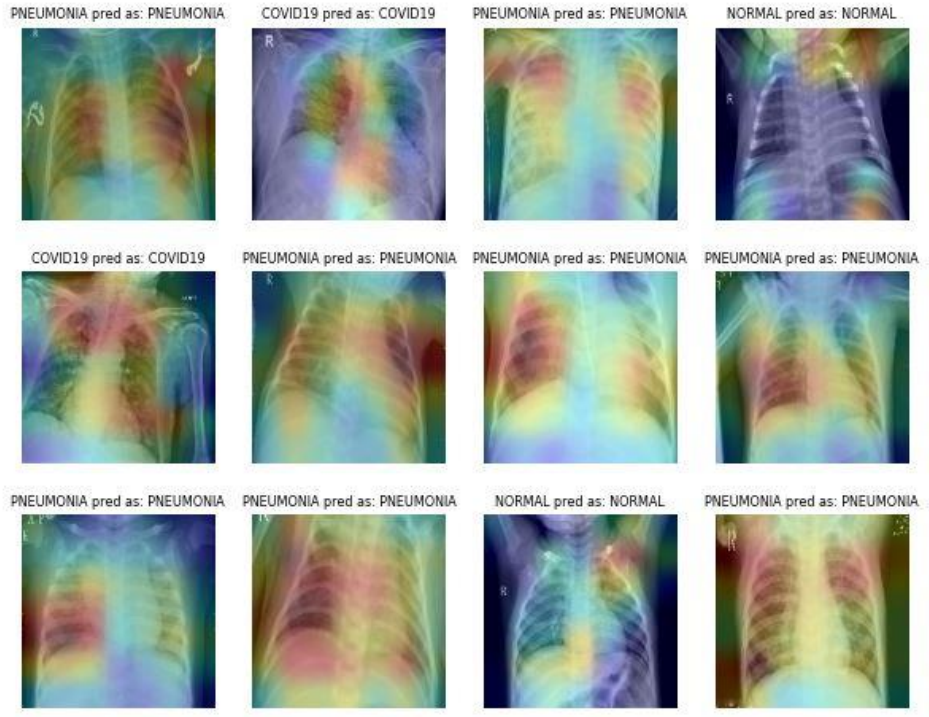


Image from ibrahimsobh.github.io | kaggle-COVID19-Classification

Data Augmentation: Keras Demo

```
from keras.preprocessing.image import ImageDataGenerator

image_gen = ImageDataGenerator(
    rescale=1. / 255,
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    channel_shift_range=9,
    fill_mode='nearest'
)

train_flow = image_gen.flow_from_directory(train_folder)
model.fit_generator(train_flow, train_flow.n)
```

Deep Learning for Computer Vision (CNNs): TO-DO Project

Project: Chest X-ray Image Classification for Disease Diagnosis | Chest X-ray (Covid-19 & Pneumonia)