Machine Learning crash course

(Part-5)

Ahmed Hammad

Convolution based neural network for image recognition

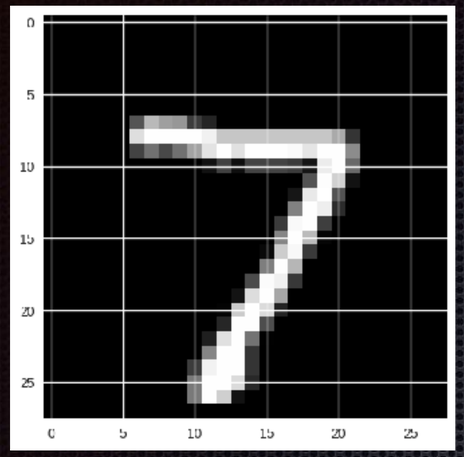
Introduction

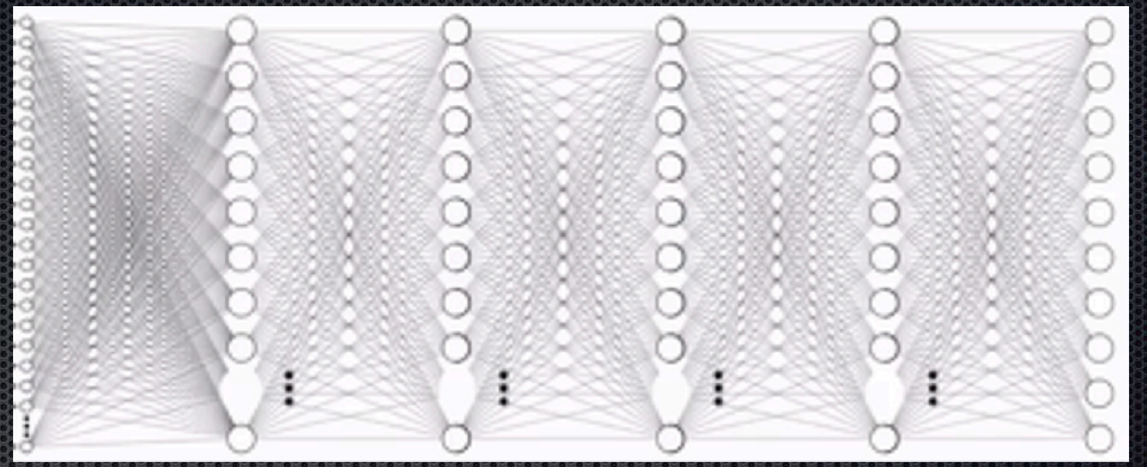
Why do we need CNN models ? DNNs can also analyze images !!

Introduction

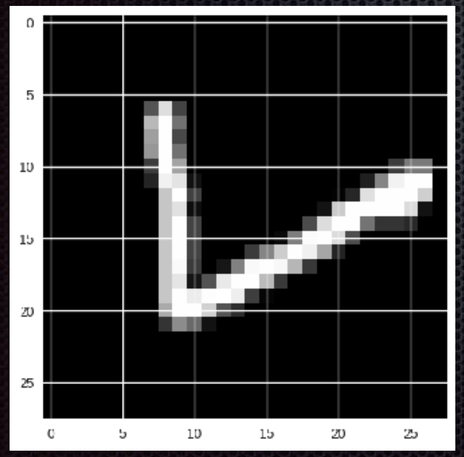
Trained DNN from lecture 4

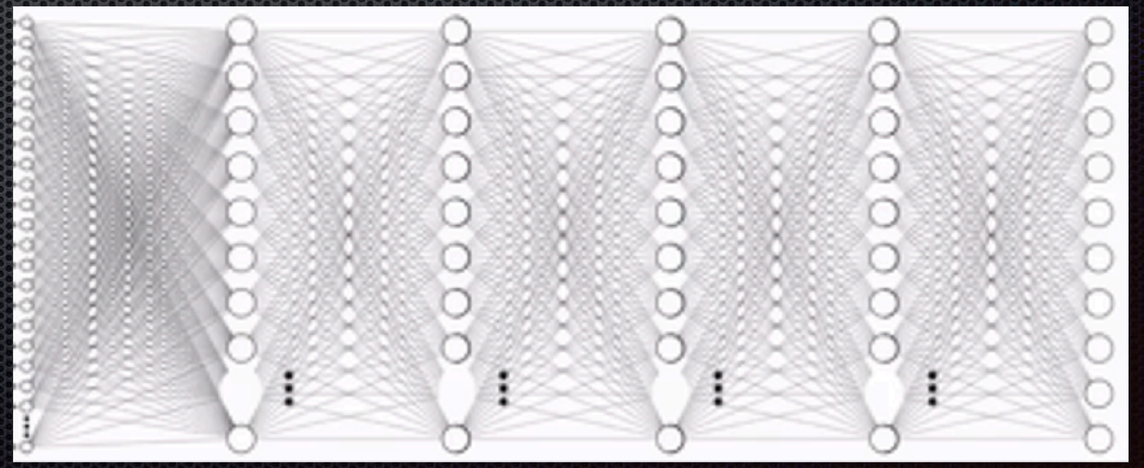
Why do we need CNN models ? DNNs can also analyze images !!





1.5e-20
4.6e-16
1.5e-14
1.5e-15
3.4e-23
1.2e-22
4.2e-27
1.0e+00
1.2e-19
3.4e-09

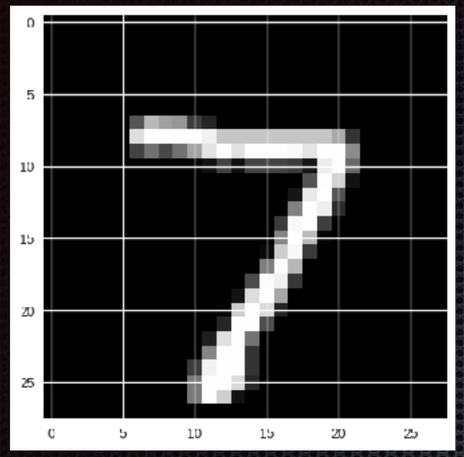


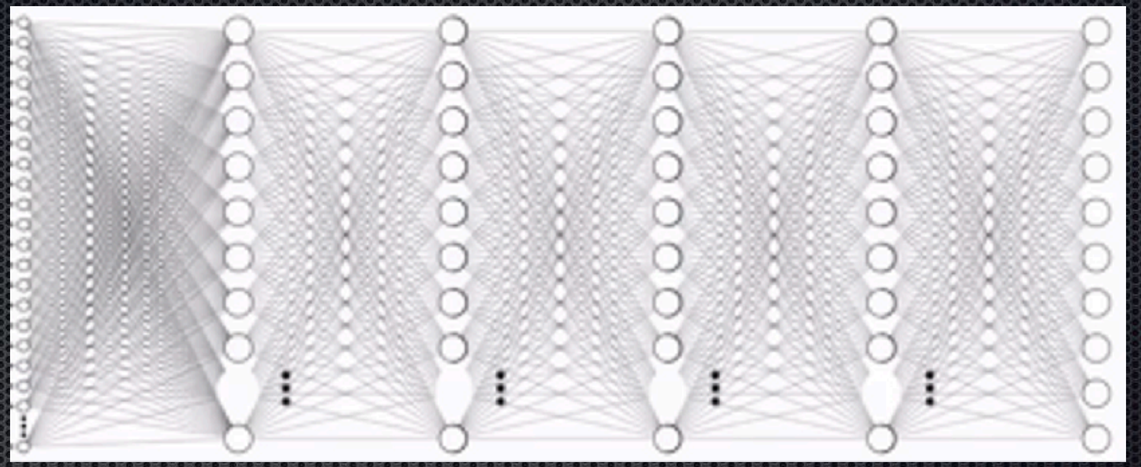


2.3e-04
1.8e-06
2.1e-04
4.3e-07
9.5e-06
1.0e-02
9.7e-01
1.2e-07
1.8e-02
1.3e-06

Introduction

Why do we need CNN models ? DNNs can also analyze images !!





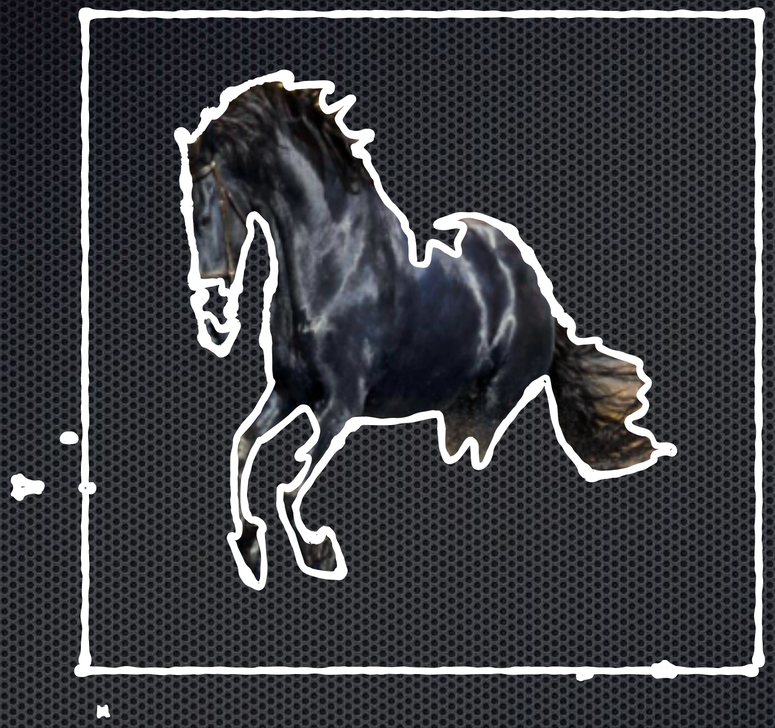
As a fully connected model

$$\text{Neuron Output} = \text{Activation function}\left(\sum_m \omega_m \, x_m + b\right)$$

The model learns the correlation between all pixels into the image and construct a global information about the image

Introduction

Original image

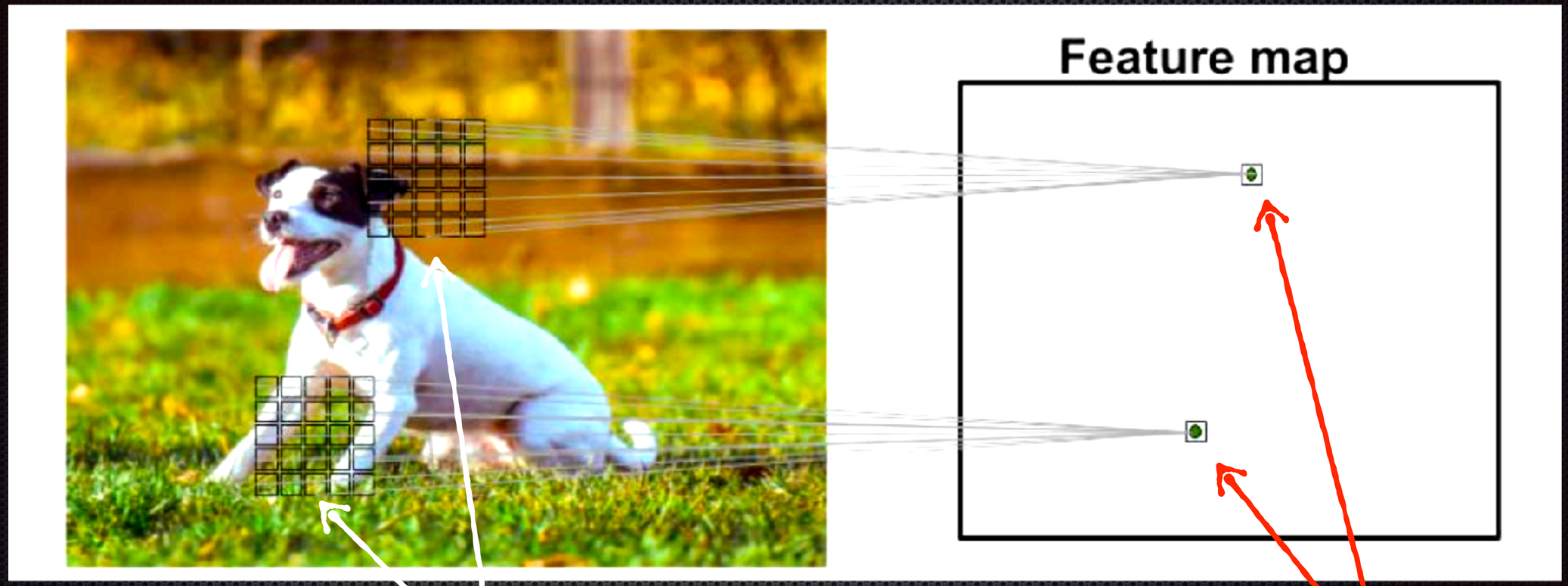


Learned global information

Learned local information

Can we construct a model that learns the local information individually as well as the global information ?

Introduction



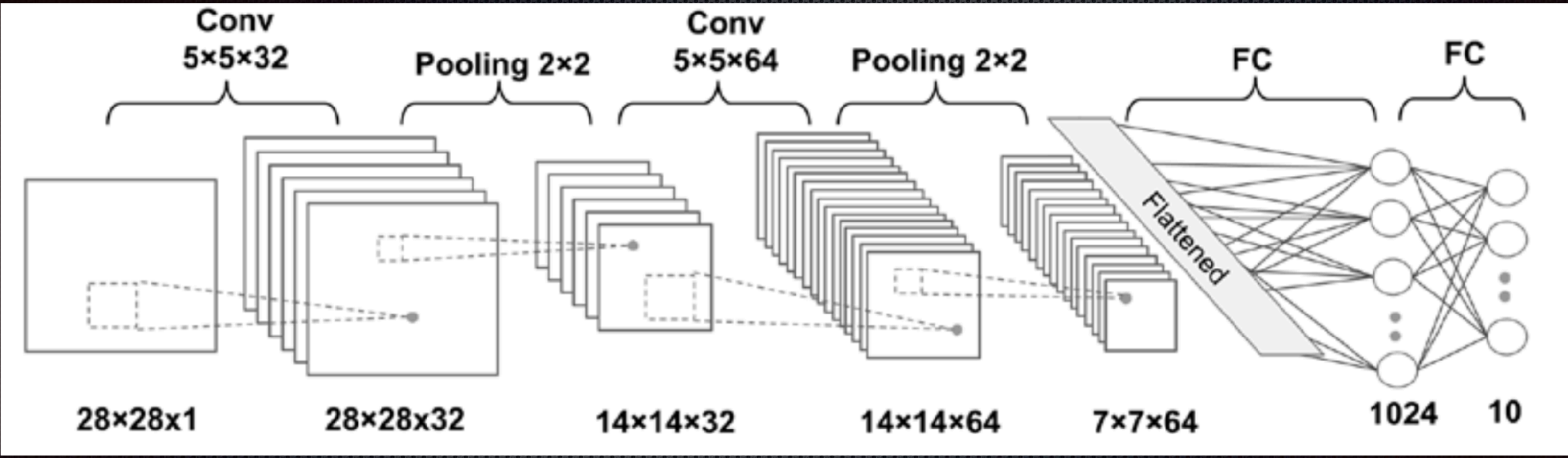
Filters to capture the local features of the images

Disconnected Patches

Filters share their weights internally but not with other filters

Introduction

Example of deep CNN model



CNN = Convolution layers + Fully connected layers

Introduction

How does the CNN work in detail ?

