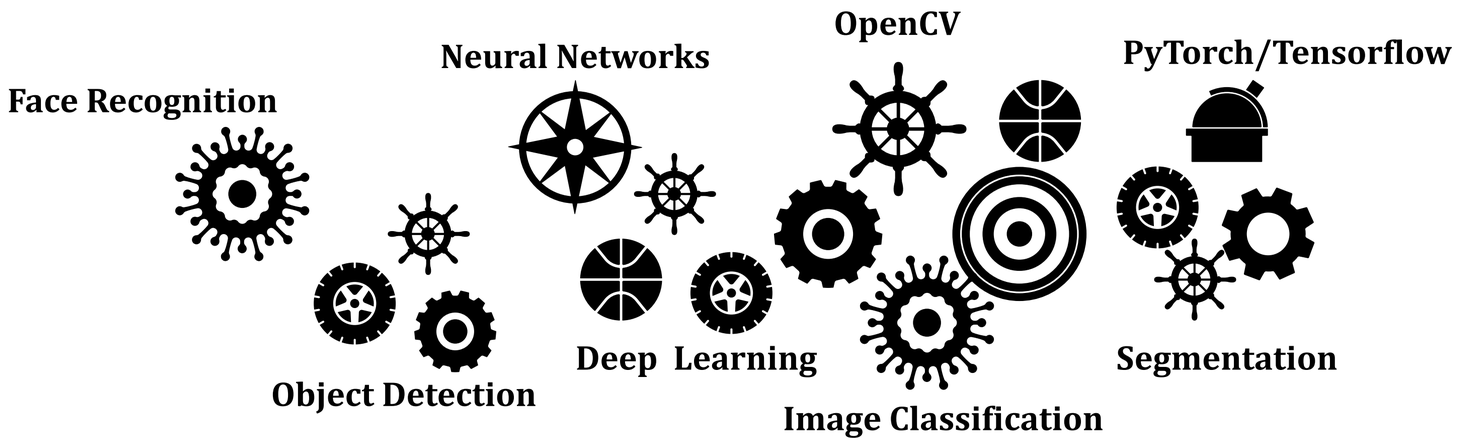
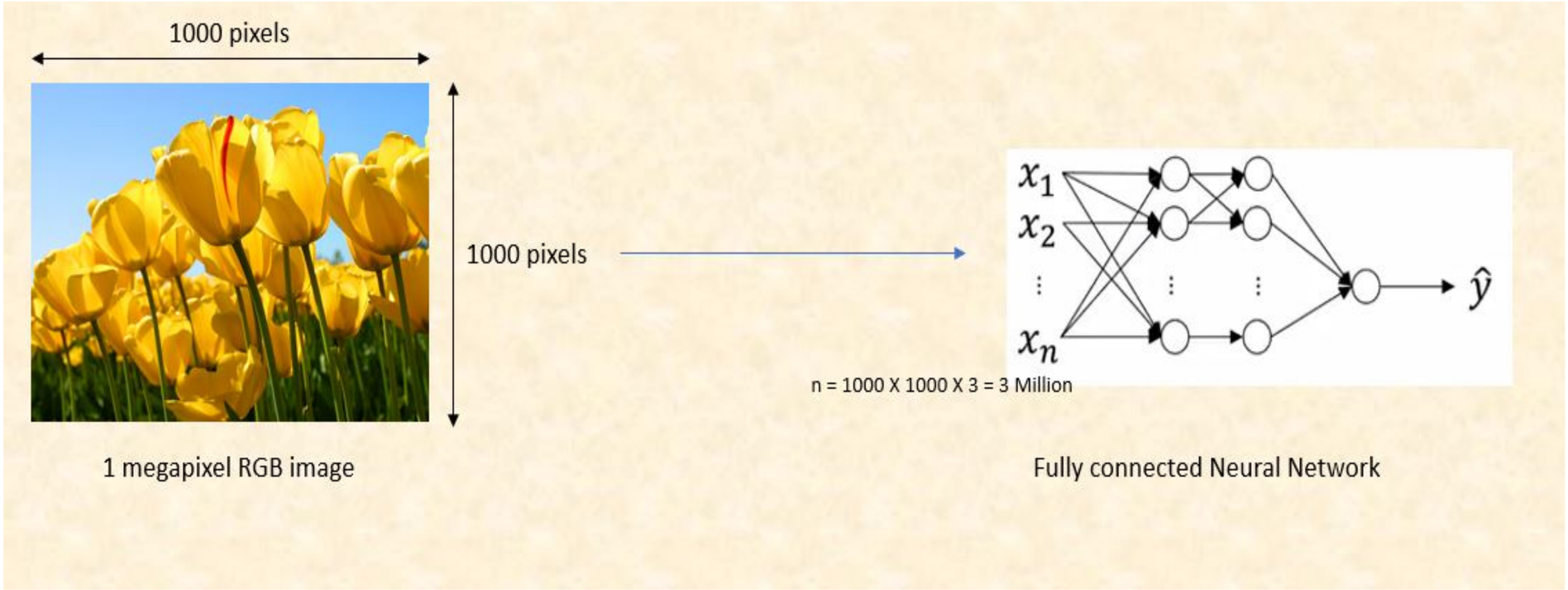
MPSSDEGB
कौशल से रोजगार, समृद्धि अपार

FOUNDATION FOR INNOVATION AND TECHNOLOGY TRANSFER
भारतीय प्रौद्योगिकी संस्थान दिल्ली
Indian Institute of Technology Delhi

Computer Vision



Research gap (ANN to CNN)



Guess?

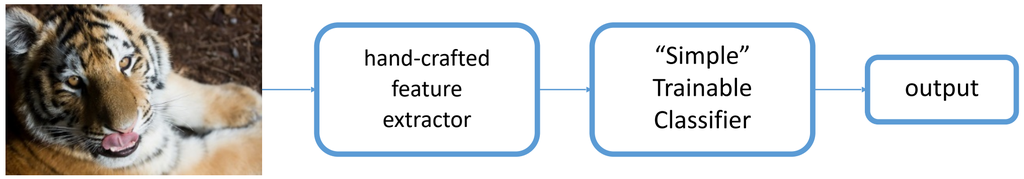
- computationally very expensive
- may lead to overfitting
- little correlation between two closely situated individual pixels.

Convolutional Neural Network - CNN

- CNN
- key concepts
- Back bone networks - vgg and lenet.
- hands-on exercises.
- hyper parameter tuning.

NEED FOR DEEP LEARNING

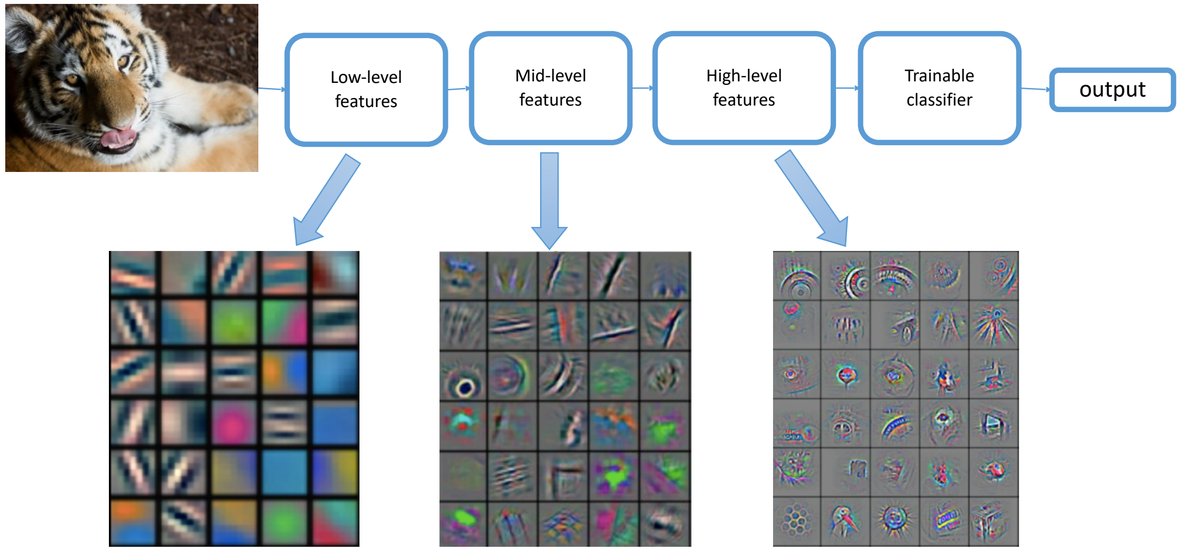
- Traditional pattern recognition models use hand-crafted features and relatively simple trainable classifier.



- This approach has the following limitations:
  - It is very tedious and costly to develop hand-crafted features
  - The hand-crafted features are usually highly dependents on one application, and cannot be transferred easily to other applications

Deep Learning

- Deep learning (a.k.a. representation learning) seeks to learn rich hierarchical representations (i.e. features) automatically through multiple stage of feature learning process.



Convolutional Neural Networks

- Each layer in a CNN applies a different set of filters, typically hundreds or thousands of them, and combines the results, feeding the output into the next layer in the network.

- During training, a CNN automatically learns the values for these filters.

- In the context of image classification, our CNN may learn to:

✓ Detect edges from raw pixel data in the first layer.

✓ Use these edges to detect shapes (i.e., “blobs”) in the second layer.

✓ Use these shapes to detect higher-level features such as facial structures, parts of a car, etc. in the highest layers of the network.

- The last layer in a CNN uses these higher-level features to make predictions regarding the contents of the image.

Convolutional Neural Networks

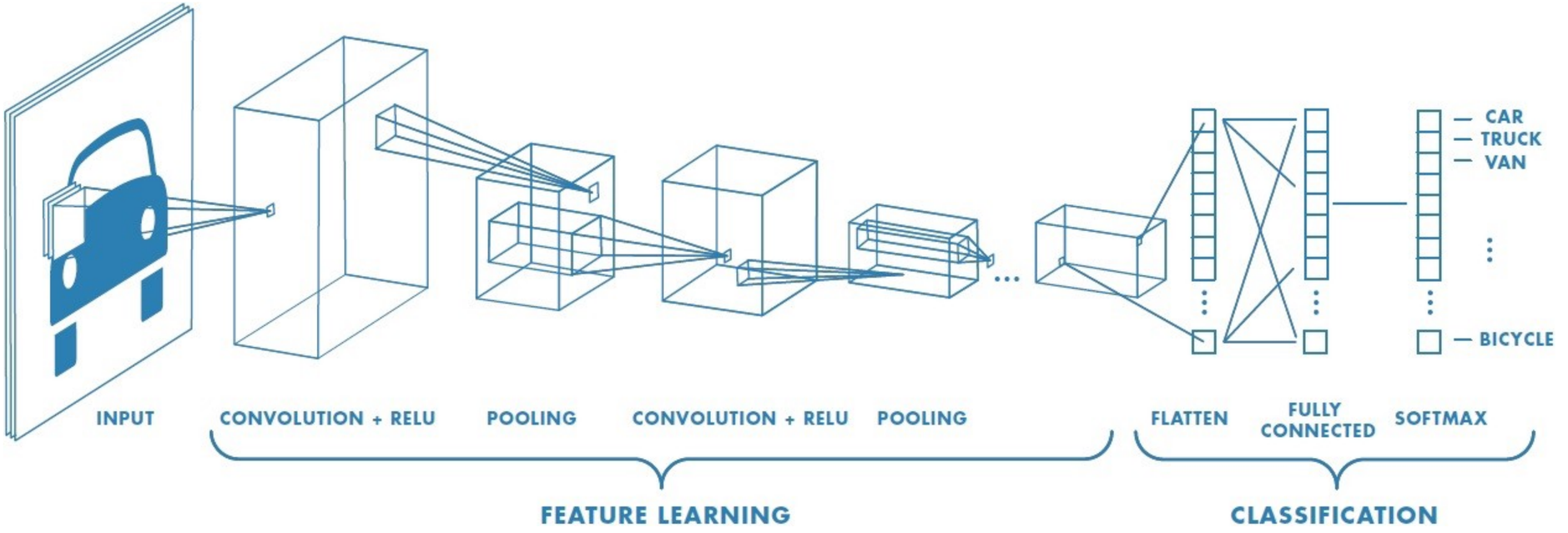
CNN consist of three layers -

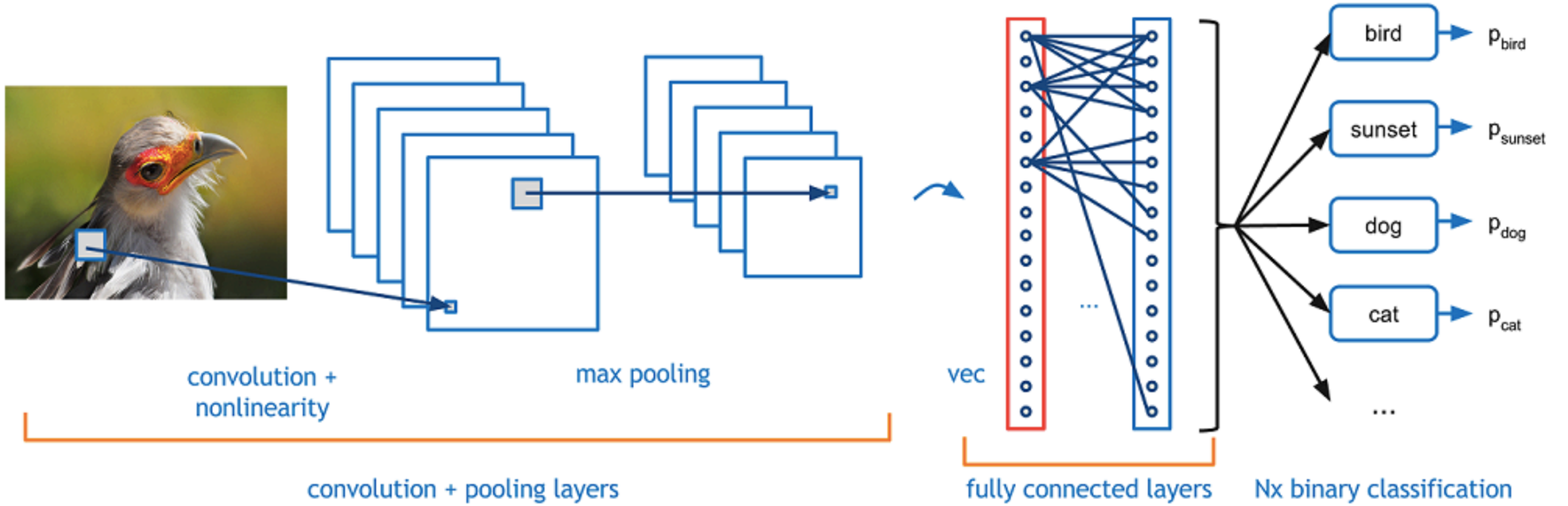
1. Convolution Layer,
2. Pooling Layer, and
3. Fully Connected Layer.