Convolution kernels

1D convolution $y = x * \omega = y[i] = \sum^{k} x[i-k]\omega[k]$

Input $x[i]$

1	2	3	4	5

Dimension (1,5)

Filter $\omega[k=0]$

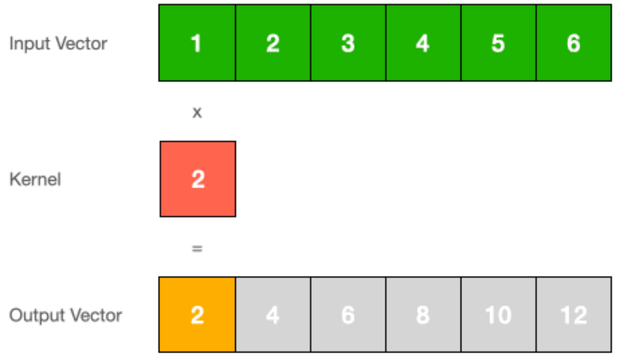
2

Feature map $y[i]$

2	4	6	8	10

Dimension (1,5)

https://jinglescode.github.io/2020/11/01/how-convolutional-layers-work-deep-learning-neural-networks/



Convolution kernels

1D convolution $$y = x * \omega = y[i] = \sum^{k} x[i-k]\omega[k]$$

Input $x[i]$

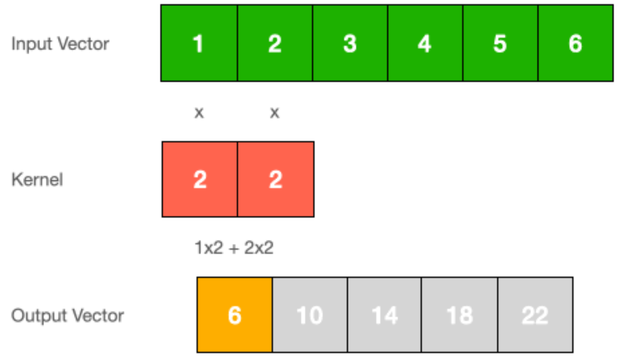
Dimension (1,5)

Filter $\omega[k = 0, 1]$

Feature map $y[i]$

Dimension (1,4)

https://jinglescode.github.io/2020/11/01/how-convolutional-layers-work-deep-learning-neural-networks/



Convolution kernels

1D convolution $$y = x * \omega = y[i] = \sum^{k} x[i-k]\omega[k]$$

Input $x[i]$

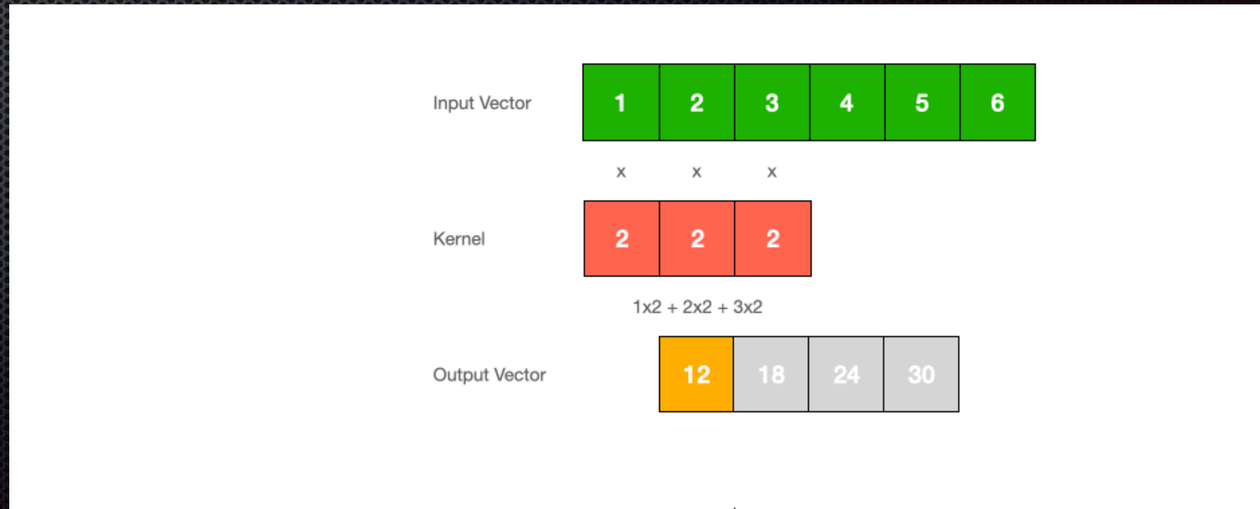
Dimension (1,5)

Filter $\omega[k = 0, 1, 2]$

Feature map $y[i]$

Dimension (1,3)

https://jinglescode.github.io/2020/11/01/how-convolutional-layers-work-deep-learning-neural-networks/



Padding

Padding is used to keep the same dimensions of the output as the input

Input $x_p[i]$

Dimension
(1,5) + padding

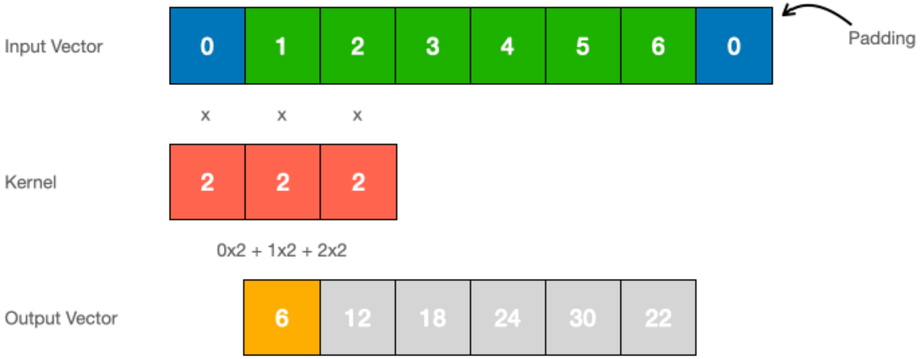
Filter $\omega[k = 0, 1, 2]$

Feature map $y[i]$

Dimension
(1,5)

$$y[i] = \sum_{k=0}^{k=m-1} x_p[i + m - k]\omega[k]$$

https://jinglescode.github.io/2020/11/01/how-convolutional-layers-work-deep-learning-neural-networks/



Shifts of the filter (strides)

Stride: the step size of the shifting filter

Input $x[i]$

1	2	3	4	5	6

Dimension
(1,5)

Filter $\omega[k = 0, 1, 2]$

2	2	2

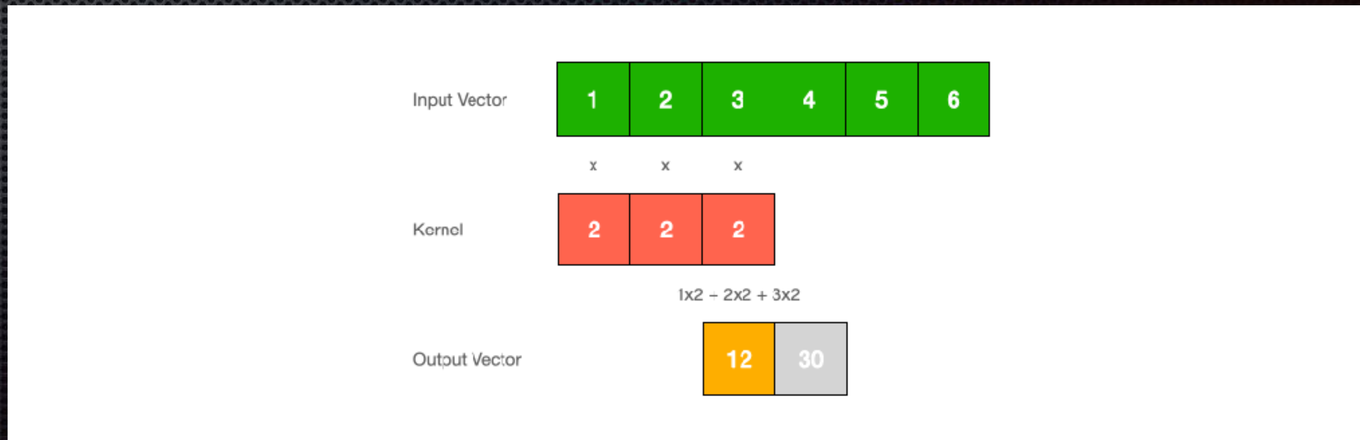
Stride = 3

Feature map $y[i]$

12	30

Dimension
(1,3)

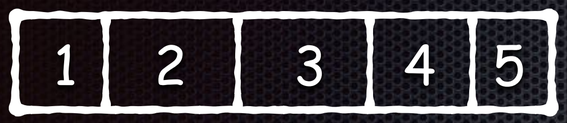
https://jinglescode.github.io/2020/11/01/how-convolutional-layers-work-deep-learning-neural-networks/



Size of the convolution output

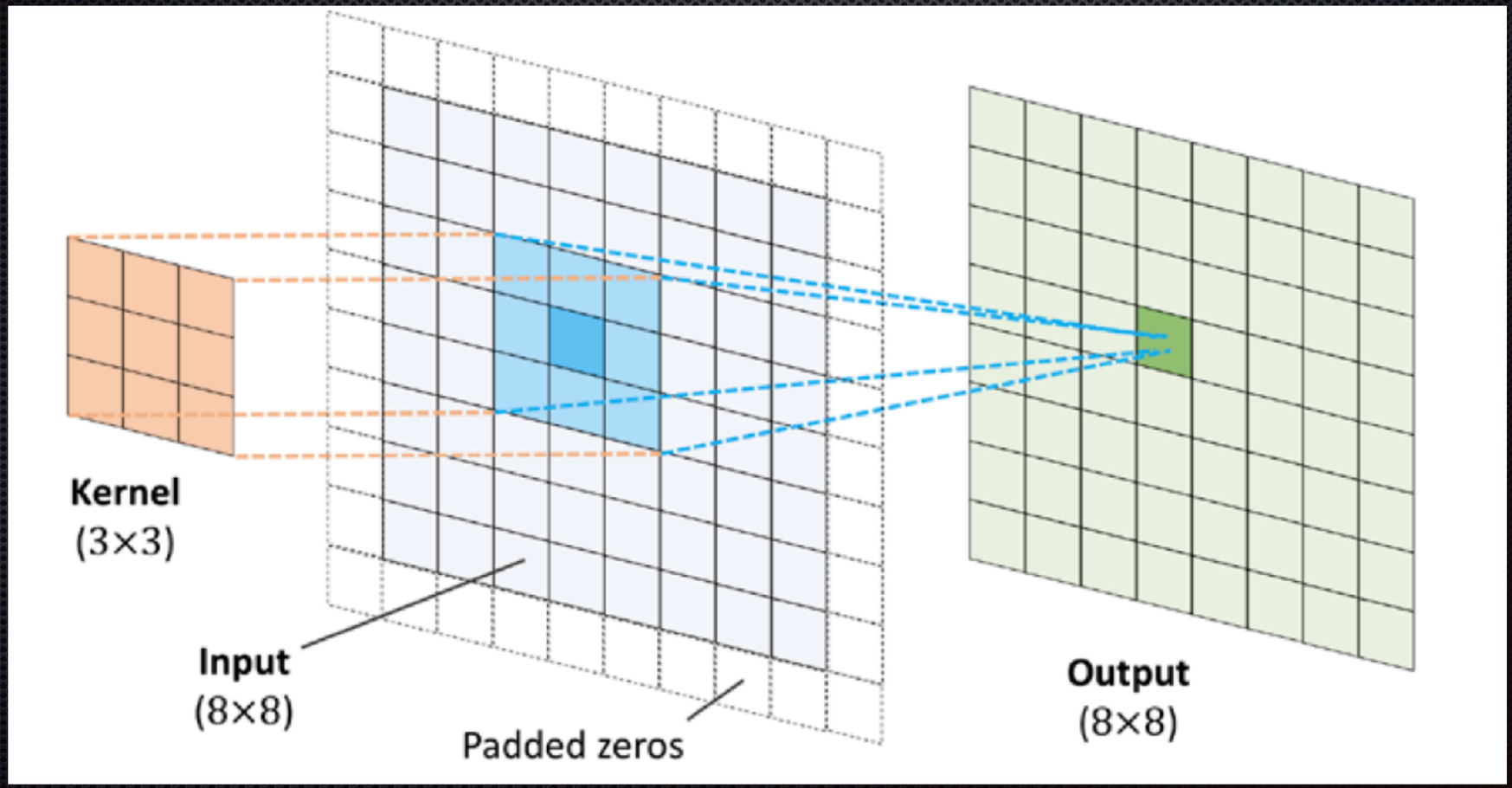
$$\text{Convolution output} = \frac{\text{Size of input vector} + 2 * \text{Padding} - \text{filter size}}{\text{Stride}} + 1$$

Please confirm....

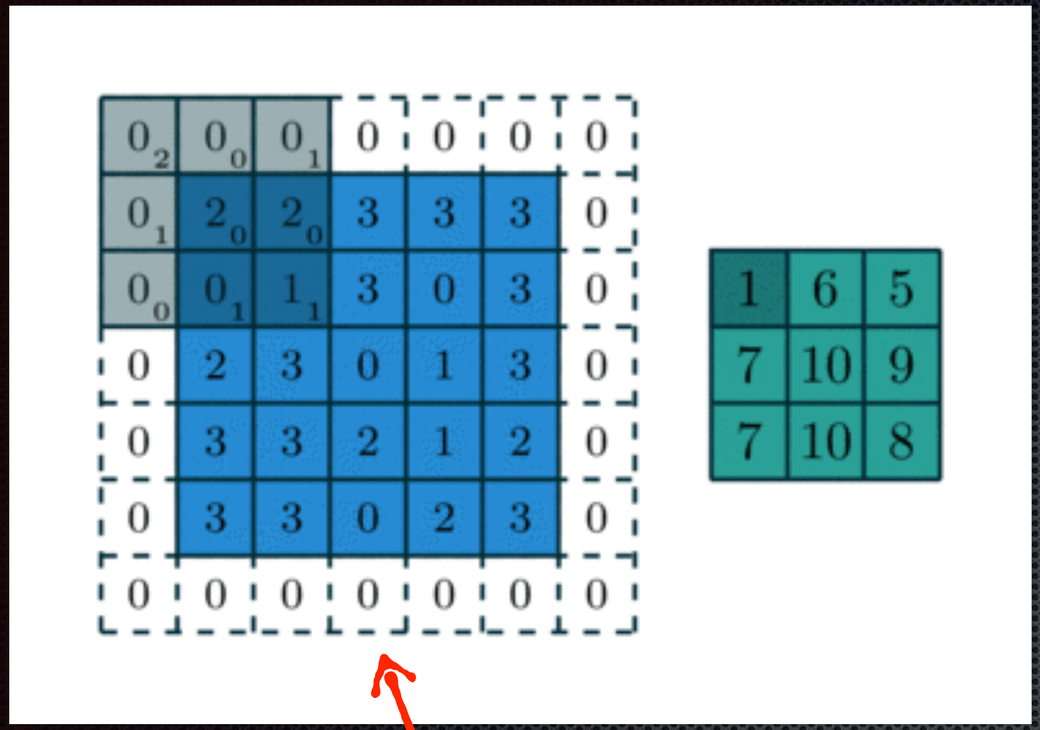


Convolution in two dimensions

$$y[i,j] = \sum_{k_1=0}^{k_1=m_1-1} \sum_{k_2=0}^{k_2=m_2-1} x_p[i+m_1-k_1, j+m_2-k_2]\omega[k_1,k_2]$$



Convolution in two dimensions



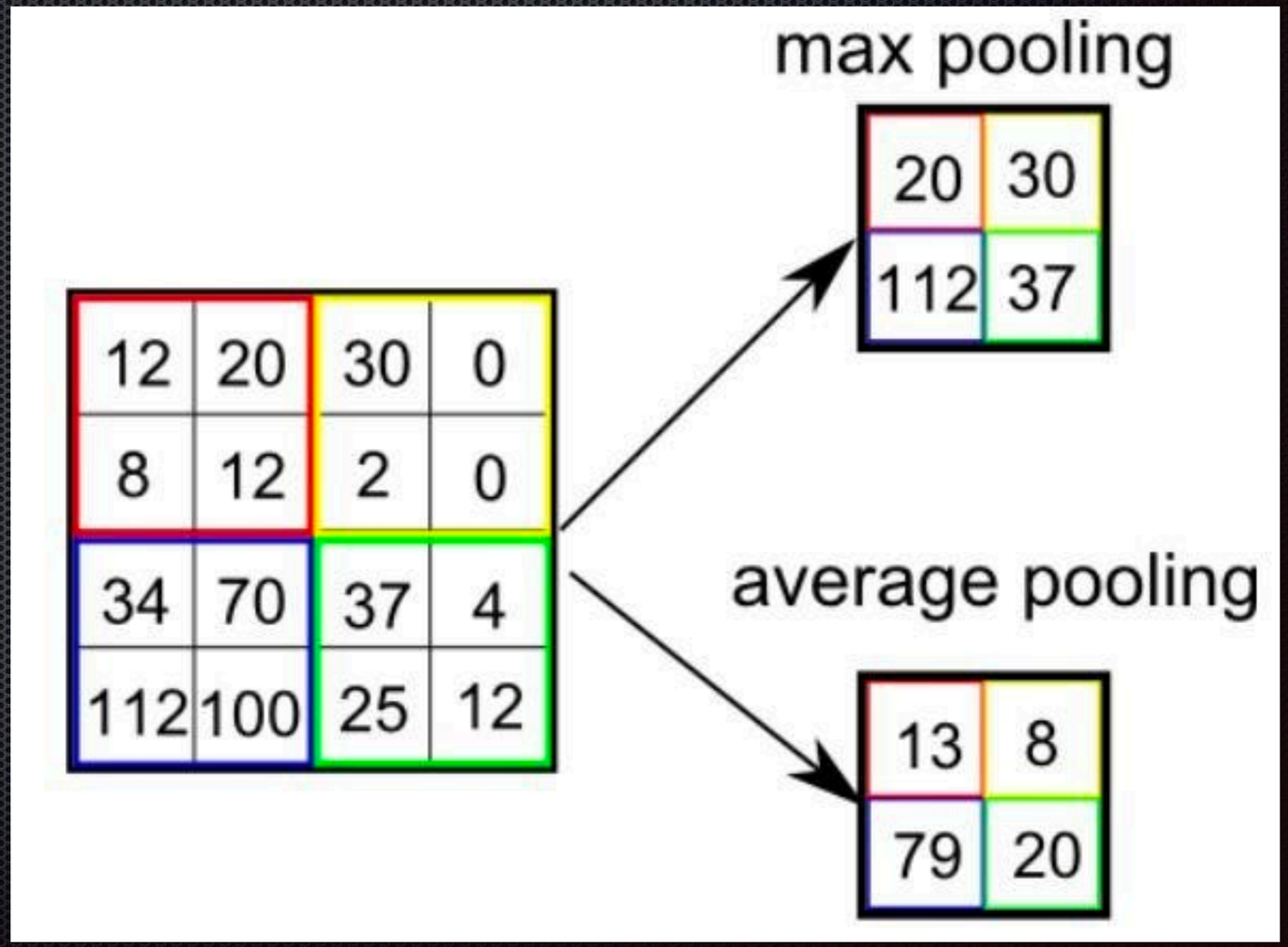
Local weights sharing

$$\text{Convolution output} = \frac{\text{Size of input vector} + 2*\text{Padding} - \text{filter size}}{\text{Stride}} + 1$$

$$\text{Convolution output} = \frac{(5,5) + 2*(1,1) - (3,3)}{(2,2)} + 1 = (3,3)$$

Pooling

Pooling reduces the high dimensions Feature space to lower dimensions space (Latent space) to capture the characteristic information only



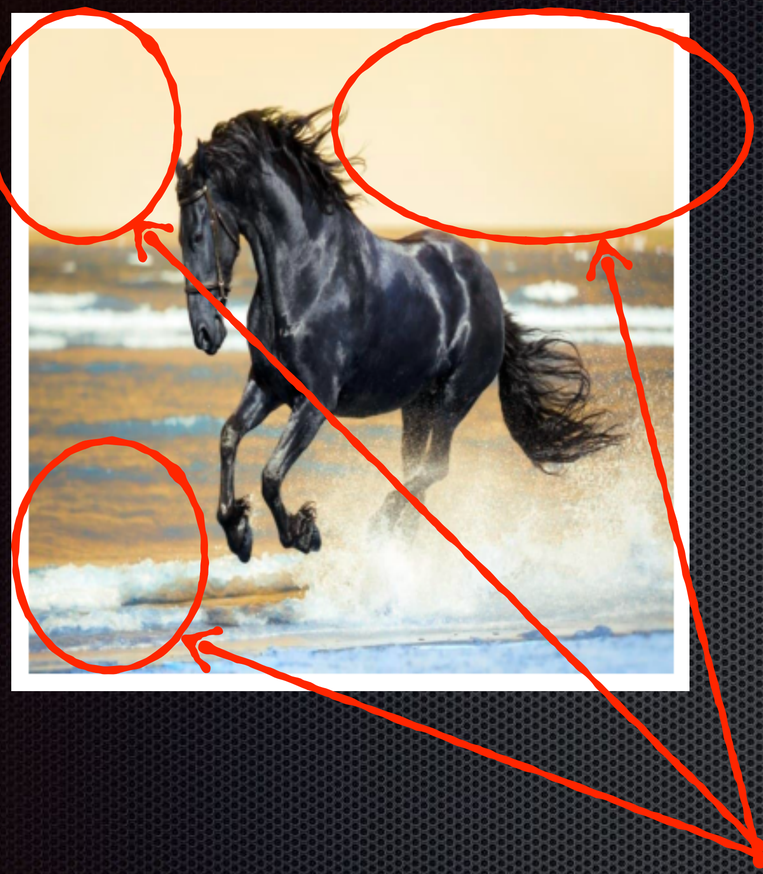
Back propagation

Back propagation is the same as in the fully connected DNN discussed in lecture 4, the difference that in CNN we deals with matrices.

I leave the deviation to the students as exercise, if you fail I am going to upload the full derivation by the end of the school

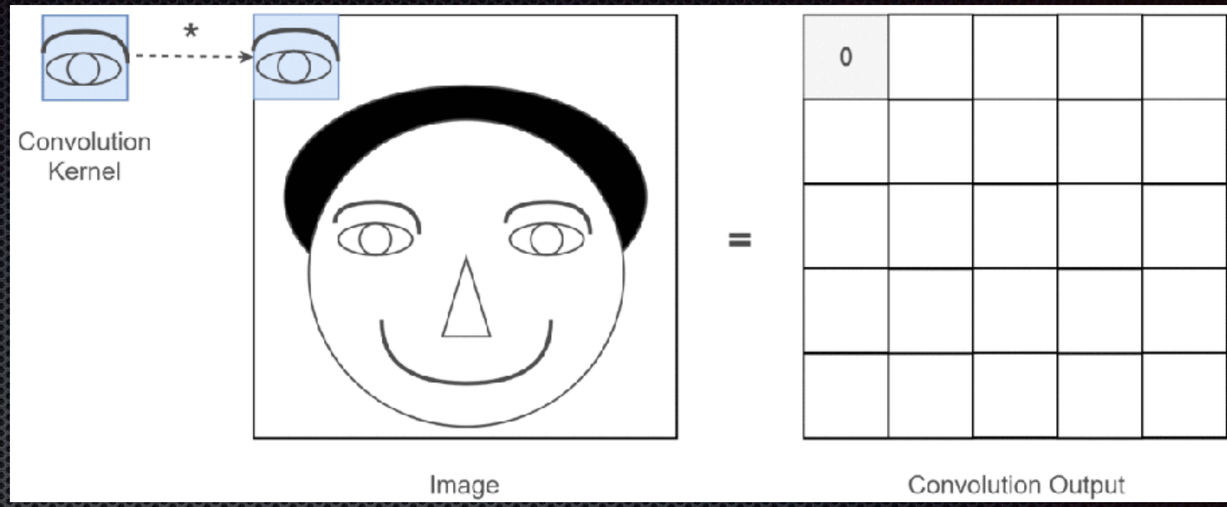
Please try it yourself and don't wait for the solution

Pooling

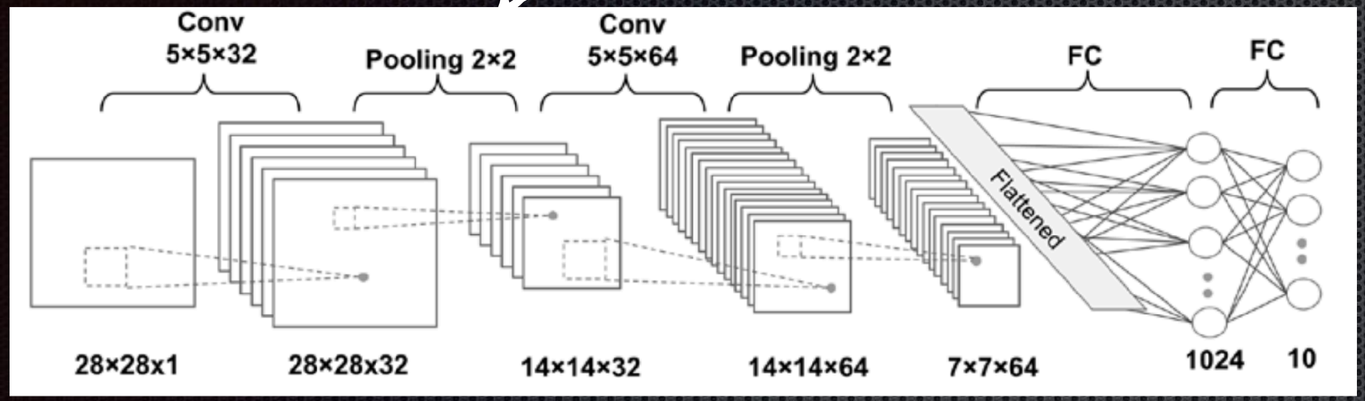


Redundant information which
can be diluted using pooling layers

All together



Lower dimensions



Using many filters to learn more local features, e.g. nose, eyes, etc

After the captured local information is mapped onto the decomposed latent space we use a fully connected layers to analyze these information and learn a global information about the input image

See lecture 4

Overfitting in CNN models

Tricks to avoid the overfitting in CNN models

- Using L1 or L2 regularization
- Early stopping
- Random dropout
- Data augmentation