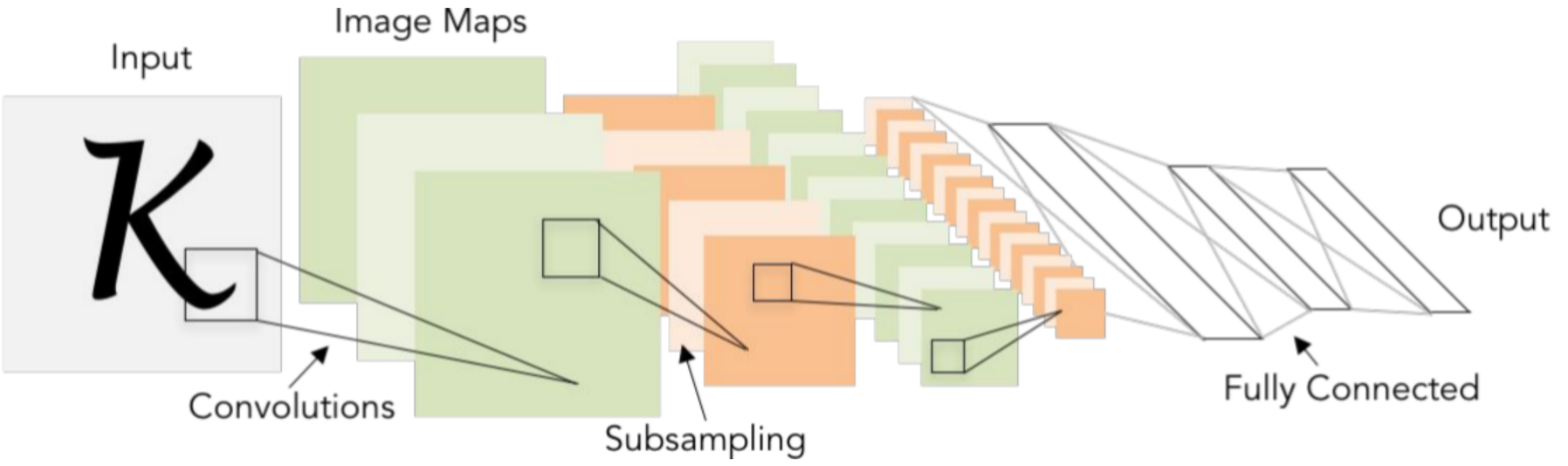
ConvNet is architecture is formed stacking these layers.

- The output can be a softmax layer indicating whether there is a cat or something else. You can also have a sigmoid layer to give you a probability of the image being a cat.

The content is added from different sources






Input
Image Maps
Output
Convolutions
Subsampling
Fully Connected

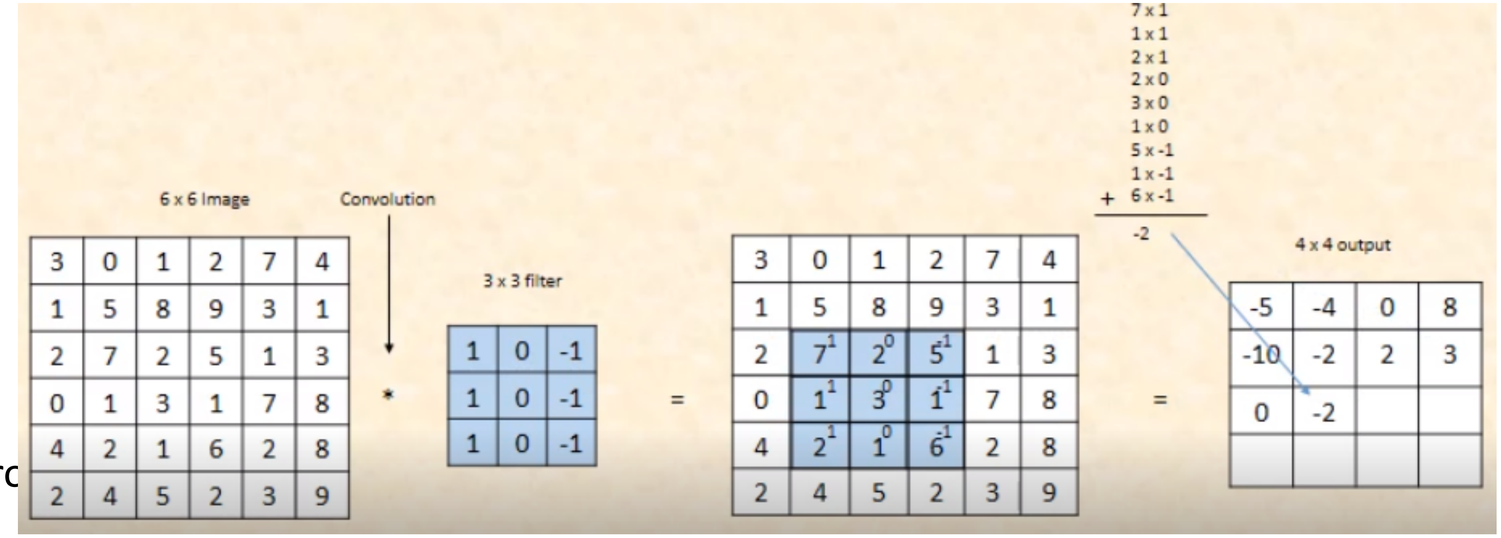
Mathematical Intuition - CNN

Convolution operation -

- convolving a 6 x 6 grayscale image with a 3 x 3 matrix called filter or kernel to produce a 4 x 4 matrix.

- dot product between the filter and the first 9 elements of the image matrix and fill the output matrix.

- pro

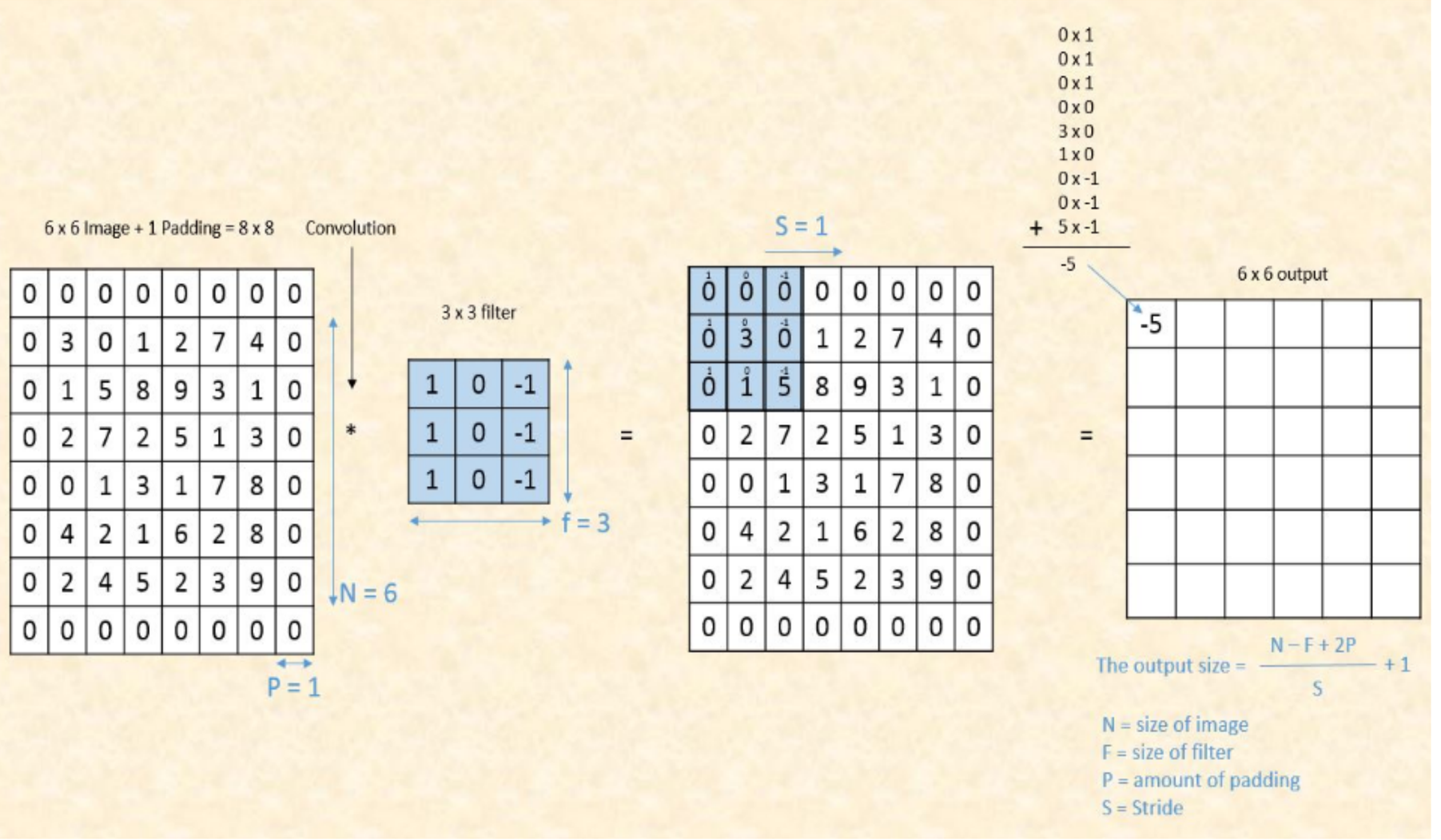


Challenges with convolution operation:

- Output may shrink.
- Data loss from image corners.

Solution - Padding (size of the output image is equal to input image size.)

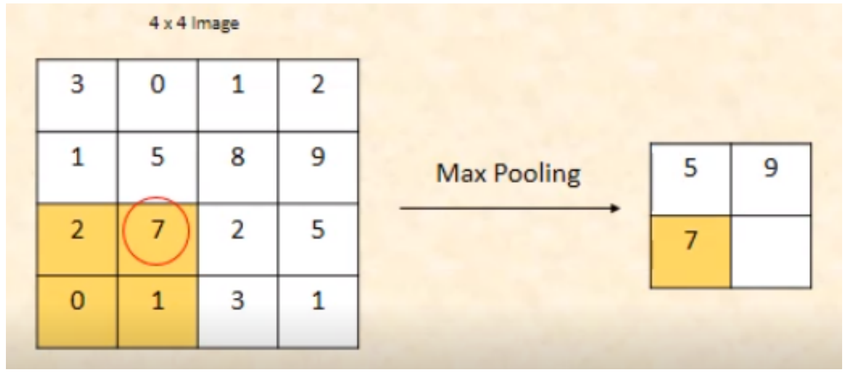
The content is added from different sources

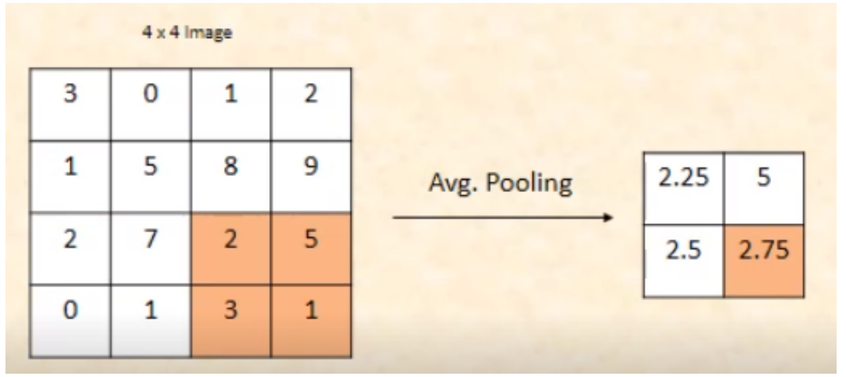


Pooling Layer:

- Reduce number of parameters.

- In pooling layer, we have two hyperparameters filter size and stride which are fixed only once.

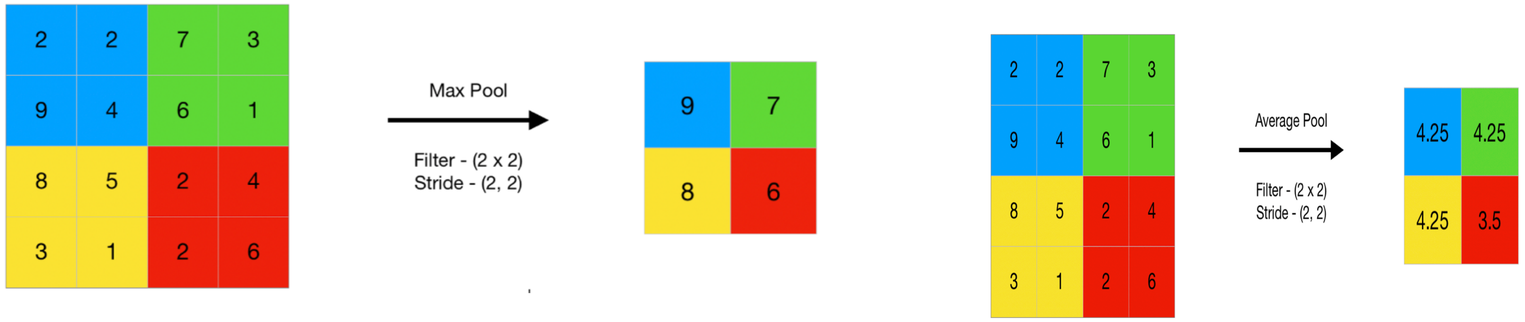




- Pooling layers are used to reduce the dimensions of the feature maps. Reduces the number of parameters to learn and the amount of computation performed in the network.