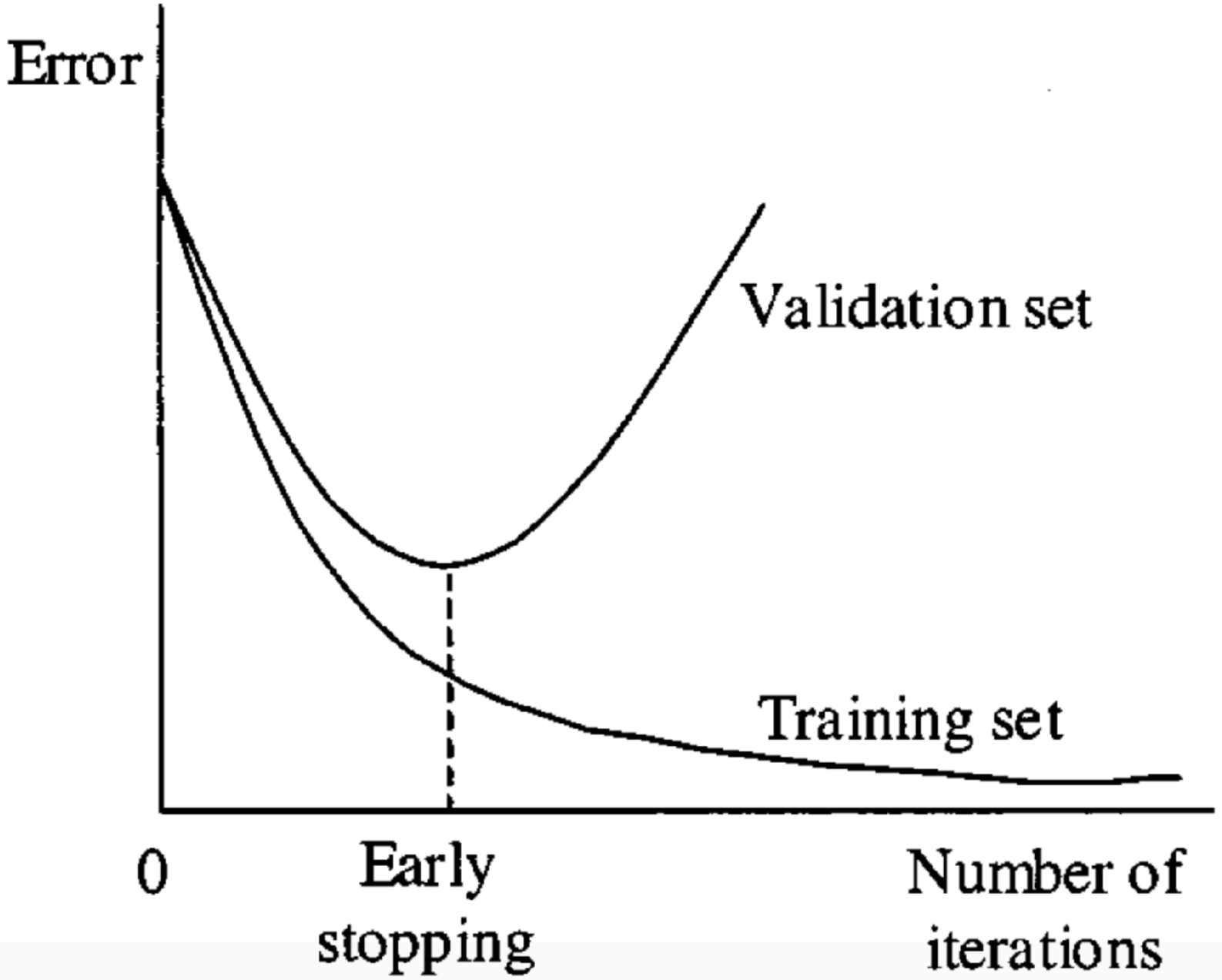
Overfitting in CNN models

Early stopping

Terminate your model training to the best fitting value

Implementation in tensorflow

tensorflow.keras.callbacks.EarlyStopping()

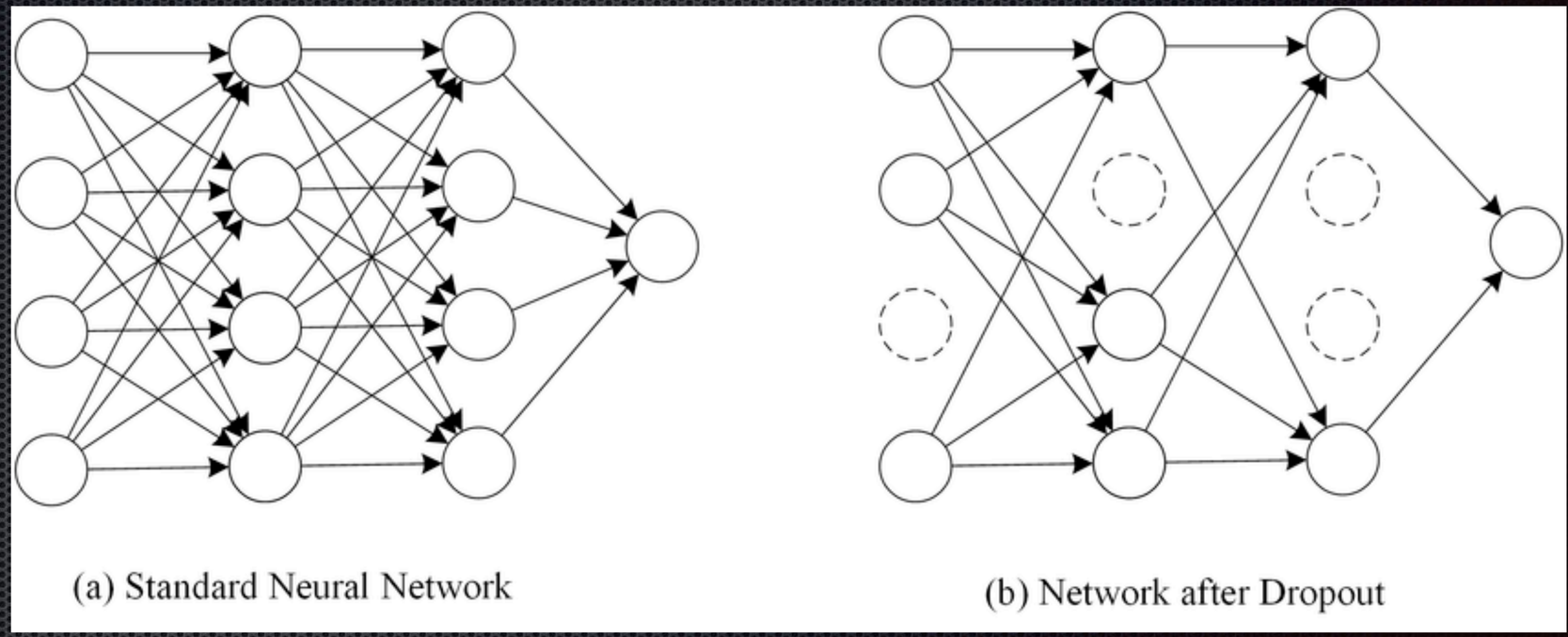


Overfitting in CNN models

Random dropout

Implementation in tensorflow

tensorflow.keras.layer.dropout(rate)



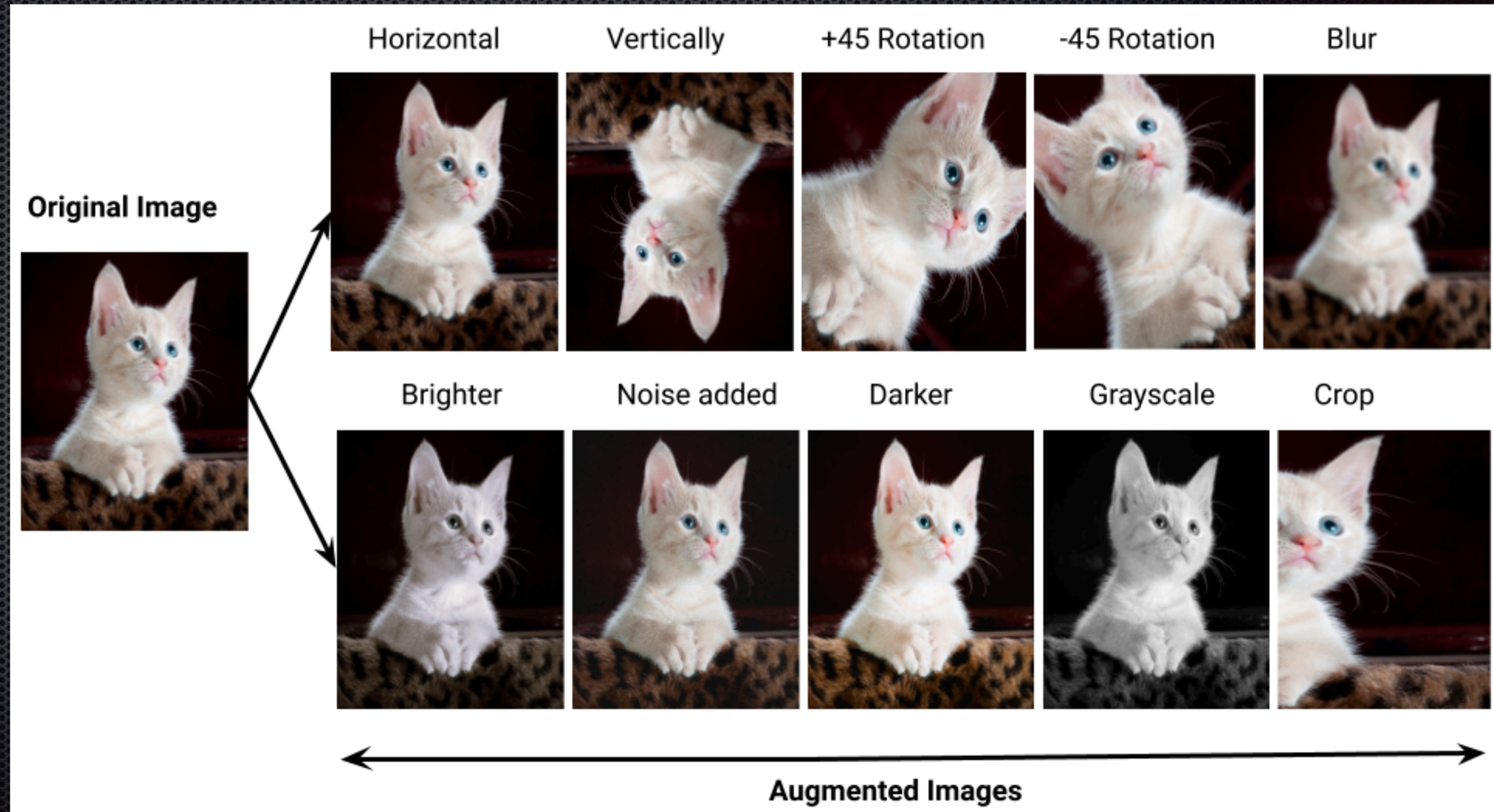
There is no theoretical proof on how the random dropout can avoid the overfitting but empirically it works fine!!

Overfitting in CNN models

Data augmentation

Adding more augmented images enhances the model generalization to the unseen test data

Data augmentation can be used to handle the imbalanced data



Example – MNIST data

Dealing with colored images

Colored images are decomposed of different color channels

Red, Green, Blue

Colored image (RGB)

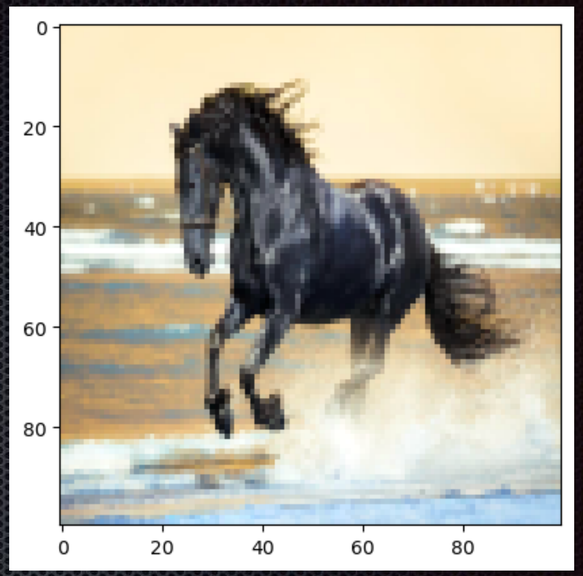
Dimensions (length, width,3)

Binary image

Dimensions (length, width,1)

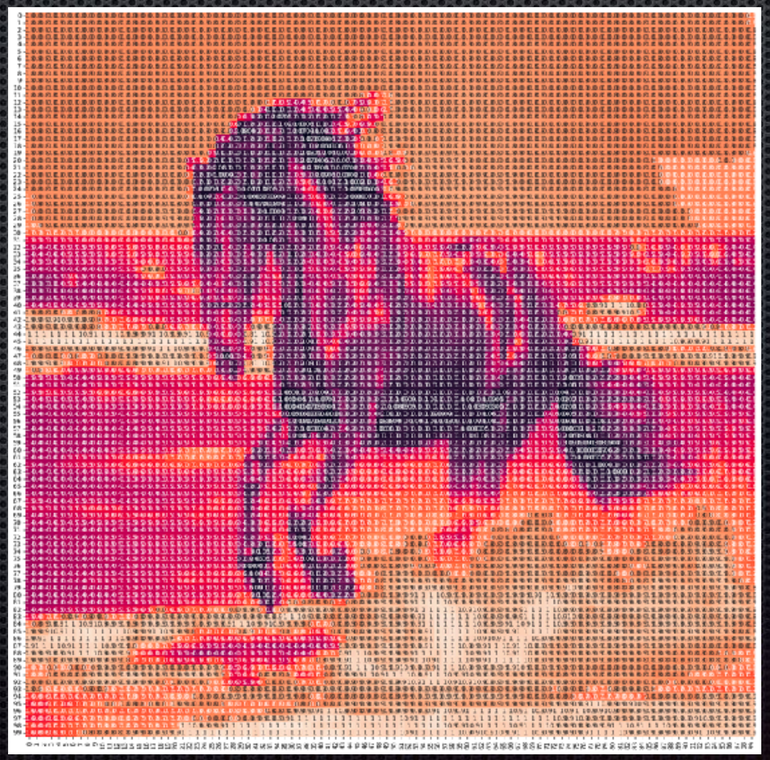
Dealing with colored images

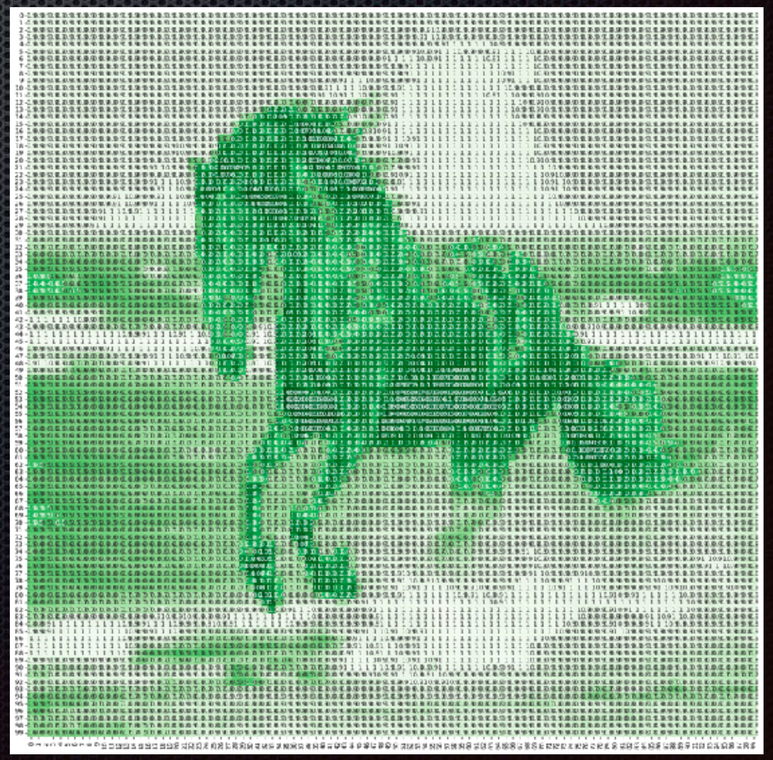
Original image
Dimension: (100,100,3)