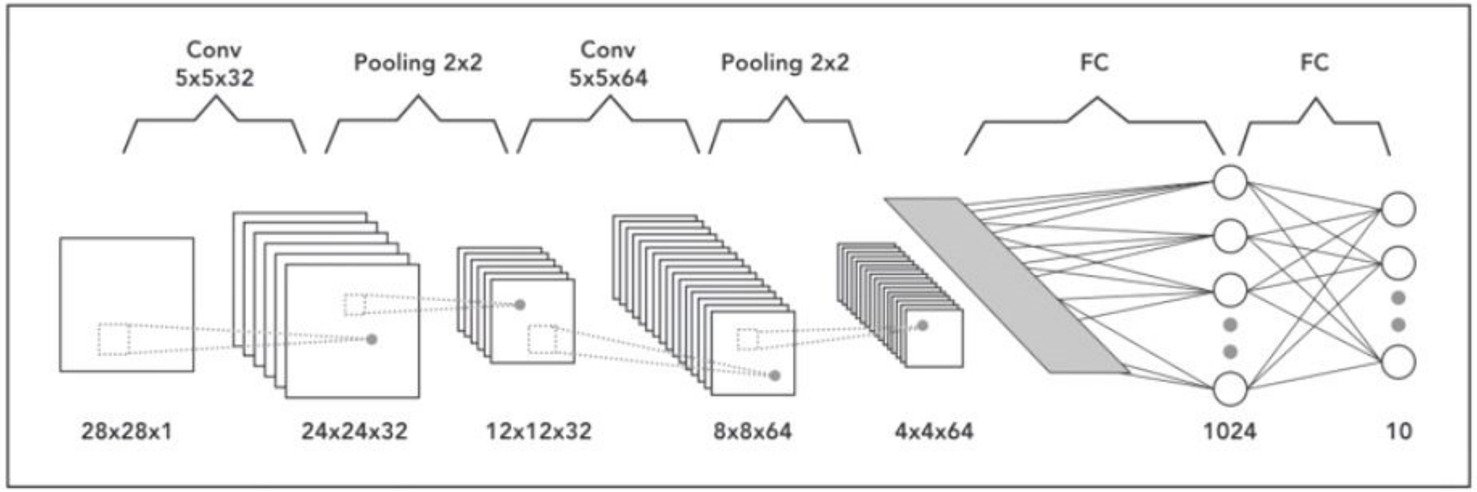
- The pooling layer summarises the features present in a region of the feature map generated by a convolution layer. So, further operations are performed on summarised features instead of precisely positioned features generated by the convolution layer. This makes the model more robust to variations in the position of the features in the input image.

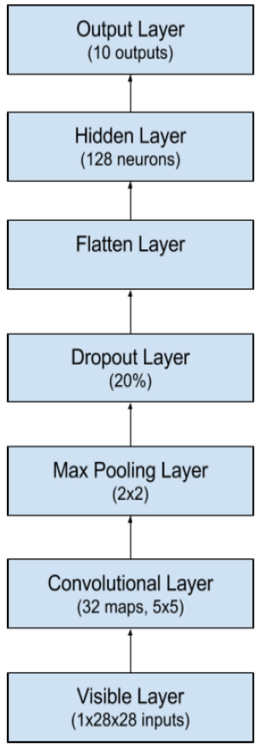
- Average pooling computes the average of the elements present in the region of feature map covered by the filter. Thus, while max pooling gives the most prominent feature in a particular patch of the feature map, average pooling gives the average of features present in a patch.



Architecture:

The content is added from different sources

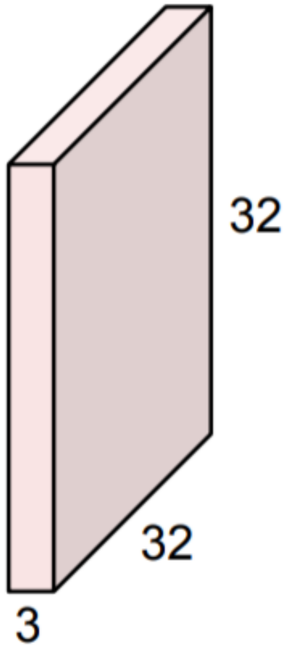




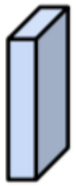
CNN Architecture for MNIST

Convolve the filter with the image
i.e. “slide over the image spatially,
computing dot products”

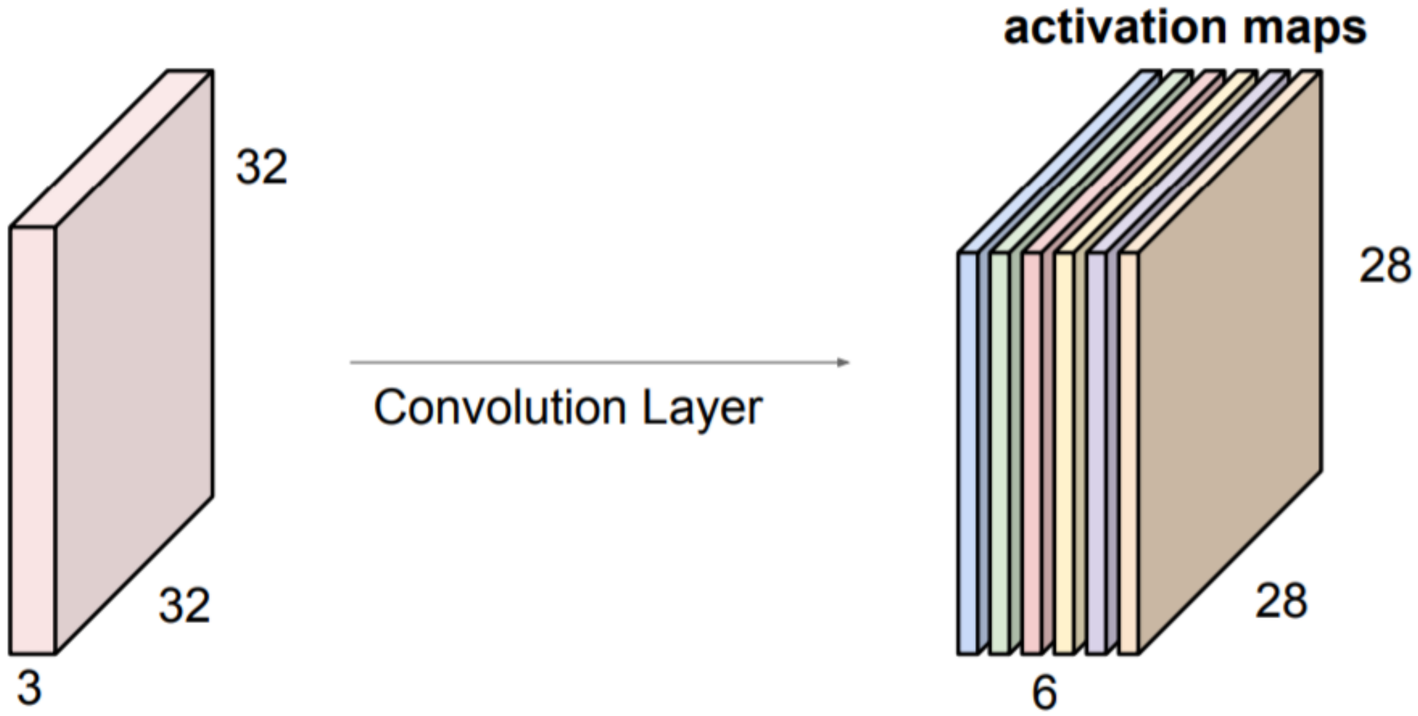
32x32x3 image



5x5x3 filter



Convolve the filter with the image
i.e. “slide over the image spatially,
computing dot products”



Ex: we have 6 - > 5x5 filters, we'll get 6 separate activation maps. "new image" of size 28x28x6!

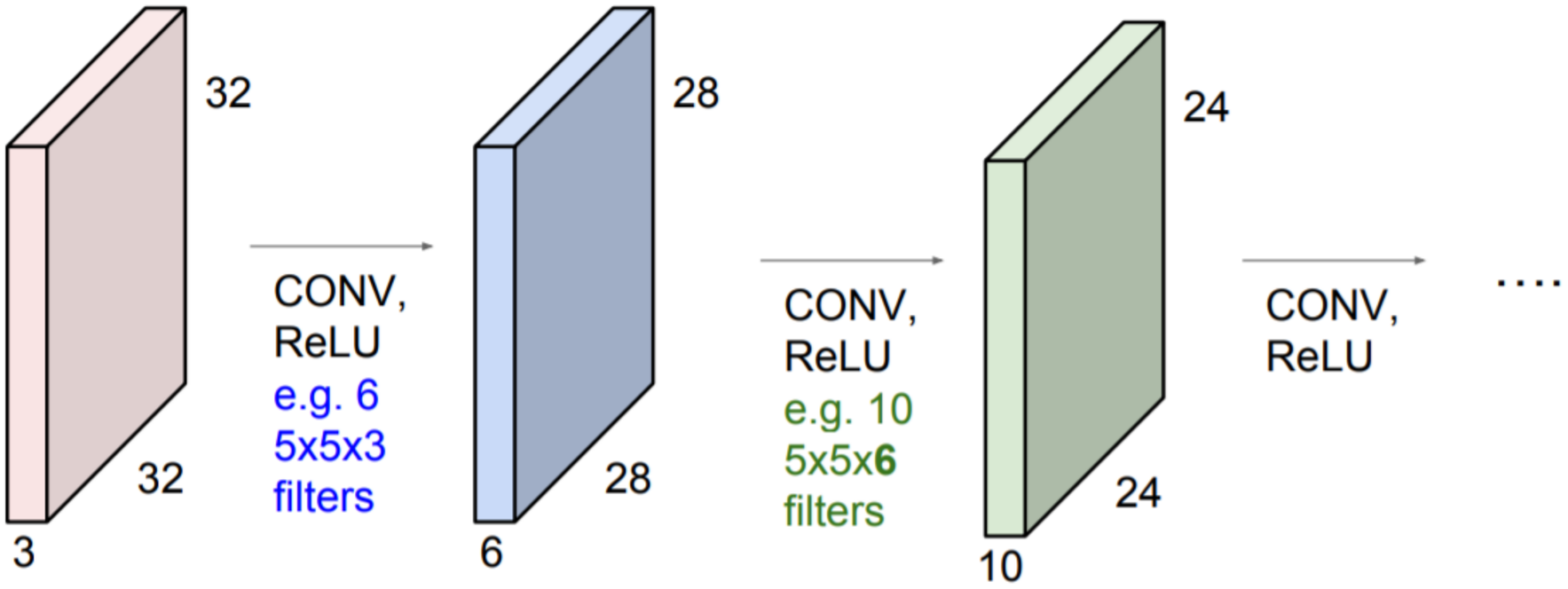

32
32
3
CONV,
ReLU
e.g. 6
5x5x3
filters
28
28
6
CONV,
ReLU
e.g. 10
5x5x6
filters
24
24
10
CONV,
ReLU
....

Image Input Data

•Each image has the same size of 32 pixels wide and 32 pixels high, and pixel values are between 0 and 255, e.g. a matrix of 32 32 1 or 1,024 pixel values.

Convolutional Layer

•We dene a convolutional layer with 10 filters and a receptive field 5 pixels wide and 5 pixels high and a stride length of 1. Because each filter can only get input from (i.e. see) 5 *5 (25) pixels at a time, we can calculate that each will require 25 + 1 input weights (plus 1 for the bias input).

•Dragging the 5 *5 receptive field across the input image data with a stride width of 1will result in a feature map of 28 *28 output values or 784 distinct activations per image.

•We have 10 filters, so that is 10 different 28 *28 feature maps or 7,840 outputs that will be created for one image.

Pool Layer

- We dene a pooling layer with a receptive field with a width of 2 inputs and a height of 2 inputs. We also use a stride of 2 to ensure that there is no overlap.
- This results in feature maps that are one half the size of the input feature maps. From 10 different 28 *28 feature maps as input to 10 different 14*14 feature maps as output.

Fully Connected Layer

- Finally, we can flatten out the square feature maps into a traditional at fully connected layer.
- We can dene the fully connected layer with 200 hidden neurons, each with 10 14 14 input connections, or 1,960 + 1 weights per neuron.
- That is a total of 392,200 connections and weights to learn in this layer.

Summary of CNN

- CNN helps in analyzing the imaginary.
- Q -> What human see vs what system see?
- Size of image reducing with increase in stride value.
- Padding the input image with zero - retains depth.
- Kernal/filter that extracts useful info (edges, color, ...)
- Pooling helps in reducing the spatial size of the image.
- Output volume is controled by 3 parameters (no. of filters, stride, zero padding) - ([W-F+2P]/S) + 1.
- Output layer in CNN is a FC layer.

CNN: INPUT => CONV => RELU => FC => SOFTMAX

CNN that accepts an input, applies a convolution layer, then an activation layer, then a fully-connected layer, and, finally, a softmax classifier to obtain the output classification probabilities.

Quiz

Ex:

Q1. Consider input image of 32X32X3 and we have 10 -> 5x5 filters with stride 1, pad 2.

What is output volume?

Q2 . Number of parameters in this layer?

Key

- (32+2*2-5)/1+1 = 32 spatially, so 32x32x10.

- each filter has 5*5*3 + 1 = 76 params (+1 for bias) => 76*10 = 760

Datasets to work with CNN

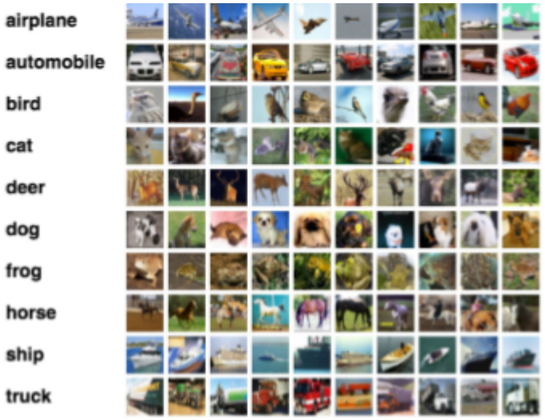
MNIST:

- classify the handwritten digits 0–9.
- 60,000 training images and 10,000 testing images.
- 28×28 grayscale.

Kaggle Dogs vs. Cats:

- 3-class animals dataset consisting of 1,000 images per dog, cat, and panda class respectively for a total of 3,000 images.

CIFAR-10:



- CIFAR-10 consists of 60,000 32×32×3 (RGB) images resulting in a feature vector dimensionality of 3072.
- standard benchmark dataset for image classification
- airplanes, automobiles, birds, cats, deer, dogs, frogs, horses, ships, and trucks.

SMILES:

13,165 grayscale images in the dataset, with each image having a size of 64×64.

CALTECH-101:

- dataset of 8,677 images includes 101 categories.
- popular benchmark dataset for object detection.