Blue dimension
Dimension: (100,100,1)

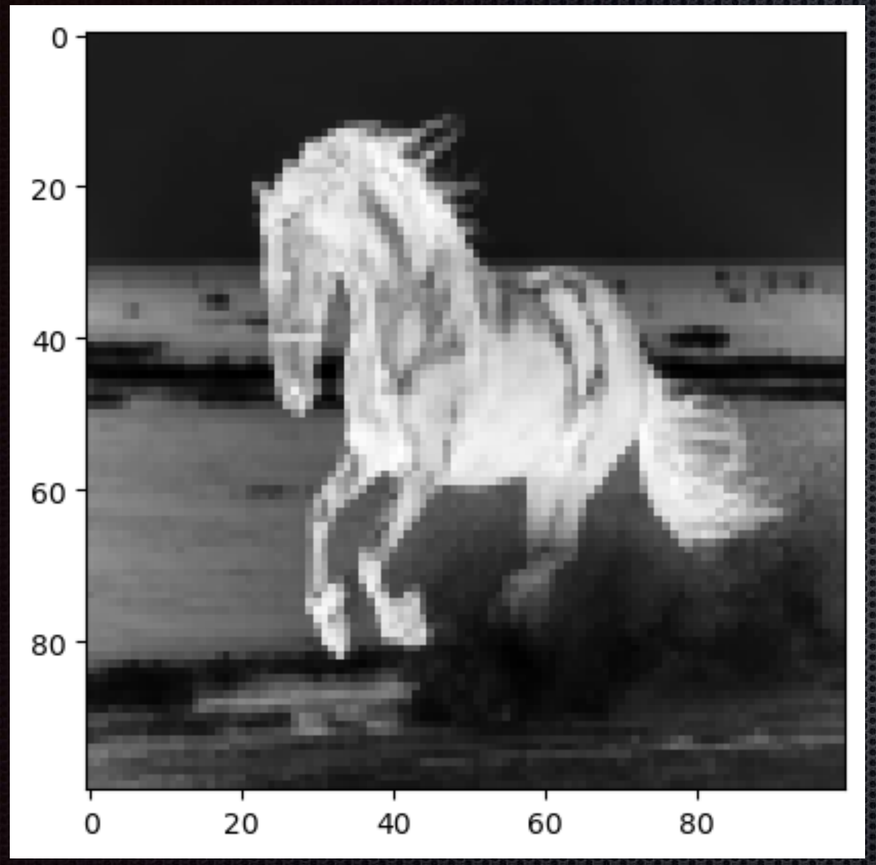
Red dimension
Dimension: (100,100,1)



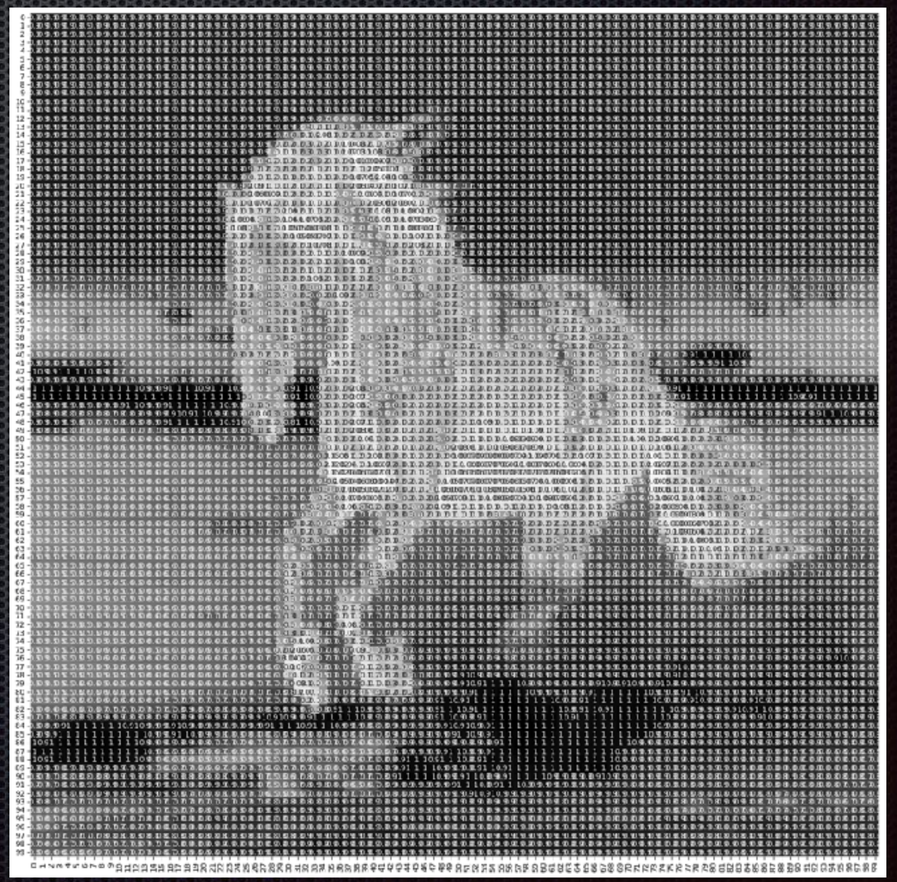


Dealing with colored images

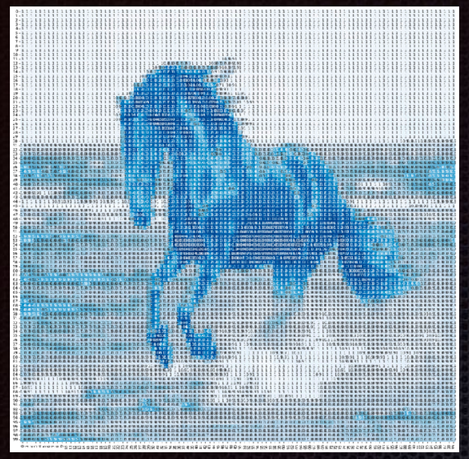
Original image
Dimension: (100,100,1)

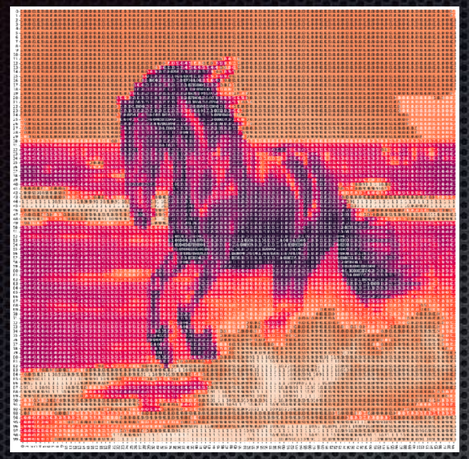


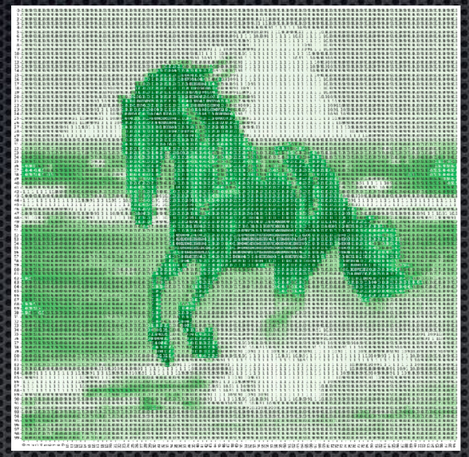
Only binary channel

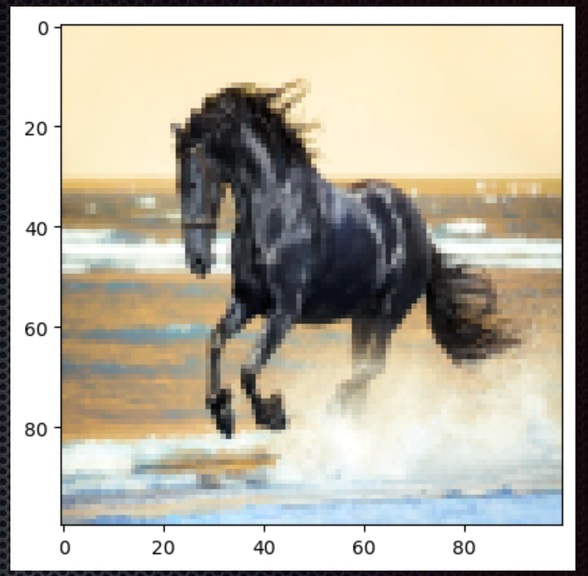


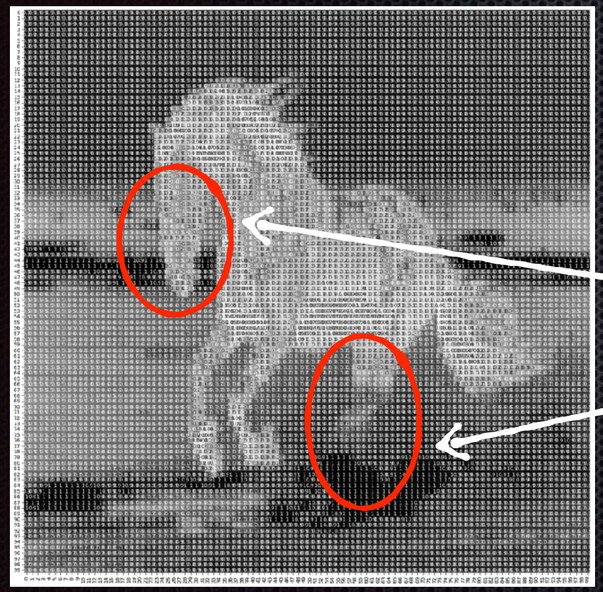
Dealing with colored images







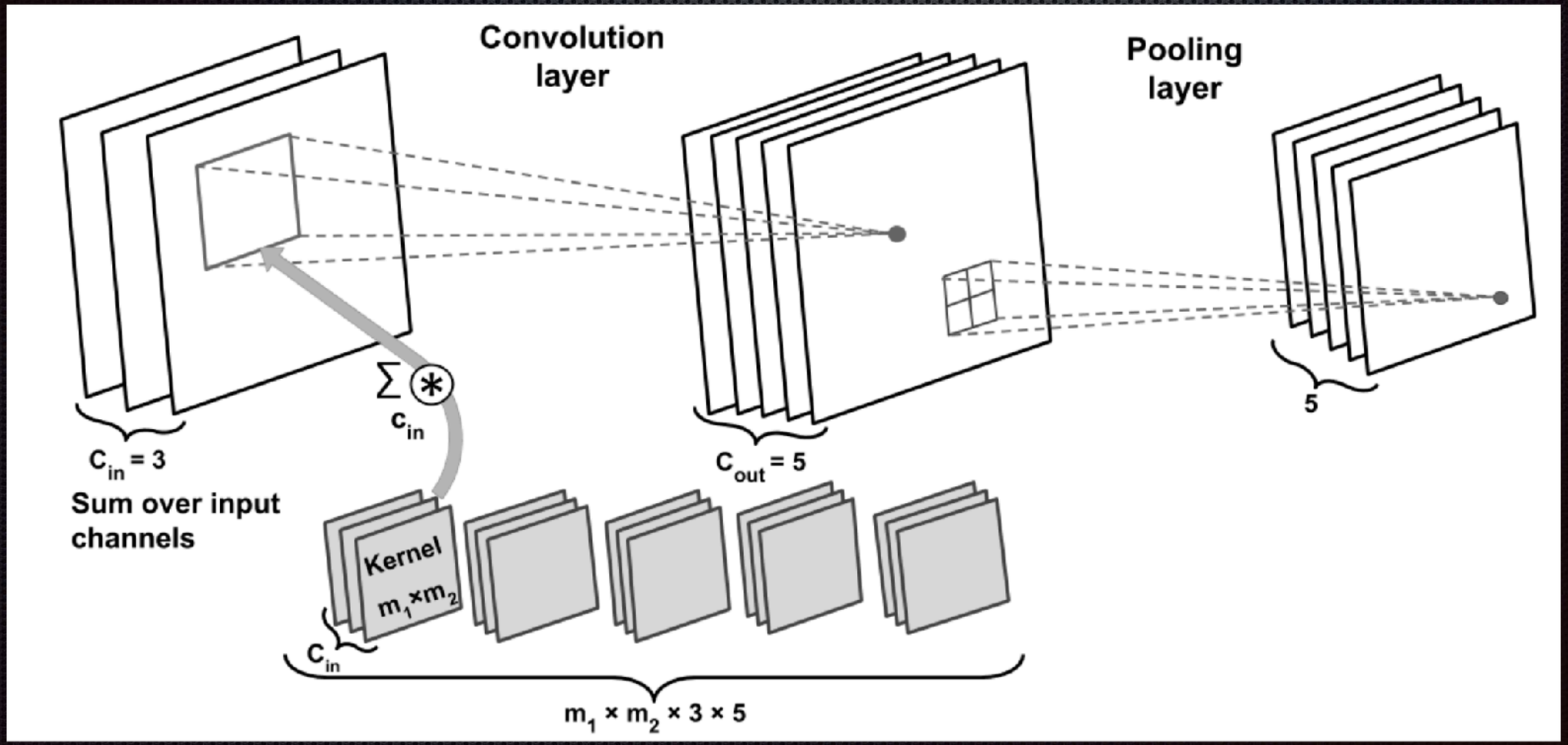




The image depth (color channels) can provide more local information than the binary image with depth 1