ImageNet Large Scale Visual Recognition Challenge (ILSVRC):

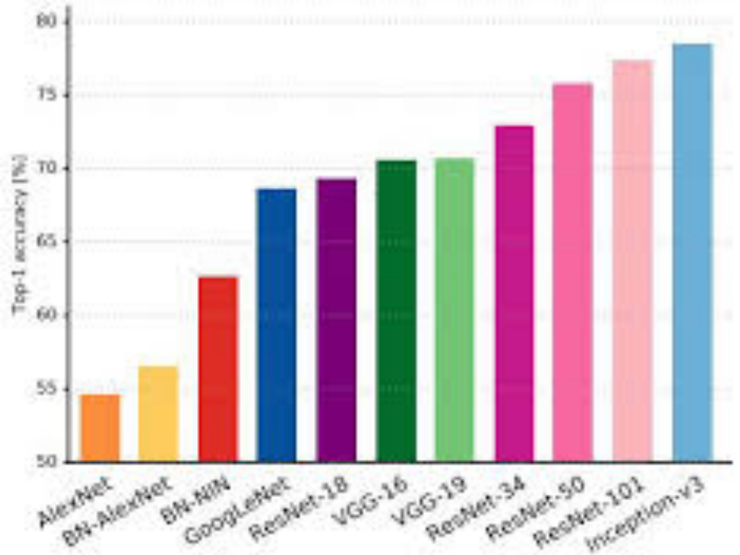
- 1,000 separate categories using approximately 1.2 million images for training, 50,000 for validation, and 100,000 for testing

pascal voc 2012 dataset:'

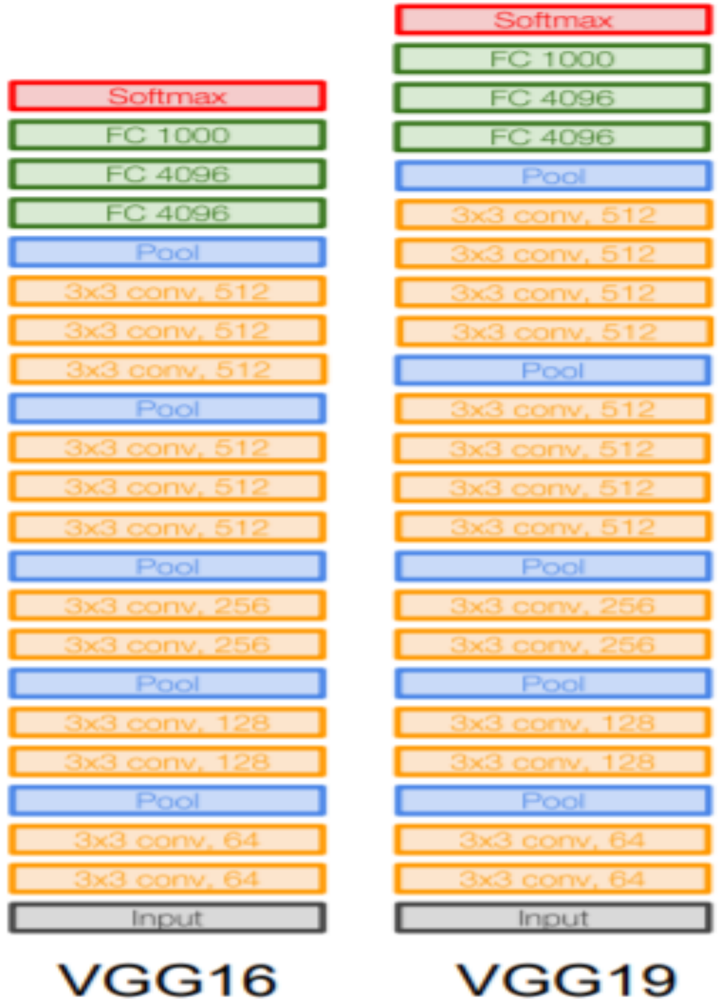
- popular dataset for building and evaluating algorithms for image classification, object detection, and segmentation.

Backbone Architectures

- VGG16
- VGG19
- Lenet
- Inception net
- Resnet
- Alexnet
- Googlenet



VGG Architecture



Achieved excellent results on the ILSVRC-2014 (ImageNet competition).

Small filters, Deeper networks

8 layers (AlexNet)
-> 16 - 19 layers (VGG16Net)

Only 3x3 CONV stride 1, pad 1
and 2x2 MAX POOL stride 2

11.7% top 5 error in ILSVRC'13
(ZFNet)
-> 7.3% top 5 error in ILSVRC'14

During training, the input to our ConvNets is a fixed-size 224 × 224 RGB image. The only pre-processing we do is subtracting the mean RGB value, computed on the training set, from each pixel. The image is passed through a stack of convolutional (conv.) layers, where we use filters with a very small receptive field: 3 × 3 (which is the smallest size to capture the notion of left/right, up/down, center). In one of the configurations we also utilise 1 × 1 convolution filters, which can be seen as a linear transformation of the input channels (followed by non-linearity). The convolution stride is fixed to 1 pixel; the spatial padding of conv. layer input is such that the spatial resolution is preserved after convolution, i.e. the padding is 1 pixel for 3 × 3 conv. layers. Spatial pooling is carried out by five max-pooling layers, which follow some of the conv. layers (not all the conv. layers are followed by max-pooling). Max-pooling is performed over a 2 × 2 pixel window, with stride 2.

A stack of convolutional layers (which has a different depth in different architectures) is followed by three Fully-Connected (FC) layers: the first two have 4096 channels each, the third performs 1000-way ILSVRC classification and thus contains 1000 channels (one for each class). The final layer is the soft-max layer. The configuration of the fully connected layers is the same in all networks.

All hidden layers are equipped with the rectification (ReLU (Krizhevsky et al., 2012)) non-linearity. We note that none of our networks (except for one) contain Local Response Normalisation (LRN) normalisation (Krizhevsky et al., 2012): as will be shown in Sect. 4, such normalisation does not improve the performance on the ILSVRC dataset, but leads to increased memory consumption and computation time. Where applicable, the parameters for the LRN layer are those of (Krizhevsky et al., 2012).

- **input size: 224 x 224;**
- **the receptive field size is 3 x 3;**
- **the convolution stride is 1 pixel;**
- **the padding is 1 (for receptive field of 3 x 3) so we keep the same spatial resolution;**
- **the max pooling is 2 x 2 with stride of 2 pixels;**
- **there are two fully connected layers with 4096 units each;**
- **the last layer is a softmax classification layer with 1000 units (representing the 1000 ImageNet classes);**
- **the activation function is the ReLU**
- **We can now calculate the number of learnable parameters**

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 224, 224, 64)	1792
conv2d_2 (Conv2D)	(None, 224, 224, 64)	36928
max_pooling2d_1 (MaxPooling2	(None, 112, 112, 64)	0
conv2d_3 (Conv2D)	(None, 112, 112, 128)	73856
conv2d_4 (Conv2D)	(None, 112, 112, 128)	147584
max_pooling2d_2 (MaxPooling2	(None, 56, 56, 128)	0
conv2d_5 (Conv2D)	(None, 56, 56, 256)	295168
conv2d_6 (Conv2D)	(None, 56, 56, 256)	590080
conv2d_7 (Conv2D)	(None, 56, 56, 256)	590080
max_pooling2d_3 (MaxPooling2	(None, 28, 28, 256)	0
conv2d_8 (Conv2D)	(None, 28, 28, 512)	1180160
conv2d_9 (Conv2D)	(None, 28, 28, 512)	2359808
conv2d_10 (Conv2D)	(None, 28, 28, 512)	2359808
max_pooling2d_4 (MaxPooling2	(None, 14, 14, 512)	0
conv2d_11 (Conv2D)	(None, 14, 14, 512)	2359808
conv2d_12 (Conv2D)	(None, 14, 14, 512)	2359808
conv2d_13 (Conv2D)	(None, 14, 14, 512)	2359808
max_pooling2d_5 (MaxPooling2	(None, 7, 7, 512)	0
flatten_1 (Flatten)	(None, 25088)	0
dense_1 (Dense)	(None, 4096)	102764544
dense_2 (Dense)	(None, 4096)	16781312
dense_3 (Dense)	(None, 1000)	4097000

Total params: 138,357,544
Trainable params: 138,357,544
Non-trainable params: 0

See Network behaviour

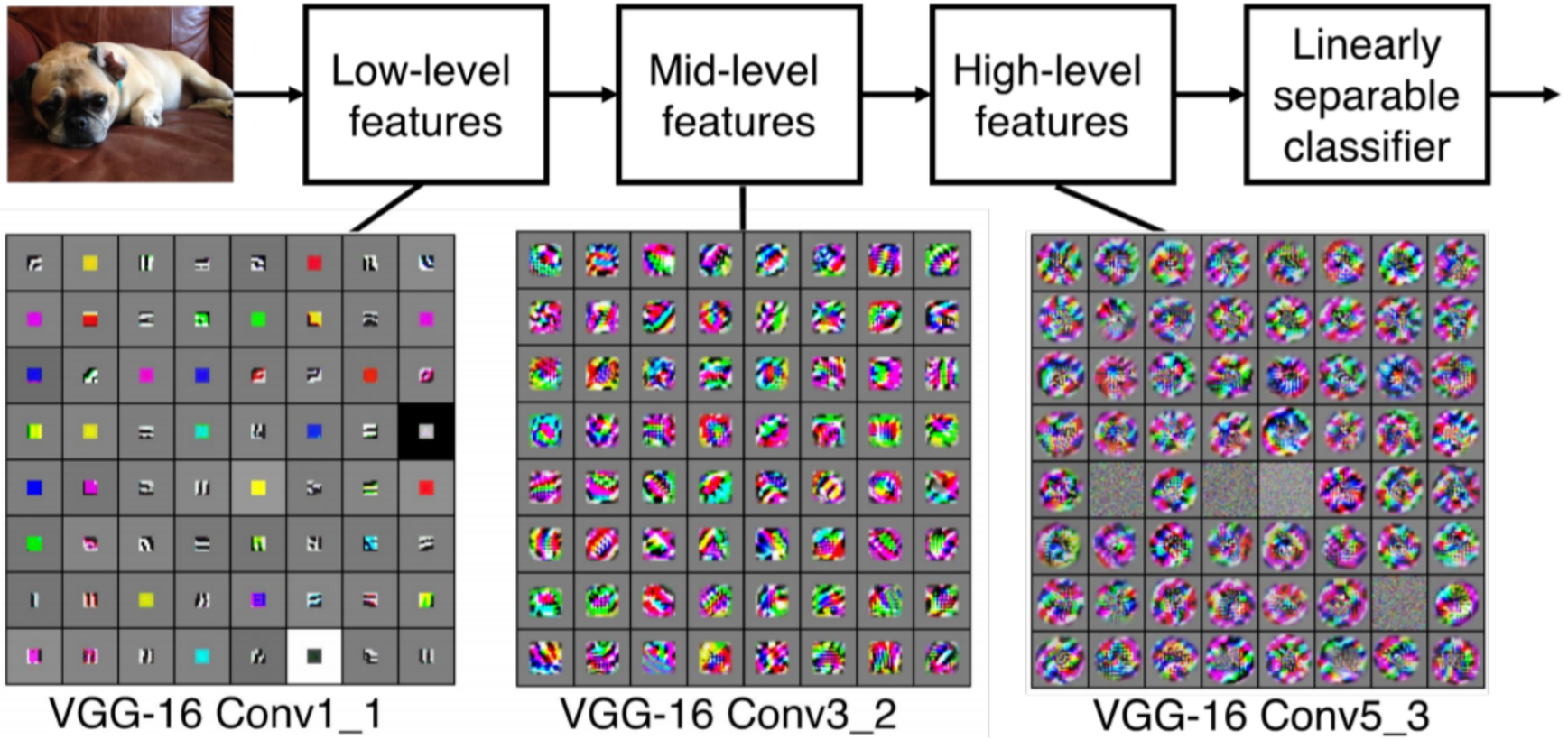
- first convolutional layer, the network has to learn 64 filters with size 3x3 along the input depth (3). Plus, each one of the 64 filters has a bias, so the total number of parameters is 64*3*3*3 + 64 = 1792.

- output of the first convolutional layer will be 224 x 224 x 64.

- In pooling layer, we have to consider the size of the window and the stride.

- To calculate the number of parameters in the fully-connected layers, we have to multiply the number of units in the previous layer by the number of units in the current layer.

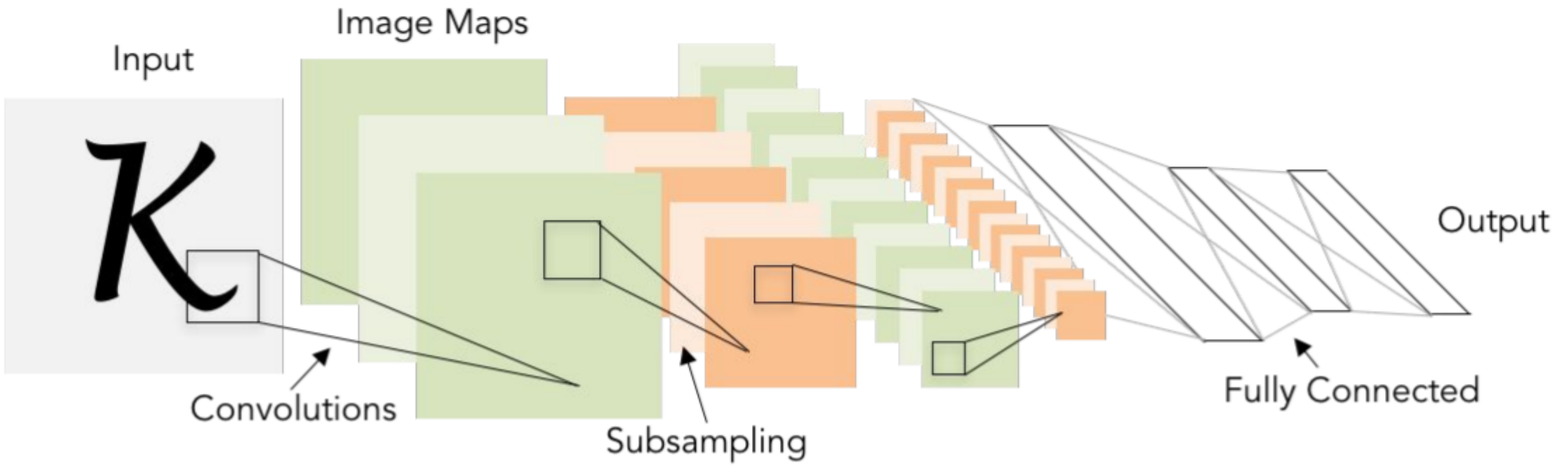
- Number of units in the last convolutional layer will be 7x7x512. So, the total number of parameters in the first fully-connected layer will be 7x7x512x4096 + 4096 = 102764544