CNN for colored images

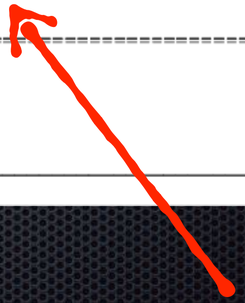
Convolution in case of color channels



Example: Smily faces

Task: Construct a CNN model to classify the smily faces

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_6 (Conv2D)            (None, 50, 50, 16)        160
conv2d_7 (Conv2D)            (None, 50, 50, 16)        2320
max_pooling2d_3 (MaxPooling  (None, 25, 25, 16)        0
2D)
dropout_5 (Dropout)          (None, 25, 25, 16)        0
conv2d_8 (Conv2D)            (None, 23, 23, 32)        4640
conv2d_9 (Conv2D)            (None, 21, 21, 32)        9248
max_pooling2d_4 (MaxPooling  (None, 10, 10, 32)        0
2D)
dropout_6 (Dropout)          (None, 10, 10, 32)        0
conv2d_10 (Conv2D)           (None, 9, 9, 64)          8256
conv2d_11 (Conv2D)           (None, 8, 8, 64)          16448
max_pooling2d_5 (MaxPooling  (None, 4, 4, 64)          0
2D)
dropout_7 (Dropout)          (None, 4, 4, 64)          0
flatten_1 (Flatten)          (None, 1024)              0
dense_3 (Dense)              (None, 128)               131200
dropout_8 (Dropout)          (None, 128)               0
dense_4 (Dense)              (None, 265)               34185
dropout_9 (Dropout)          (None, 265)               0
dense_5 (Dense)              (None, 1)                 266
=================================================================
Total params: 206,723
Trainable params: 206,723
Non-trainable params: 0
_________________________________________________________________
```

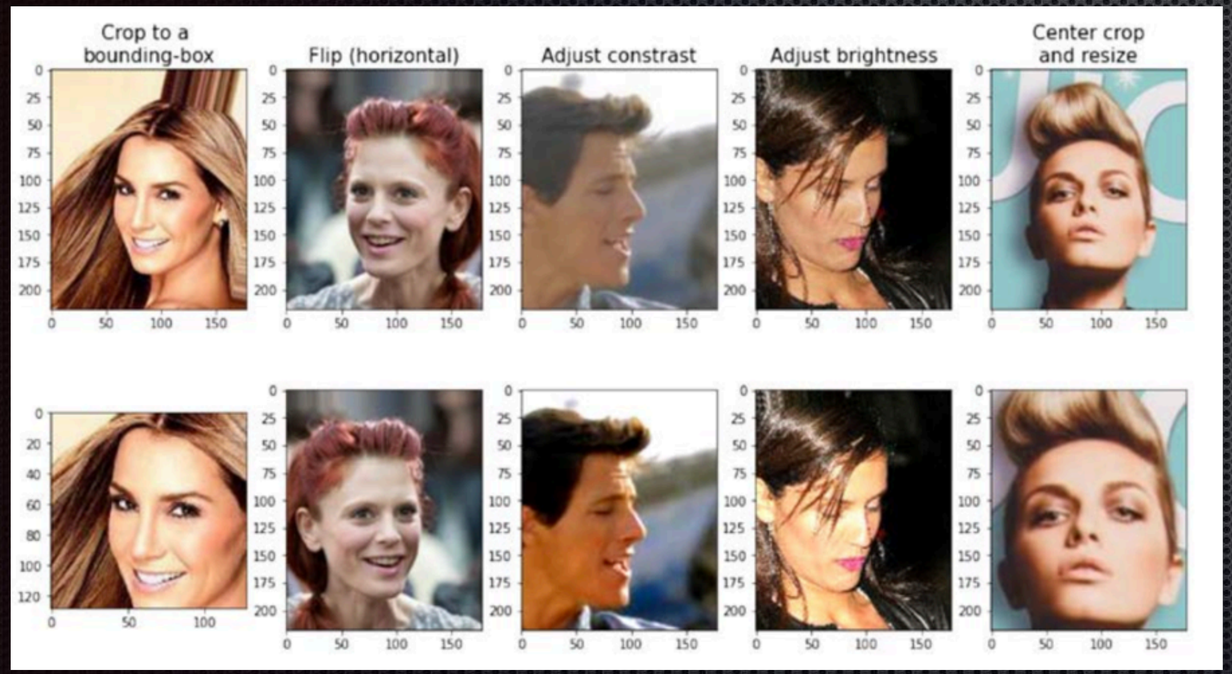


Only one output to classify either the input image has A smily face or not

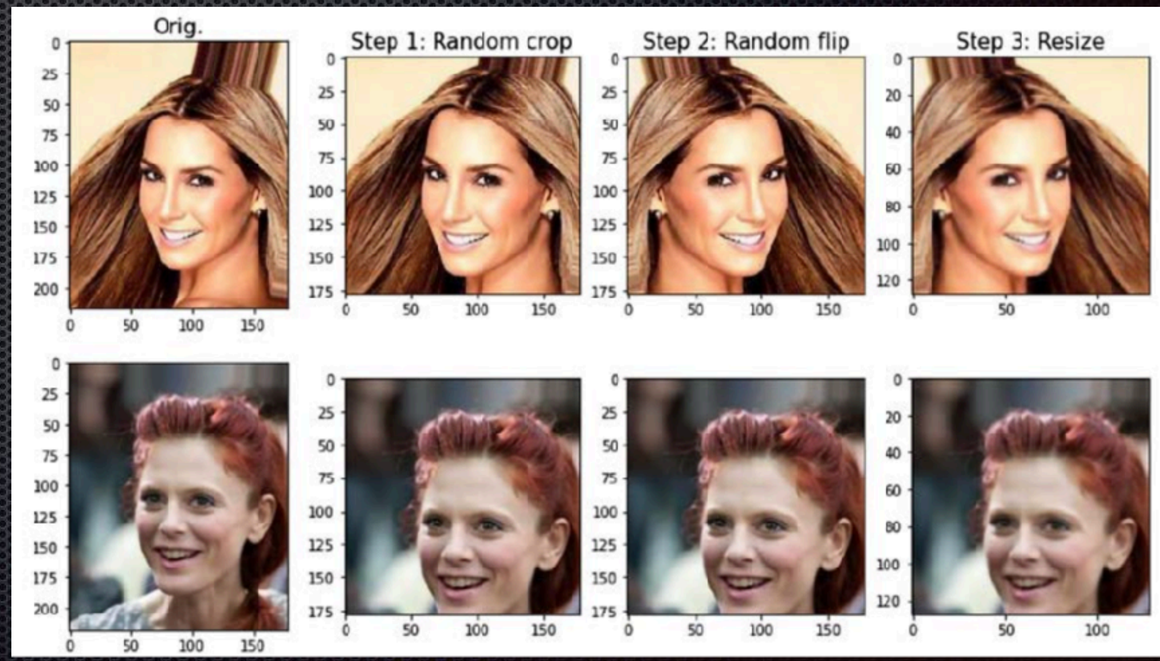
Example: Smily faces

Task: Construct a CNN model to classify the smily faces

Image preprocessing:

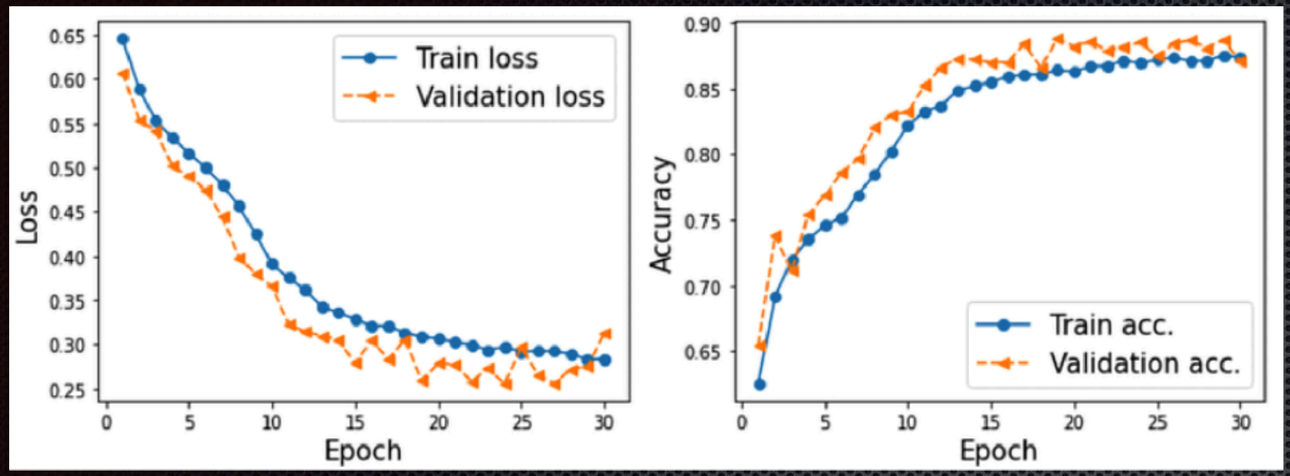


Data augmentation

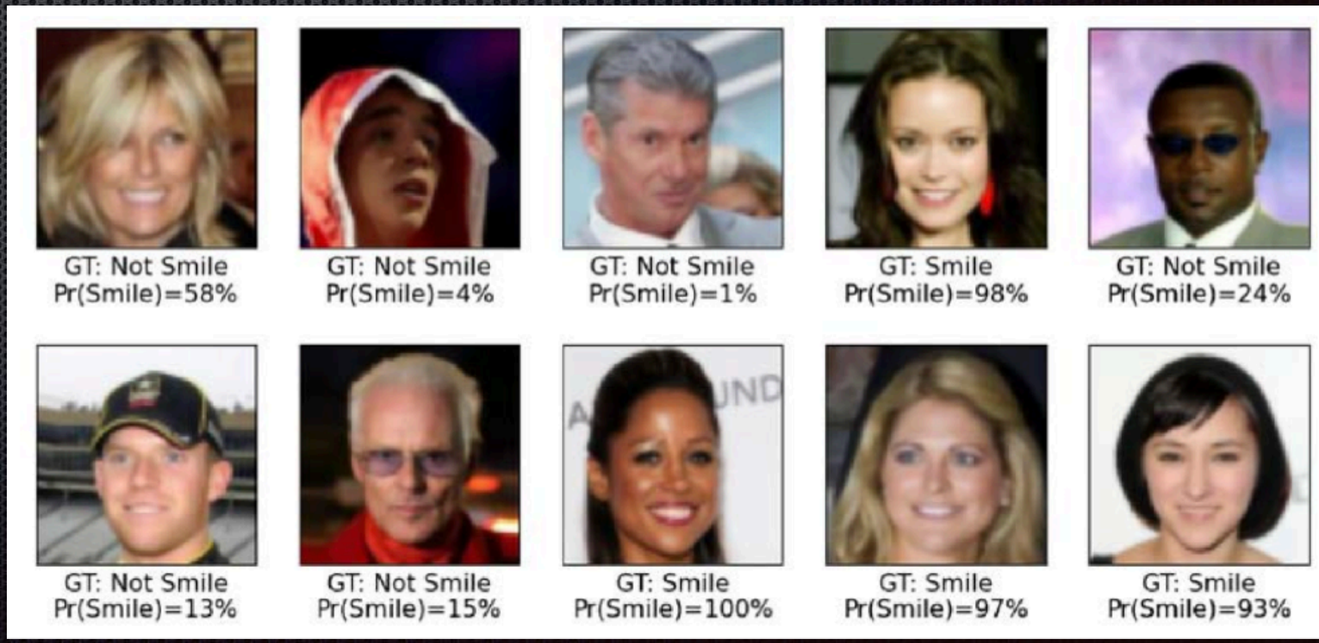


Example: Smily faces

Task: Construct a CNN model to classify the smily faces



Predictions



Transfer learning