Visualization

Lenet Architecture

[LeCun et al., 1998]



Conv filters were 5x5, applied at stride 1
Subsampling (Pooling) layers were 2x2 applied at stride 2
i.e. architecture is [CONV-POOL-CONV-POOL-FC-FC]

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 30, 30, 6)         60
_________________________________________________________________
average_pooling2d_1 (Average (None, 15, 15, 6)         0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 13, 13, 16)        880
_________________________________________________________________
average_pooling2d_2 (Average (None, 6, 6, 16)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 576)               0
_________________________________________________________________
dense_1 (Dense)              (None, 120)               69240
_________________________________________________________________
dense_2 (Dense)              (None, 84)                10164
_________________________________________________________________
dense_3 (Dense)              (None, 10)                850
=================================================================
Total params: 81,194
Trainable params: 81,194
Non-trainable params: 0
_________________________________________________________________
```

Lenet model summary

Flatten1 ->576 (How?)
(6*6*16)

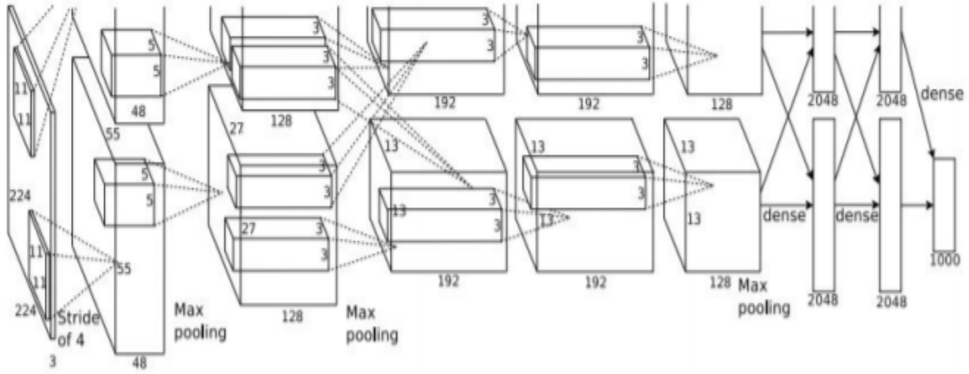
Dense_1 -> 69240 (?)
576*120+120(B)

Total -> 60+880+69240+10164+850

number_parameters
=
output_size * (input_size + 1)

Q - How we got 10164 parameters @ dense_2 layer

Alexnet



Layer		Feature Map	Size	Kernel Size	Stride	Activation
Input	Image	1	227x227x3	-	-	-
1	Convolution	96	55 x 55 x 96	11x11	4	relu
	Max Pooling	96	27 x 27 x 96	3x3	2	relu
2	Convolution	256	27 x 27 x 256	5x5	1	relu
	Max Pooling	256	13 x 13 x 256	3x3	2	relu
3	Convolution	384	13 x 13 x 384	3x3	1	relu
4	Convolution	384	13 x 13 x 384	3x3	1	relu
5	Convolution	256	13 x 13 x 256	3x3	1	relu
	Max Pooling	256	6 x 6 x 256	3x3	2	relu
6	FC	-	9216	-	-	relu
7	FC	-	4096	-	-	relu
8	FC	-	4096	-	-	relu
Output	FC	-	1000	-	-	Softmax

[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0
[27x27x96] MAX POOL1: 3x3 filters at stride 2
[27x27x96] NORM1: Normalization layer
[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2
[13x13x256] MAX POOL2: 3x3 filters at stride 2
[13x13x256] NORM2: Normalization layer
[13x13x384] CONV3: 384 3x3 filters at stride 1, pad 1
[13x13x384] CONV4: 384 3x3 filters at stride 1, pad 1
[13x13x256] CONV5: 256 3x3 filters at stride 1, pad 1
[6x6x256] MAX POOL3: 3x3 filters at stride 2
[4096] FC6: 4096 neurons
[4096] FC7: 4096 neurons
[1000] FC8: 1000 neurons (class scores)

Input: 227x227x3 images
First layer (CONV1): 96 11x11 filters applied at stride 4

Q: what is the output volume size?

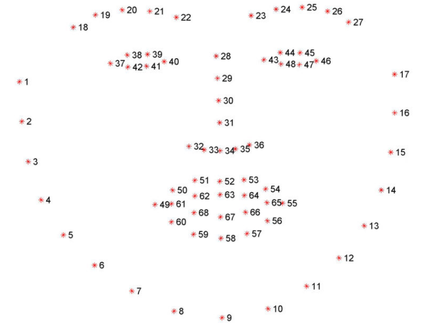
Hint: (227-11)/4+1 = 55

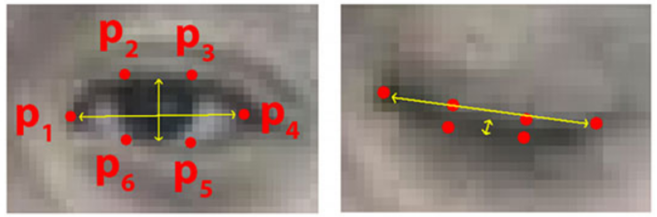
Paramaters?
Input: 227x227x3 images
First layer (CONV1): 96 11x11 filters applied at stride 4

Output volume [55x55x96]
Parameters: (11*11*3)*96 = 34974

Demo

- Build basic CNN n/w and understand its layer wide parameters.
- Apply CNN for MNIST dataset.
- Keras for CIFAR10 dataset.
- Implement Lenet architecture
- Implement facial recognition systems using CNN
- Compare performances of BB - CNN networks
- Implement driver drowsiness detection system.





Results

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 26, 26, 32)        320
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 13, 13, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 11, 11, 64)        18496
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 5, 5, 64)          0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 3, 3, 64)          36928
=================================================================
Total params: 55,744
Trainable params: 55,744
Non-trainable params: 0
_________________________________________________________________
--- 0.12456607818603516 seconds ---
```

Basic CNN architecture

Keras for cifar-10

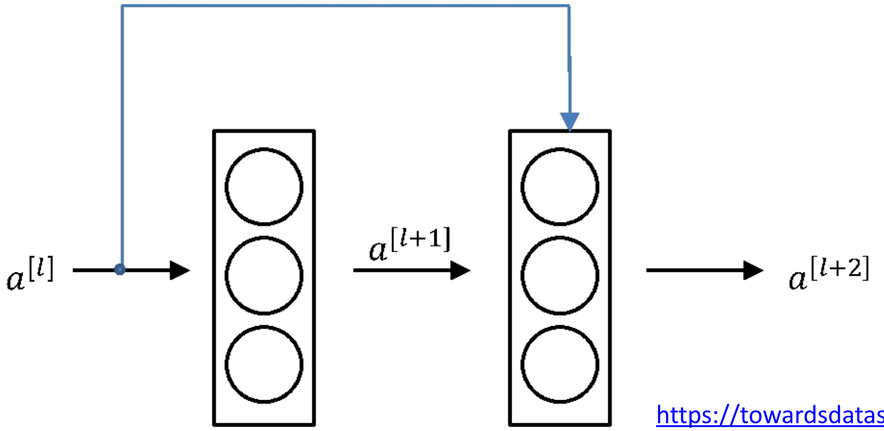
```
[INFO] loading CIFAR-10 data...
[INFO] training network...
Train on 50000 samples, validate on 10000 samples
Epoch 1/5
2019-01-20 09:07:23.620143: I tensorflow/core/platform/cpu_feature_guard.cc:141] Your CPU supports instructions that this TensorF
50000/50000 [==============================] - 138s 3ms/step - loss: 1.8398 - acc: 0.3450 - val_loss: 1.7094 - val_acc: 0.3941
Epoch 2/5
50000/50000 [==============================] - 154s 3ms/step - loss: 1.6482 - acc: 0.4182 - val_loss: 1.6671 - val_acc: 0.4047
Epoch 3/5
50000/50000 [==============================] - 145s 3ms/step - loss: 1.5664 - acc: 0.4462 - val_loss: 1.6752 - val_acc: 0.4035
Epoch 4/5
50000/50000 [==============================] - 144s 3ms/step - loss: 1.5118 - acc: 0.4661 - val_loss: 1.6056 - val_acc: 0.4212
Epoch 5/5
50000/50000 [==============================] - 144s 3ms/step - loss: 1.4620 - acc: 0.4843 - val_loss: 1.5023 - val_acc: 0.4649
[INFO] evaluating network...
              precision    recall  f1-score   support

    airplane       0.45      0.65      0.53      1000
  automobile       0.64      0.59      0.62      1000
        bird       0.29      0.53      0.37      1000
         cat       0.35      0.38      0.36      1000
        deer       0.47      0.25      0.33      1000
         dog       0.48      0.21      0.30      1000
        frog       0.51      0.53      0.52      1000
       horse       0.45      0.64      0.53      1000
        ship       0.76      0.38      0.50      1000
       truck       0.63      0.49      0.55      1000

avg / total       0.50      0.46      0.46     10000
```

```python
import numpy as np

#define the sigmoid activation function,
# which is commonly used in neural networks. It takes an input x and returns the sigmoid function's output.

def sigmoid(x):
    return 1 / (1 + np.exp(-x))

# define the derivative of the sigmoid function, which is used during backpropagation for calculating gradients.
def sigmoid_derivative(x):
    return x * (1 - x)


class NeuralNetwork:
    def __init__(self, input_size, hidden_size, output_size):
        self.weights_input_hidden = np.random.rand(input_size, hidden_size)
        self.bias_hidden = np.zeros((1, hidden_size))
        self.weights_hidden_output = np.random.rand(hidden_size, output_size)
        self.bias_output = np.zeros((1, output_size))

    def forward(self, inputs):
        self.inputs = inputs
        self.hidden_layer_input = np.dot(inputs, self.weights_input_hidden) + self.bias_hidden
        self.hidden_layer_output = sigmoid(self.hidden_layer_input)
        self.output_layer_input = np.dot(self.hidden_layer_output, self.weights_hidden_output) + self.bias_output
        self.output = sigmoid(self.output_layer_input)
        return self.output

    def backward(self, target, learning_rate):
        loss = target - self.output
        delta_output = loss * sigmoid_derivative(self.output)
        loss_hidden = delta_output.dot(self.weights_hidden_output.T)
        delta_hidden = loss_hidden * sigmoid_derivative(self.hidden_layer_output)

        self.weights_hidden_output += self.hidden_layer_output.T.dot(delta_output) * learning_rate
        self.bias_output += np.sum(delta_output, axis=0, keepdims=True) * learning_rate
        self.weights_input_hidden += self.inputs.T.dot(delta_hidden) * learning_rate
        self.bias_hidden += np.sum(delta_hidden, axis=0, keepdims=True) * learning_rate
```

```python
if __name__ == "__main__":
    input_size = 4
    hidden_size = 8
    output_size = 1

    nn = NeuralNetwork(input_size, hidden_size, output_size)

    training_data = np.array([[0, 0, 1, 1], [0, 1, 1, 1], [1, 0, 1, 0], [1, 1, 1, 0]])
    target_data = np.array([[0], [1], [1], [0]])

    for _ in range(20000):
        nn.forward(training_data)
        nn.backward(target_data, learning_rate=0.001)

    new_data = np.array([0, 1, 0, 0])
    predicted_output = nn.forward(new_data)
    print(f"Predicted output: {predicted_output[0]}")
```

ResNet

- Resnet are build out of residual network
- Basically it is a shortcut defining a residual path between a[l] and a[l+2]
- Addition of $A^{[l]}$ to $A^{[l+2]}$ makes it a residual block $A^{[l]}$ s added before non-ıinearity 'RELU' $A^{[l+2]} = g(Z^{[l+2]} + A^{[l]})$
- It is also known as skip layer as the information from layer $A^{[l]}$ skips a layer or two to pass the information deeper into the n/w teaching the identity function to the n/w which otherwise was difficult for the network to learn. hence forms partial solution to vanishing and gradient
- It allows us to train the deeper networks faster
- Resnet are built by stacking a lots of residual blocks

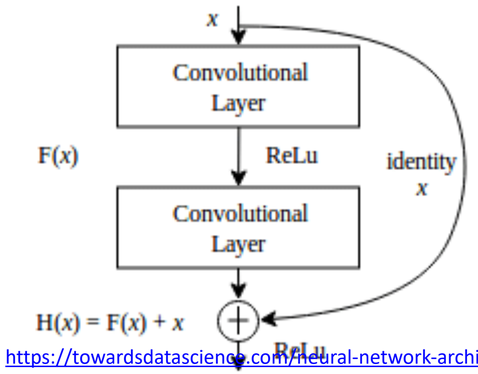


https://towardsdatascience.com/neural-network-architectures-156e5bad51ba

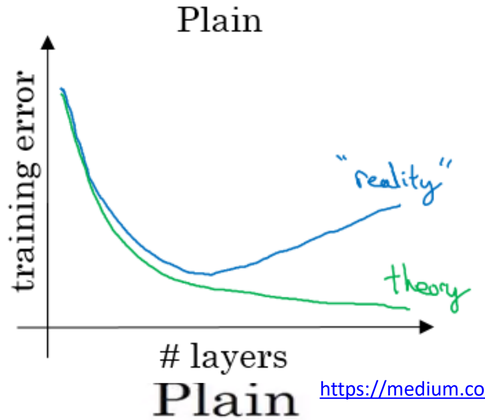
The content is added from different sources

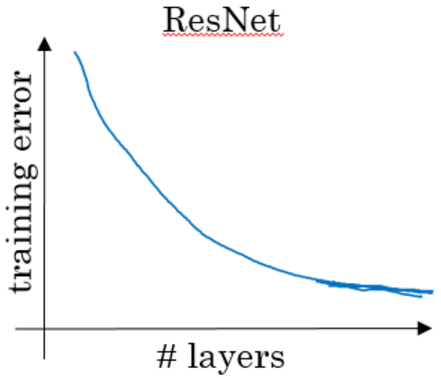
ResNet

- Presence of the many layers, ensures we do not miss out on learning at the same time, the presence of skip connections ensures, we also do not face exploding or vanishing gradients.
- Providing a win-win solution which can be clearly seen from the graph



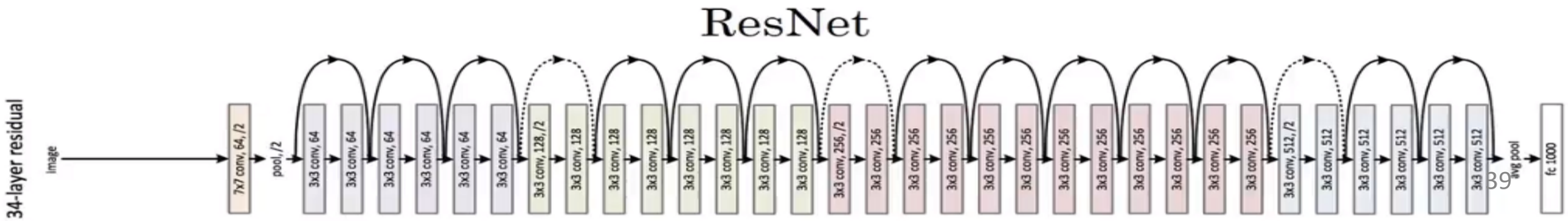
https://towardsdatascience.com/neural-network-architectures-156e5bad51ba





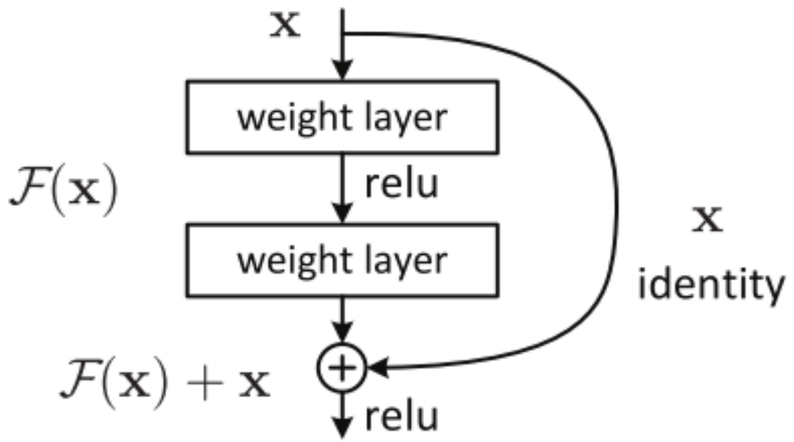
https://medium.com/@himabindu13198/how-resnets-view-the-world-ann-series-2-53693d572407

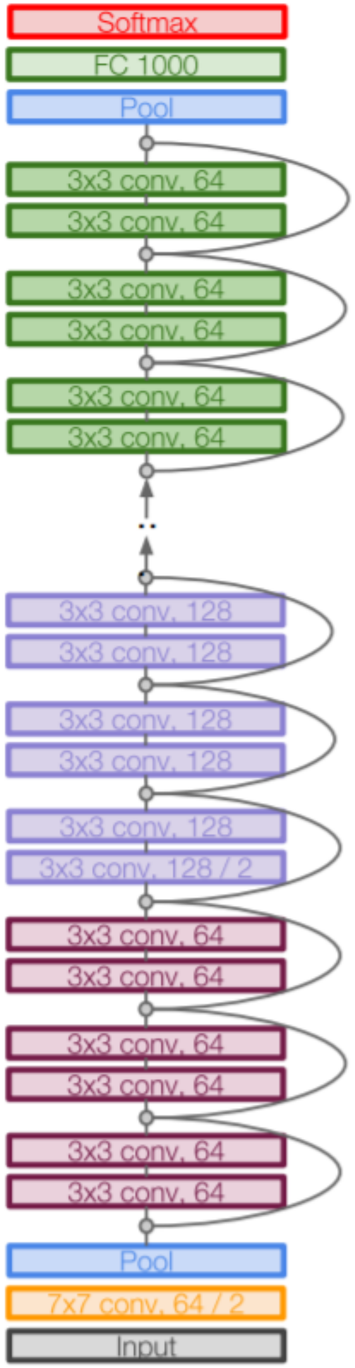
https://images.app.goo.gl/wnkGKpUQQdB9UHSF7



from different sources

ResNet



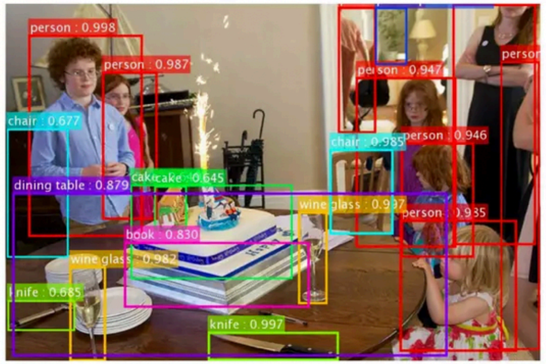


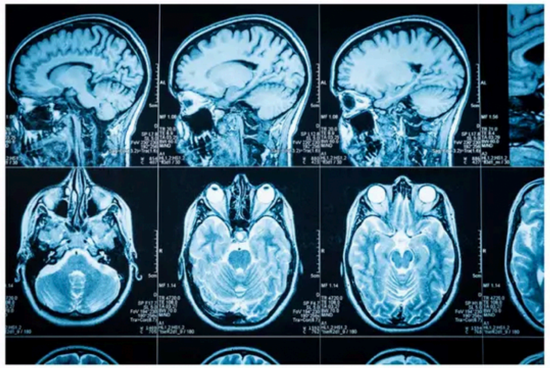
...ontent is added from different sources

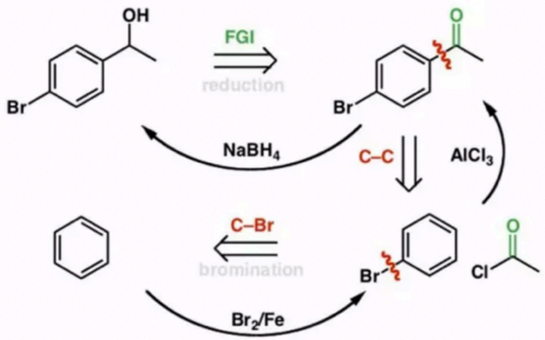
The content is added from different sources

Self-supervised Learning

- DNNs achieve **remarkable success** in various applications
  - They usually **require massive amounts of manually labeled data**
  - The **annotation cost is high** because
    - It is **time-consuming**: e.g., annotating bounding boxes of all objects
    - It requires **expert knowledge**: e.g., medical diagnosis and retrosynthesis







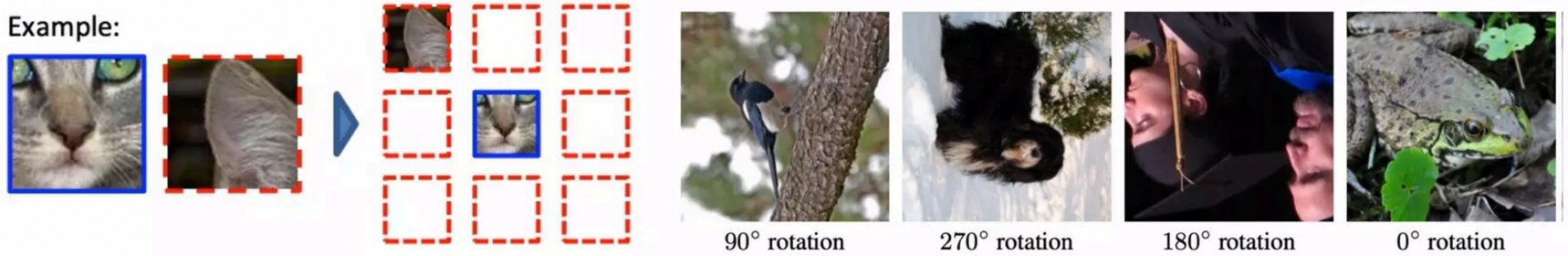
- But, **collecting unlabeled samples is extremely easy** compared to annotation

- **Q.** How to utilize the **unlabeled samples** for learning DNNs?

https://www.slideshare.net/sangwoomo7/selfsupervised-learning-lecture-note

Self-supervised Learning

- **Self-supervision?**
  - It is **a label** constructed **from only input signals** **without human-annotation**
  - Using self-supervision, one can apply supervised learning approaches
  - Examples: Predicting relative location of patches[1] or rotation degree[2]



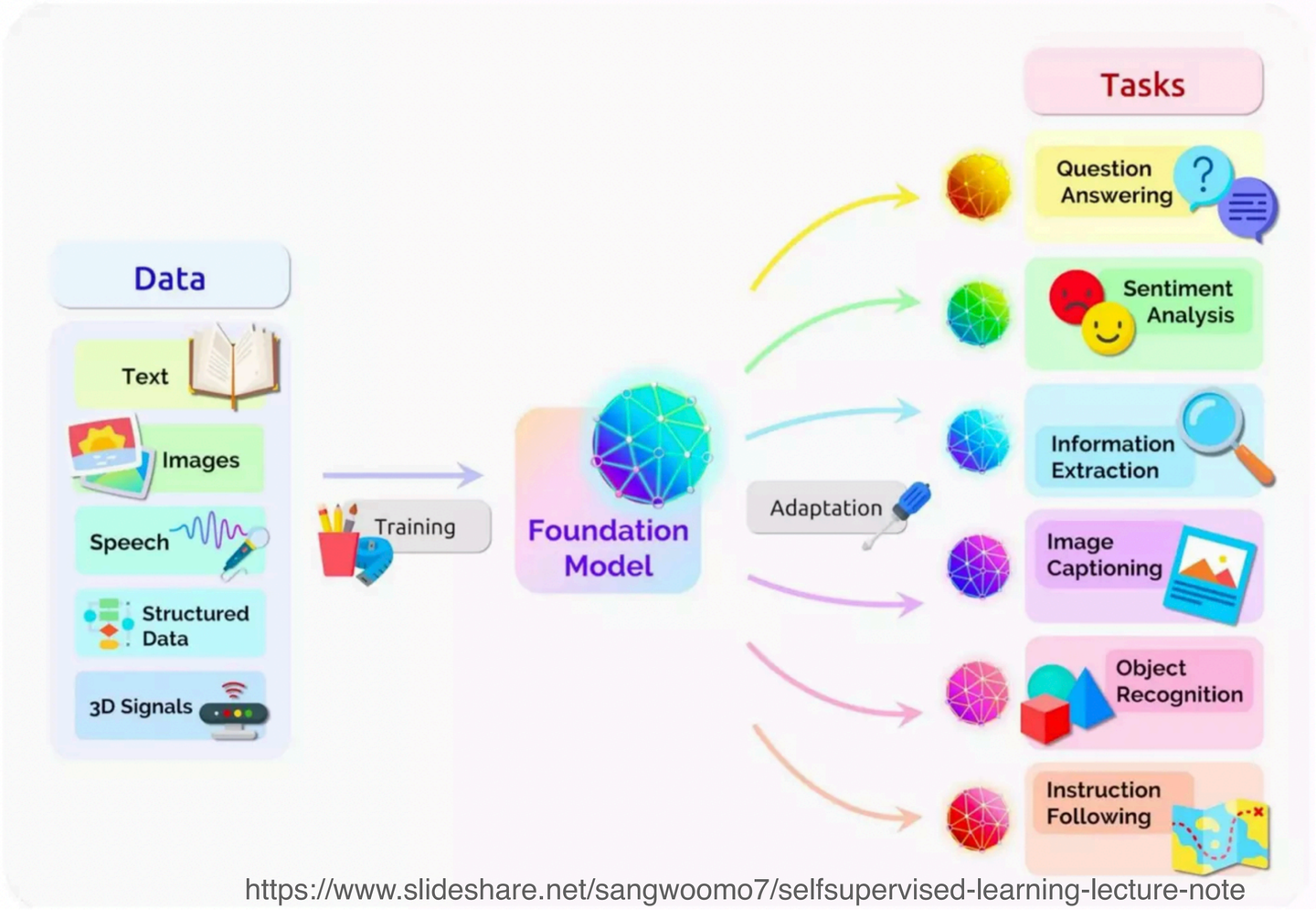
- What can we learn from self-supervised learning?
  - To predict (well-designed) self-supervision, one might **require high-level understanding of inputs**
  - E.g., we should know ⬚ is the right ear of the cat for predicting locations
  - Thus, **high-level representations could be learned w/o human-annotation**

https://www.slideshare.net/sangwoomo7/selfsupervised-learning-lecture-note

The content is added from different sources

Self-supervised Learning

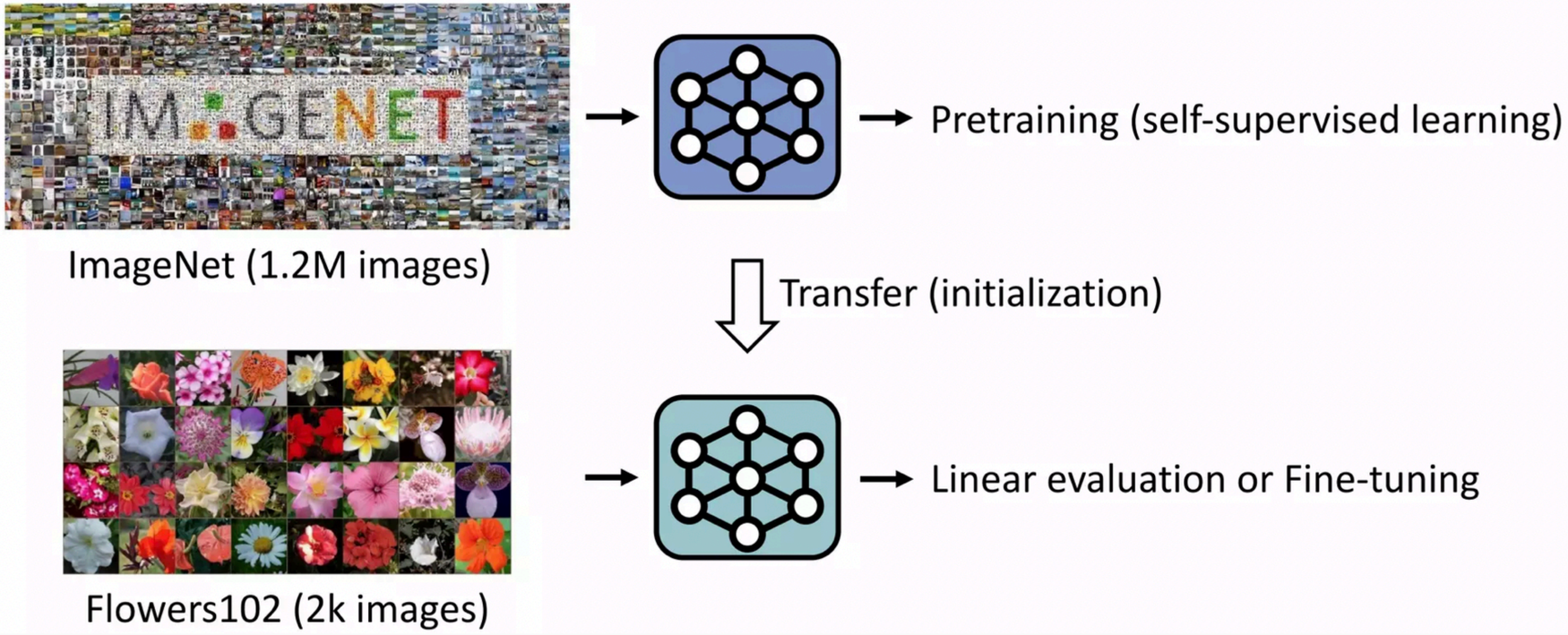
- **Foundation Models**
  - Fixing a foundation model (e.g., trained via self-supervised learning) and only adapting a simple task-specific model is sufficient for many problems
    - E.g., linear classifier upon the SimCLR/BERT backbone