Question: Training large data is computationally expensive, can we use one of the pre-trained models on different images sizes to our problem ?

Transfer learning

Question: Training large data is computationally expensive, can we use one of the pre-trained models on different images sizes to our problem ?

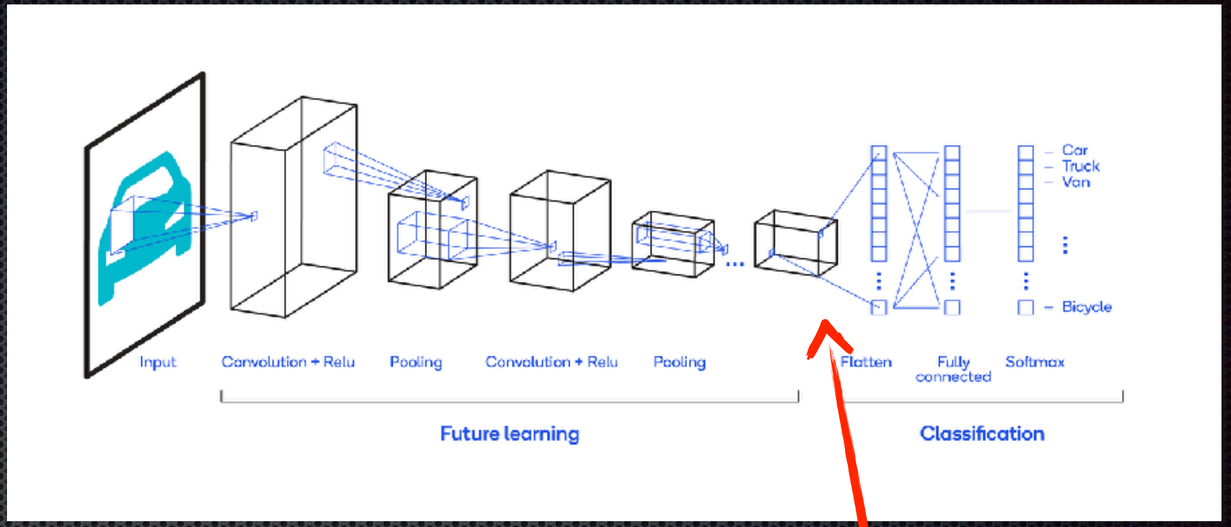
One can use the pretreated models on different images sizes by replacing the input and the output layer of the model with trainable layers and freeze all other layers. This process is called "transfer learning"

List of the pertained model by tensorflow Hub: https://tfhub.dev/

List of the pertained model by pytorch Hub: https://pytorch.org/vision/stable/models.html

Contrastive learning

Why contrastive learning ?



Information in the bottleneck is controlled by the structure of the network and thus we need an extra metric to control it.

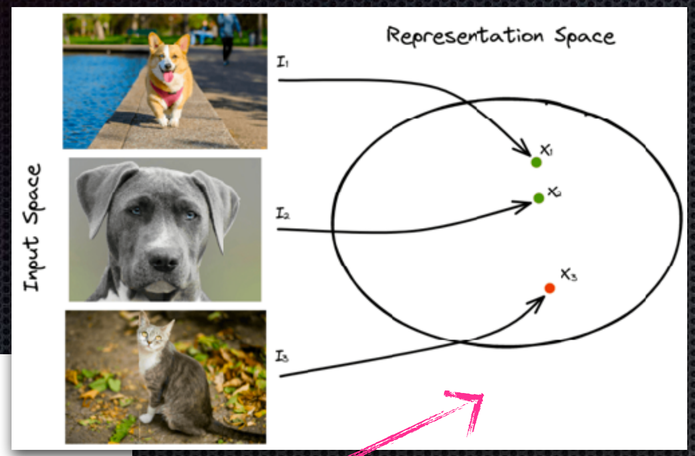
Any convolution based model works as the following:

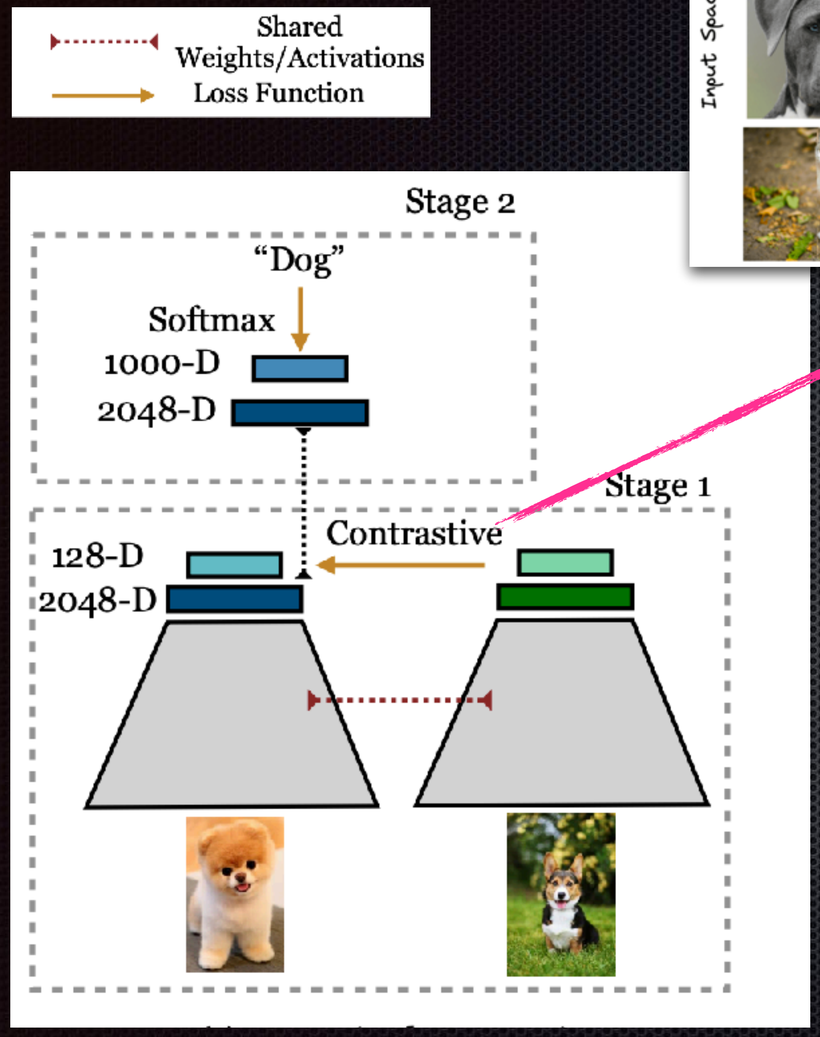
1- Feature extraction in high dimensions feature space

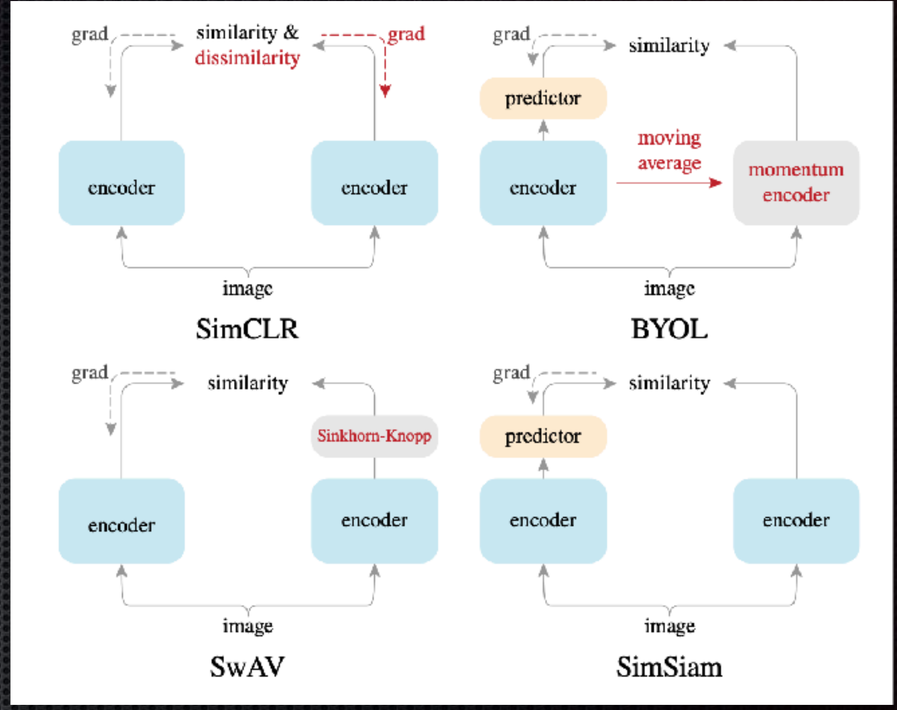
2- Decompose the extracted features into lower dimensions space (bottleneck)

3- Use FC network to analyze the information in the bottleneck

Contrastive learning



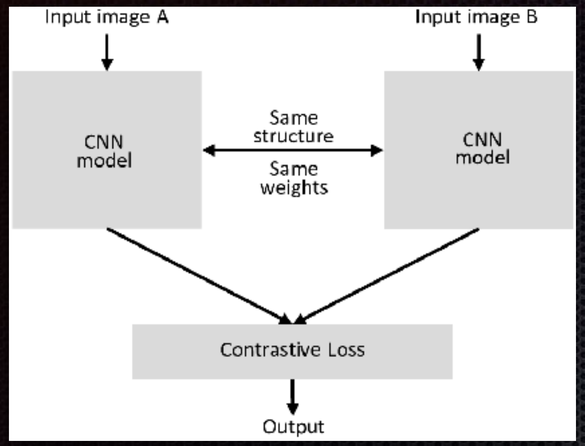




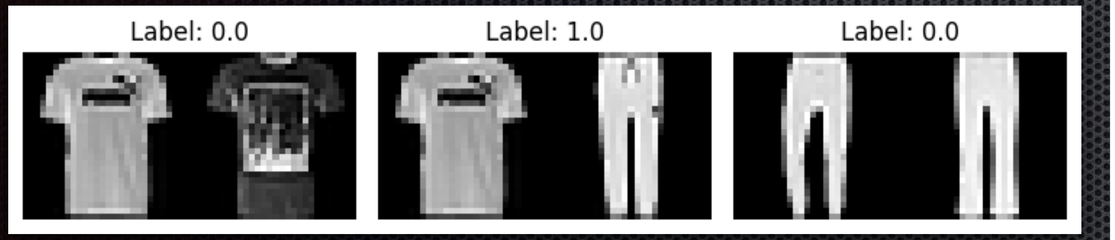
Wide list of contrastive models

Twins encoder with shared weights, two steps training with data augmentation

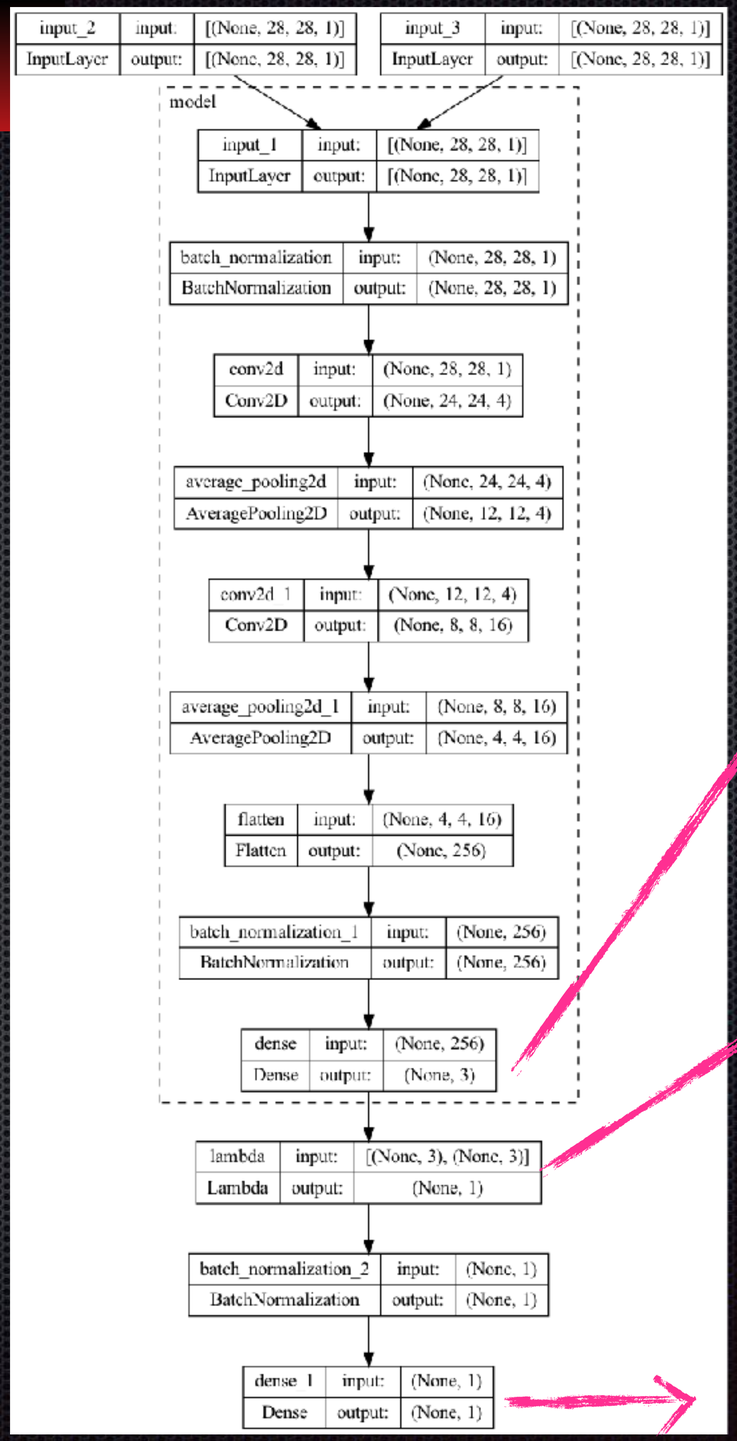
Contrastive learning



Supervised twins encoder with shared weights.



Contrastive Loss to. Minimize the Euclidean distance between similar objects and maximize the distance between different objects



Siamese bottleneck with dimension 3

Projection layer to compute the Euclidean distance

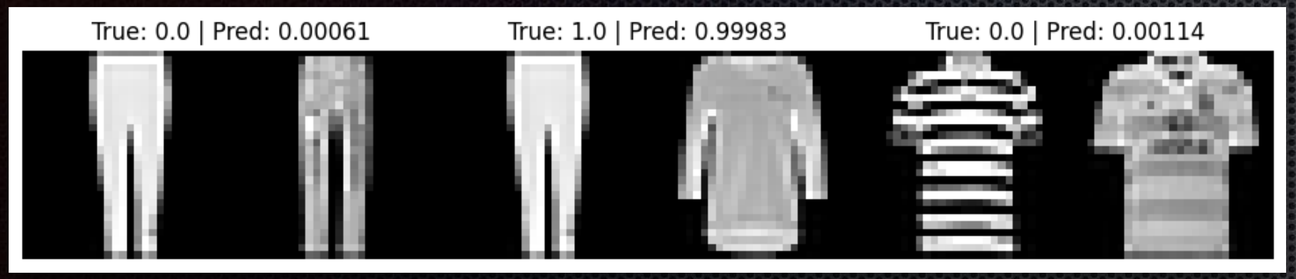
Lambda layer

Model output

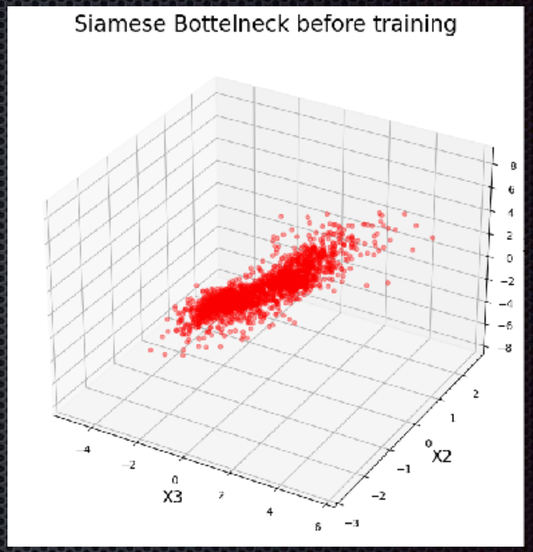
$$D = \sqrt{\sum_{i=1}^{3}(X_i - X_i^{'})^2}$$

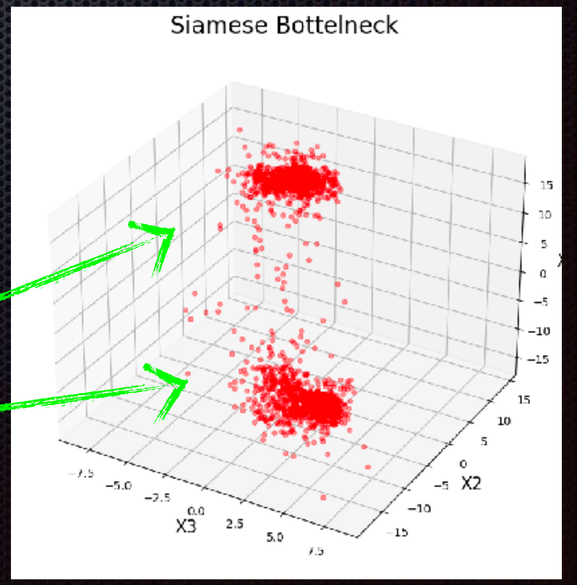