https://www.slideshare.net/sangwoomo7/selfsupervised-learning-lecture-note

Self-supervised Learning

- **How to evaluate the quality of self-supervision?**
  1. Self-supervised learning in a large-scale dataset (e.g., ImageNet)
  2. Transfer the pretrained network to various downstream tasks
     - Linear probing: freeze the network and training only the linear classifier
       ⇒ it directly evaluates the learned representation qualities
     - Fine-tuning whole parameters



https://www.slideshare.net/sangwoomo7/selfsupervised-learning-lecture-note

The content is added from different sources

Self-supervised Learning

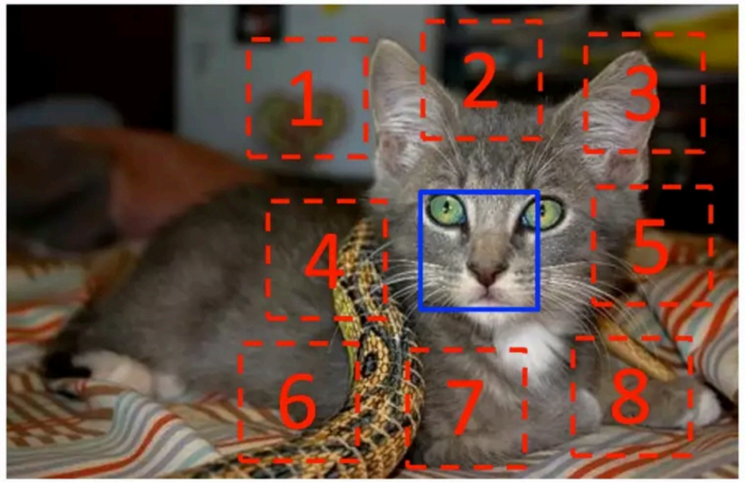
- Current SSL approaches can be categorized into **3 groups:**
  - Let X be data, $Z(X)$ be representation, and Y be pretext label
  - I denotes mutual information (MI) of two random variables

1. **Pretext task:** Maximize $I(Z(X); Y)$ where Y is pretext label of X
2. **Invariance:** Maximize $I(Z(X_1); Z(X_2))$ where X_1, X_2 are invariant data
3. **Generation:** Maximize $I(Z(\tilde{X}); X)$ where $\tilde{X}$ is perturbed version of X

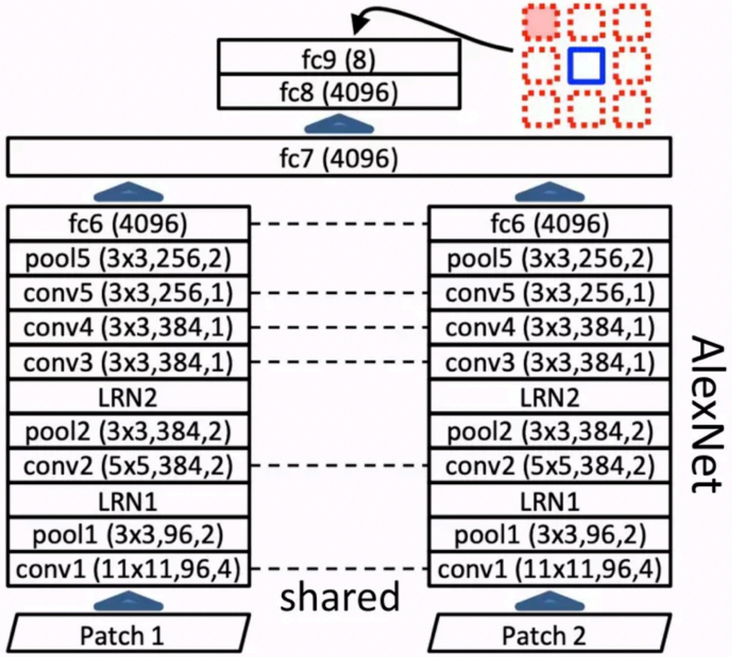
https://www.slideshare.net/sangwoomo7/selfsupervised-learning-lecture-note

Self-supervised Learning

- **Context Prediction** [Doersch et al., 2015]
  - From a natural image, extract 3x3 patches
  - **Patch 1:** The center patch
  - **Patch 2:** Select one of other patches randomly
  - **Task:** Given **Patch 1 & 2**, predict the location of the second patch (8-way classification)



X = (,); Y = 3

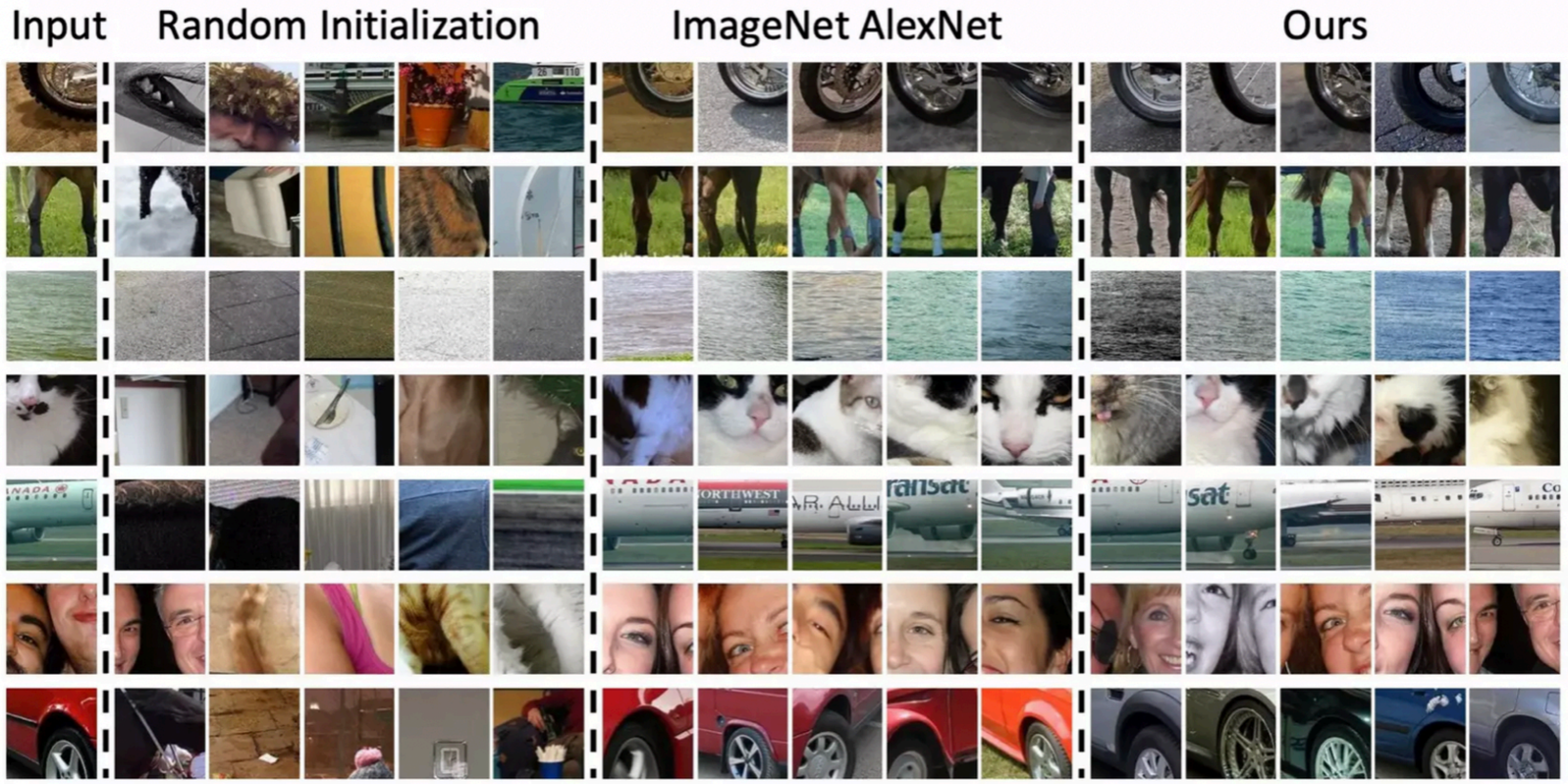


- Each patch's embedding is computed by one **shared** encoder (AlexNet)

https://www.slideshare.net/sangwoomo7/selfsupervised-learning-lecture-note

Self-supervised Learning

- **Context Prediction** [Doersch et al., 2015]
  - **Task:** Given **Patch 1 & 2**, predict the location of the second patch (8-way classification)
  - This pretext task assigns similar representations to semantically similar patches
    - Qualitative analysis of nearest neighbors of learned representations

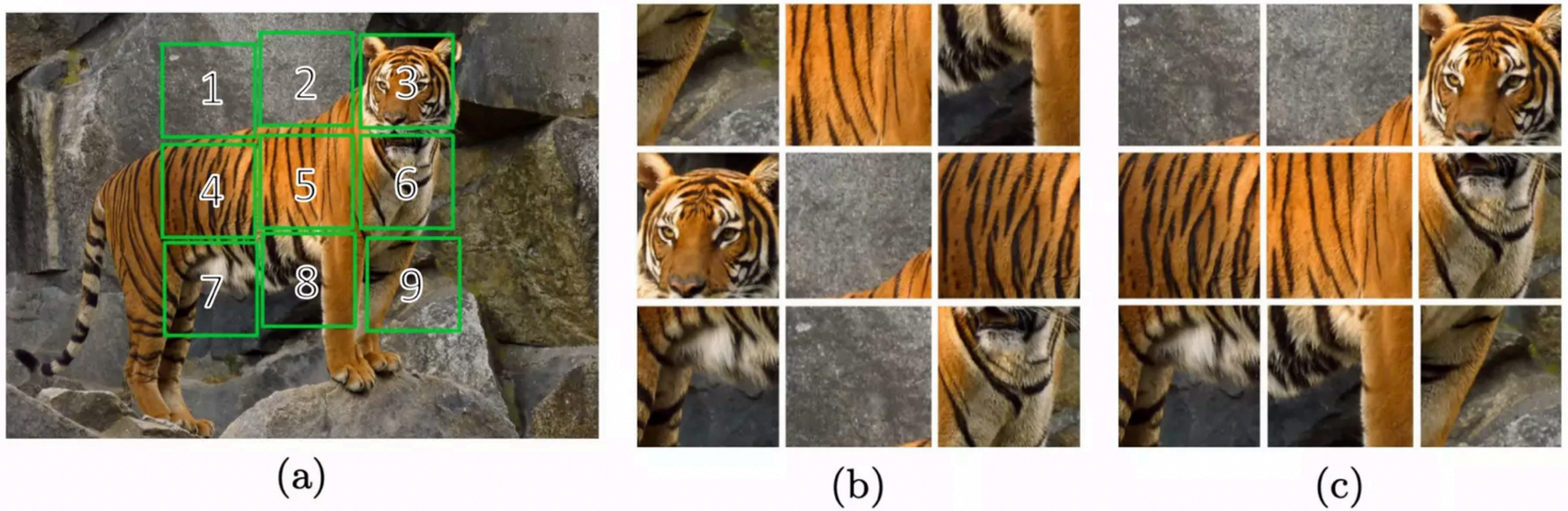


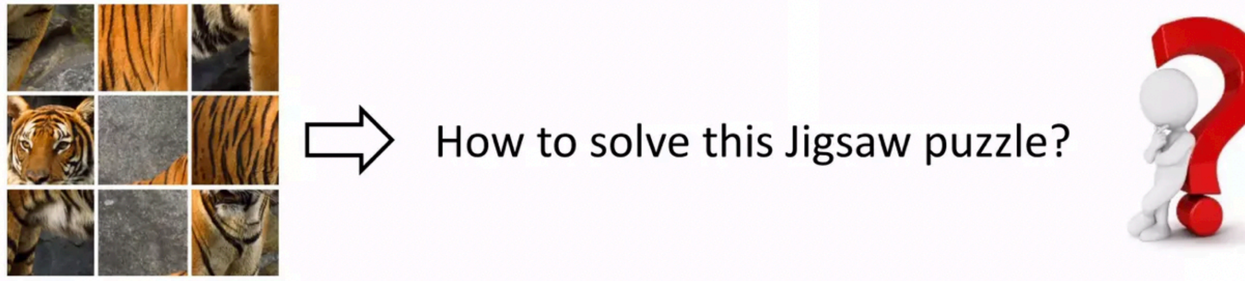
https://www.slideshare.net/sangwoomo7/selfsupervised-learning-lecture-note

The content is added from different sources

Self-supervised Learning

- **Jigsaw Puzzle** [Noroozi & Favaro, 2016]
  - Extension of [Doersch et al., 2015]
  - (a) Extract 3x3 patches from a natural image; (b) **permute** the patches randomly
  - **Task:** From (b) the shuffled patches, find **which permutation is applied**