Contrastive learning

Siamese predictions



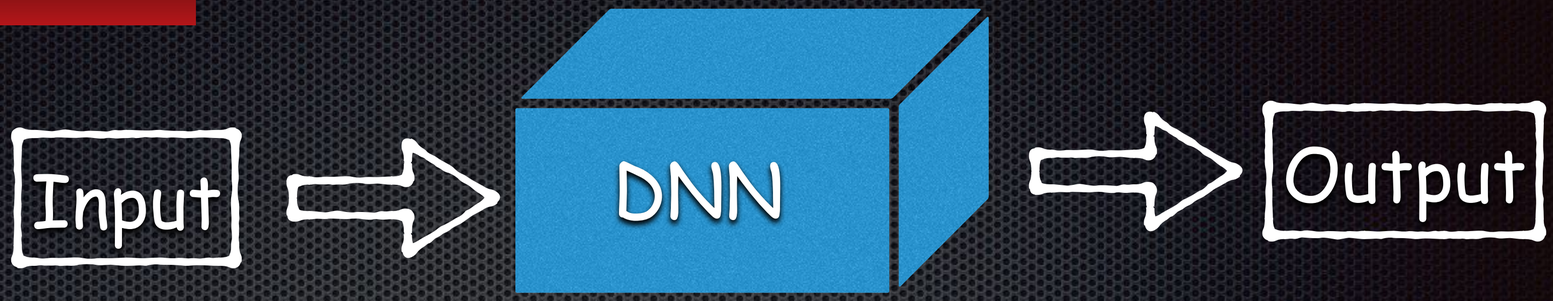
Siamese with one step training,
cannot be used for classification !!





Different images have better
representation in the model bottleneck

CKA similarity



DNN models are treated as black boxes which predictions are very hard to explain according to the learned information in the hidden layers.

CKA enables quantitative comparisons of representations within and across networks.

$$X \in \mathbb{R}^{d \times P_1} \; Y \in \mathbb{R}^{d \times P_2}$$

$$\mathrm{CKA}(M, N) = \frac{\mathrm{HSIC}(M, N)}{\sqrt{\mathrm{HSIC}(M, M)\mathrm{HSIC}(N, N)}}$$

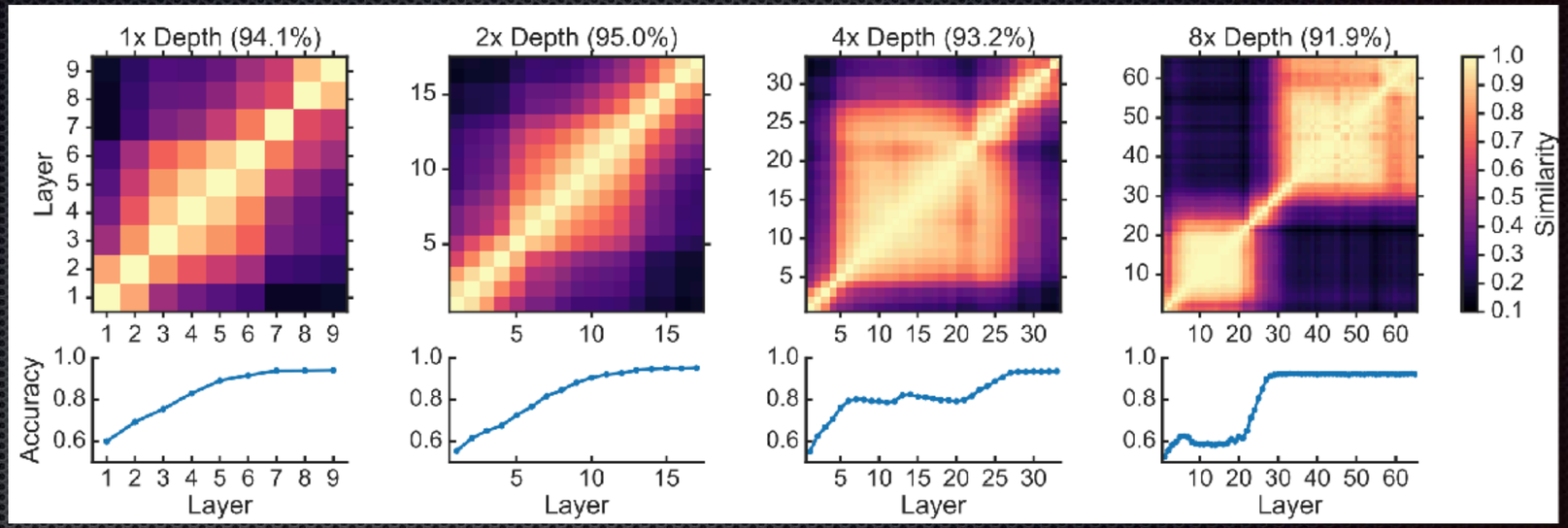
$$M = XX^\top \; N = YY^\top$$

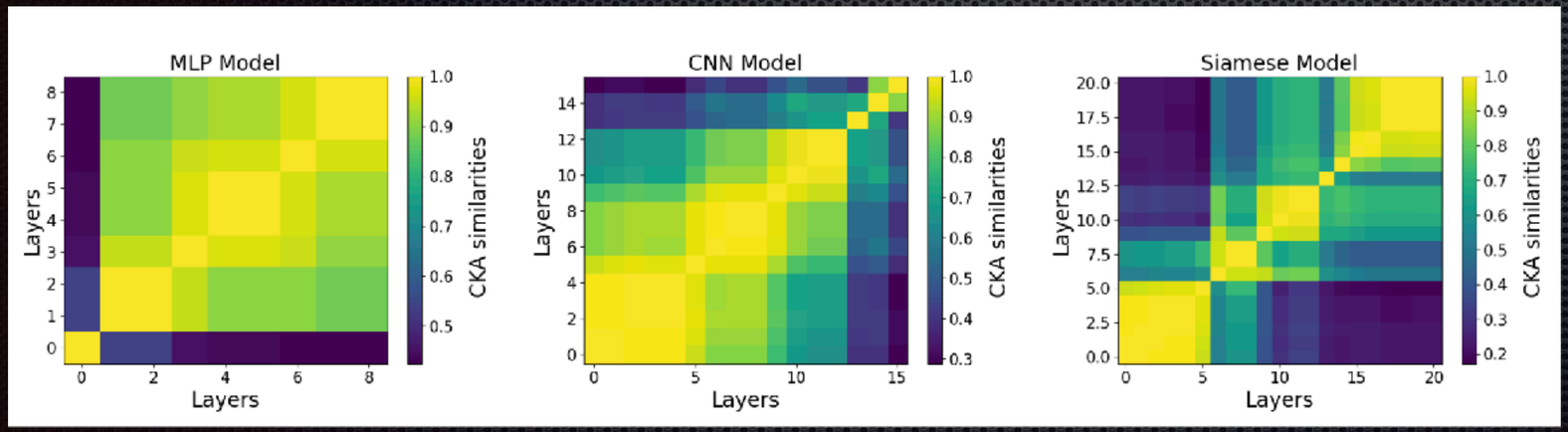
Always the Gram matrices M, N has dimensions Of (d,d)

$$\mathrm{HSIC}(M, N) = \frac{1}{(d-1)^2}\, tr(MHNH)$$

CKA similarity

arxiv:1905.00414





All models trained on CIFAR10 data sets

Image sparsity (hidden danger)

Sparsity is large number of zero pixels which are randomly distributed

1- Increases the features space with un-needed information

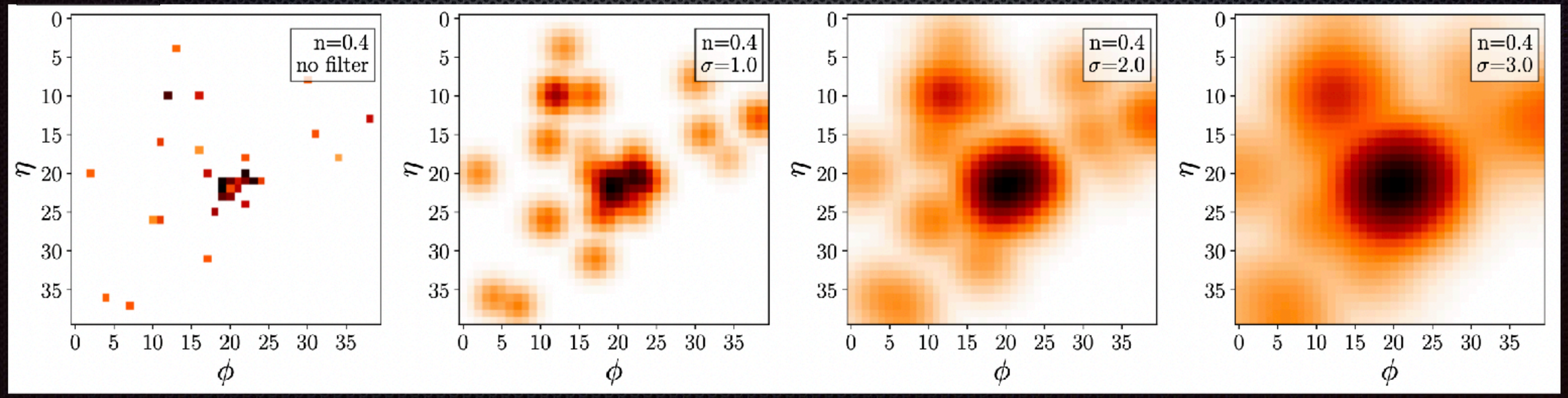
2- Increases the complexity of ML model with higher computational sources

3- Easily to overfit

4- The ML model captures redundant information which reduce its classification performance

In High energy physics we smear the momentum of the input hadrons

Tilman Plehn, etal
arXiv:2202.00686



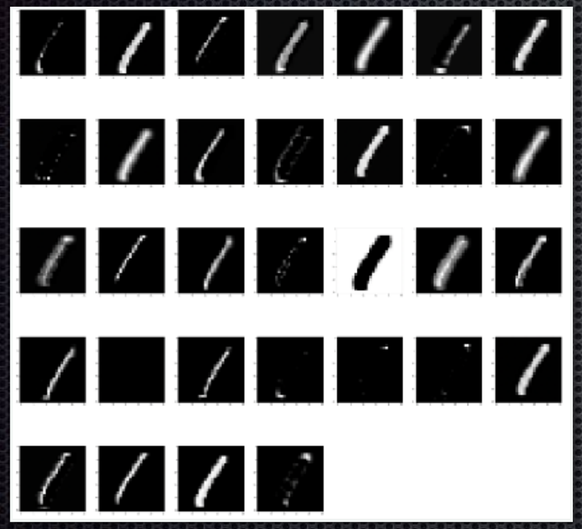
Sparse CNN

Regularize the layers activity : turning off kernels that capture low level information

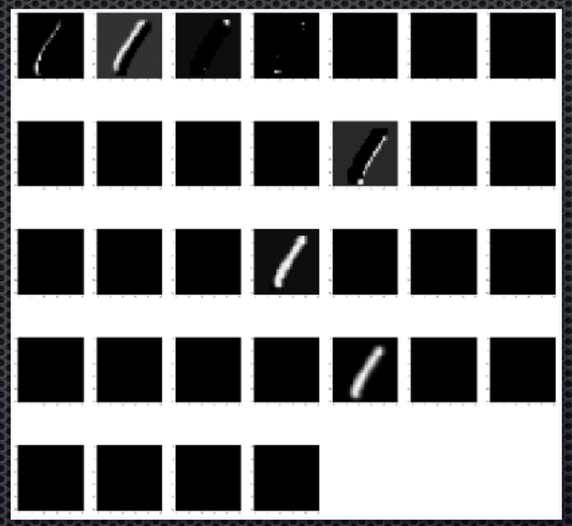
Example:

Output of the first Conv. Layer with 32 kernels after training on MNIST data set.

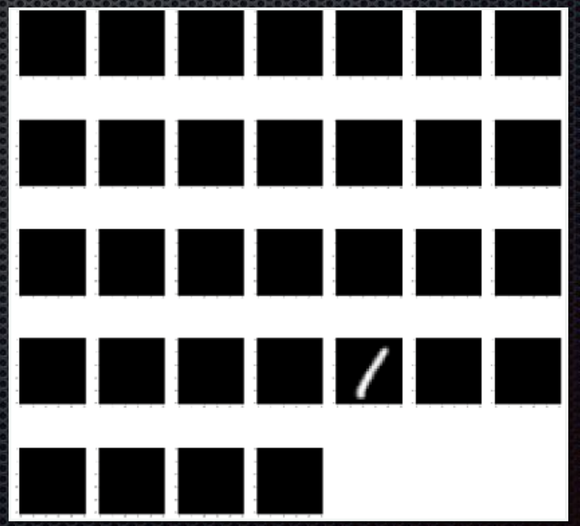
$\alpha = 0$



$\alpha = 0.01$



$\alpha = 0.5$



If the kernels of the Conv layers capture redundant features, all neurons in the flatten layer will be fired with mild weights which reduce the classification performance

To be continued…