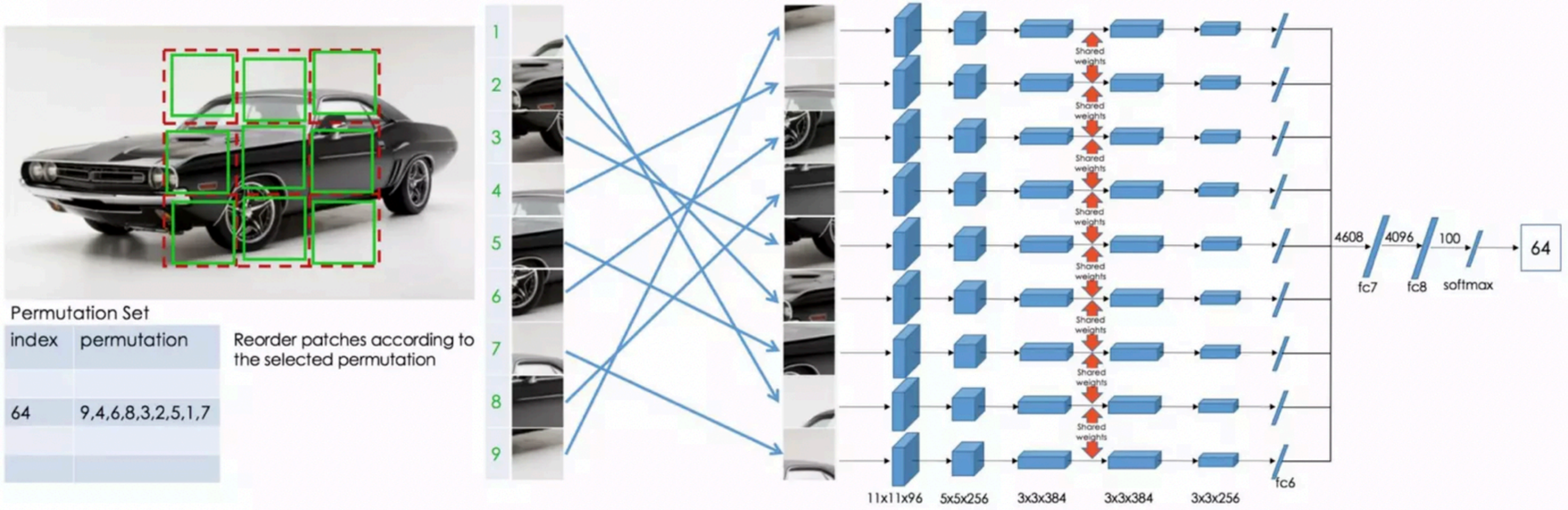


https://www.slideshare.net/sangwoomo7/selfsupervised-learning-lecture-note

The content is added from different sources

Self-supervised Learning

- **Jigsaw Puzzle** [Noroozi & Favaro, 2016]
  - Extension of [Doersch et al., 2015]
  - (a) Extract 3x3 patches from a natural image; (b) **permute** the patches randomly
  - **Task:** From (b) the shuffled patches, find **which permutation is applied**

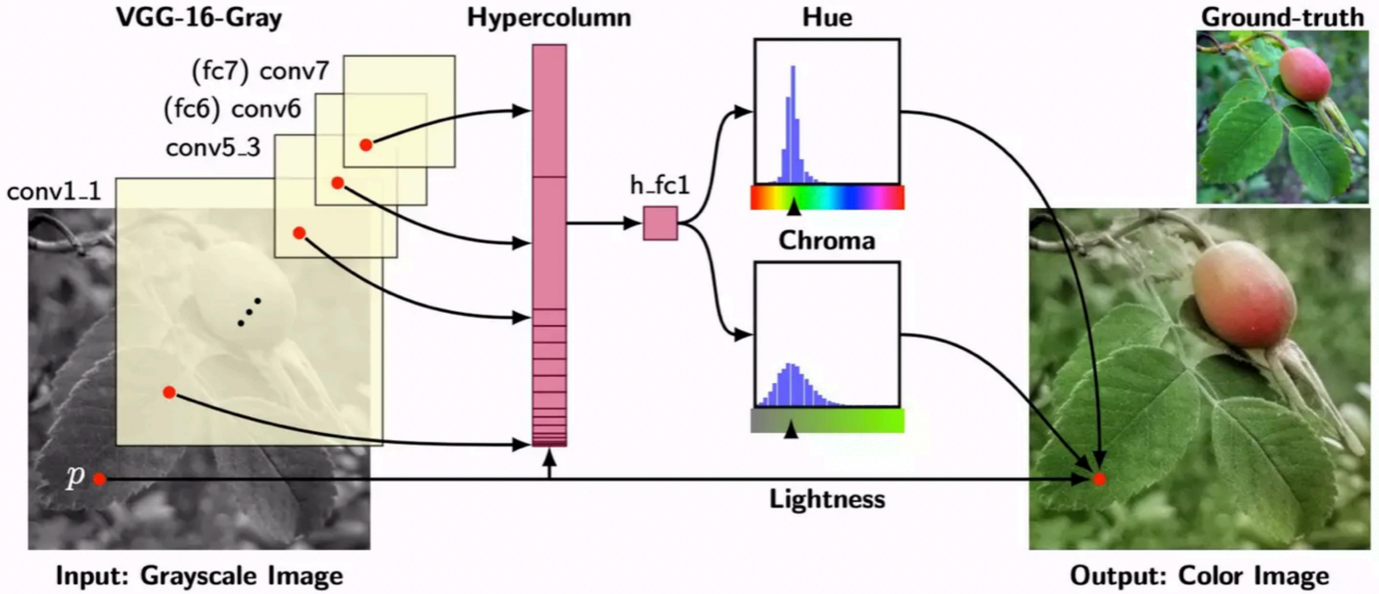


  - Each patch's embedding is computed by one **shared** encoder
  - There are too many permutations (9!=362k) ⇒ choose a subset of them
    - Empirically, **neither simple nor ambiguous tasks** achieve better performance

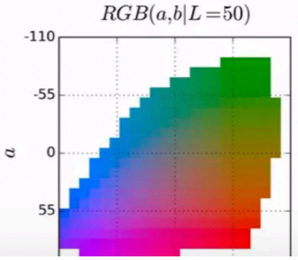
https://www.slideshare.net/sangwoomo7/selfsupervised-learning-lecture-note

Self-supervised Learning

- **Colorization** [Zhang et al., 2016]
  - **Task:** Predict color information from a grayscale image
  - Colorization requires **dense prediction**
  - **⇒ Hypercolumn:** concatenate feature maps of different layers



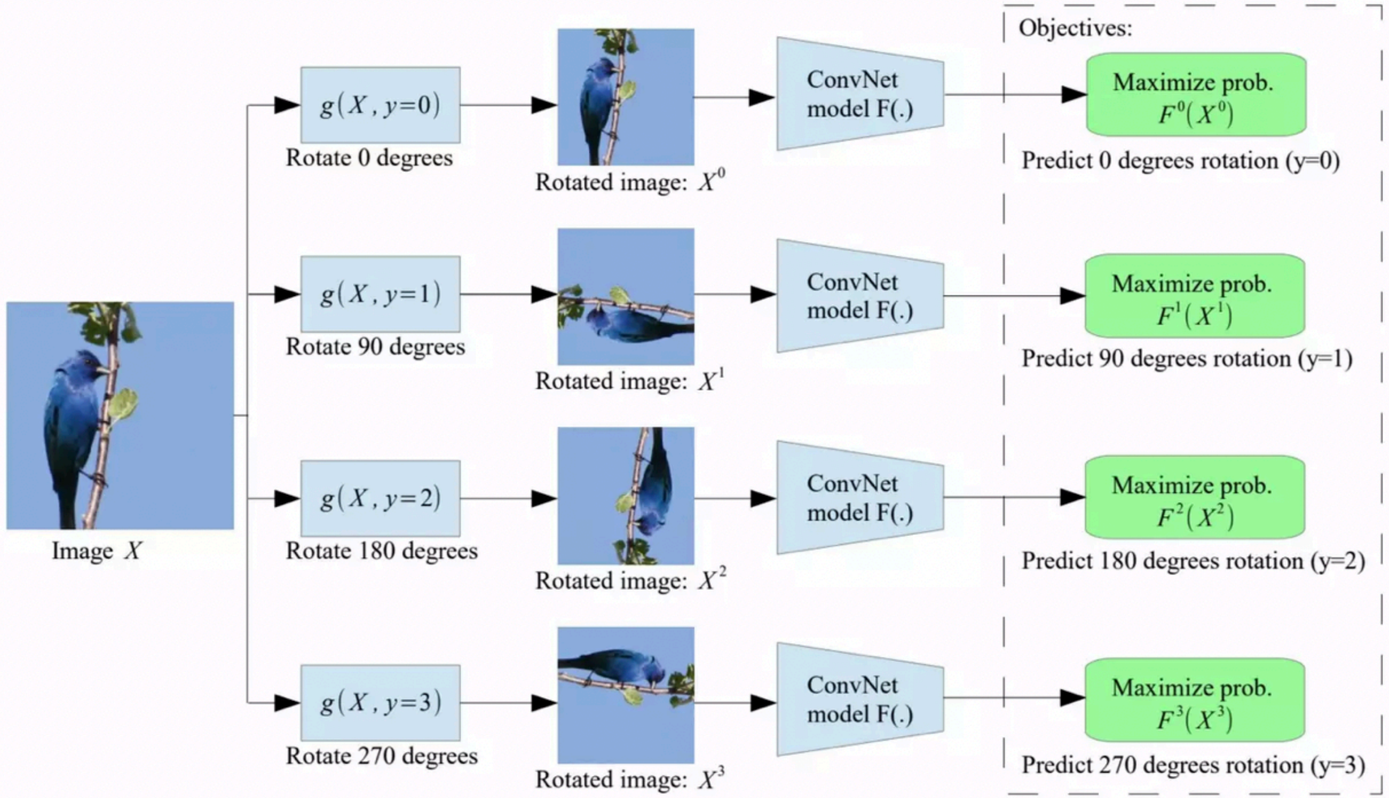
- Formulate colorization as **classification** instead of regression
  - Quantize Lab color space for classification
  - This can handle **multi-modal** color distributions well



https://www.slideshare.net/sangwoomo7/selfsupervised-learning-lecture-note

Self-supervised Learning

- **Rotation** [Gidaris et al., 2018]
  - **Task: Predict the rotation degree** from a rotated image (4-way classification)



- What is the optimal number of classes (rotations)?
  - Empirically, using 4 rotations (0°, 90°, 180°, 270°) is best

https://www.slideshare.net/sangwoomo7/selfsupervised-learning-lecture-note

Self-supervised Learning

- **Limitations** on handcrafted pretext tasks
  1. **Domain-specific knowledge is required to design self-supervision**
     - In different domains (e.g., audio), existing methods might be not working

Core idea of invariance-based learning:

- **Invariance:** Representations of related samples should be similar
- **Contrast** (optional)**:** Representations of unrelated samples should be dissimilar

Positive pair $f($  $) \approx f($  $)$

Negative pair $f($  $) \neq f($  $)$

- **Q)** How to construct positive/negative pairs in the unsupervised setting?

https://www.slideshare.net/sangwoomo7/selfsupervised-learning-lecture-note

Self-supervised Learning

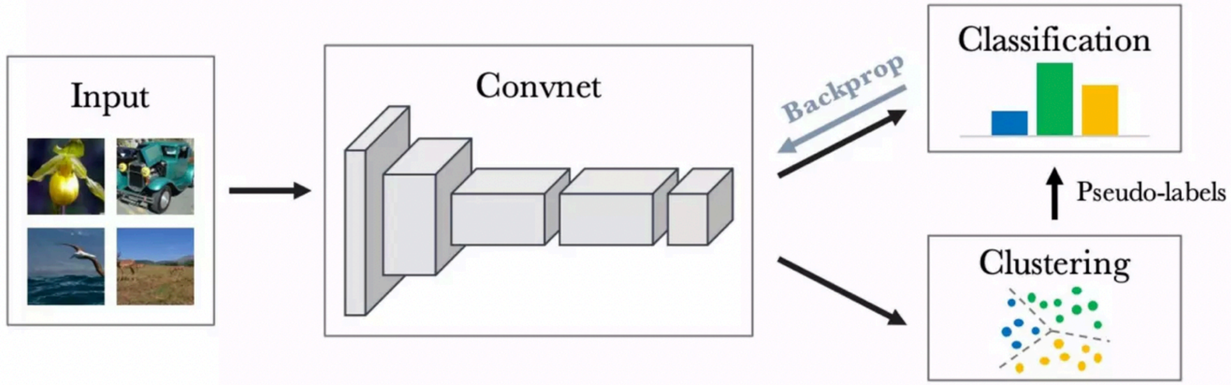
- **A)** Positive samples are constructed from
  - Similar samples (e.g., in the same cluster)
  - Same instance of different data augmentation
  - Additional structures (e.g., multi-view images, video)

  (negative samples = not positive samples)

The content is added from different sources

Self-supervised Learning

- **DeepCluster** [Caron et al., 2018]
  - **Idea:** Clustering on embedding space provides pseudo-labels



  - **Simple method:** Alternate between
    1. Clustering the features to produce pseudo-labels
    2. Updating parameters by predicting these pseudo-labels

  - How to avoid **trivial solutions**?
    - Empty cluster ⇐ feature quantization (it reassigns empty clusters)
    - Imbalanced sizes of clusters ⇐ over-sampling

https://www.slideshare.net/sangwoomo7/selfsupervised-learning-lecture-note

The content is added from different sources

Self-supervised Learning

- **Instance Discrimination** [Wu et al., 2018]
  - **Idea:** Each image belongs to an unique class
  - **Non-parameteric classifier**

$$P(i|\mathbf{v}) = \frac{\exp(\mathbf{v}_i^\top \mathbf{v}/\tau)}{\sum_{j=1}^{n} \exp(\mathbf{v}_j^\top \mathbf{v}/\tau)}$$

  - Computing $P(i|\mathbf{v})$ is **inefficient** because it requires all $\mathbf{v}_j = f_\theta(\mathbf{x}_j)$ and $\mathbf{v}_j^\top \mathbf{v}$
  - **Solution 1:** Memory bank
    - Store all $\mathbf{v}_j$ in memory and update them for each mini-batch
    - To stabilize training, representations in memory bank are **momentum-updated**

$$\mathbf{v}_i^{(t)} \leftarrow m\mathbf{v}_i^{(t-1)} + (1-m)\mathbf{v}_i^{\texttt{new}}$$

Representations in memory bank

Computed by current encoder

https://www.slideshare.net/sangwoomo7/selfsupervised-learning-lecture-note

Self-supervised Learning

- **Limitations** in contrastive learning (with negatives)
  - It is sensitive to the number of negative ⇒ a large batch size or a queue is required
  - Are all the different instances negative?

Positive pair $f($  $) \approx f($  $)$

Negative pair $f($  $) \neq f($  $)$

This relation might be not true

- **Q)** can we learn representations without negative samples?
- Simply minimizing $\|f($  $) - f($  $)\|$ leads to mode collapse, i.e., $\forall x, f(x) = c$

- **Next:** Positive-only approaches

https://www.slideshare.net/sangwoomo7/selfsupervised-learning-lecture-note