

Machine Learning crash course

(Part-5)

Ahmed Hammad

*Convolution based neural network for
image recognition*

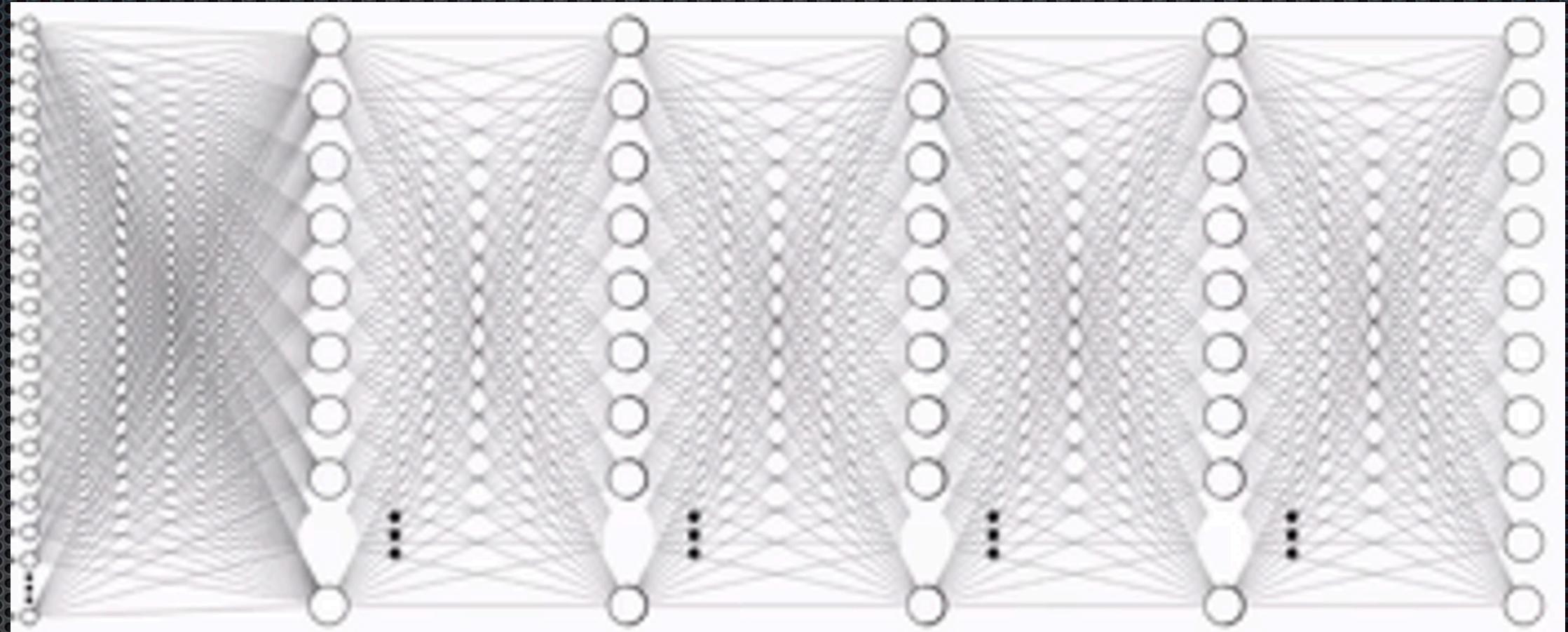
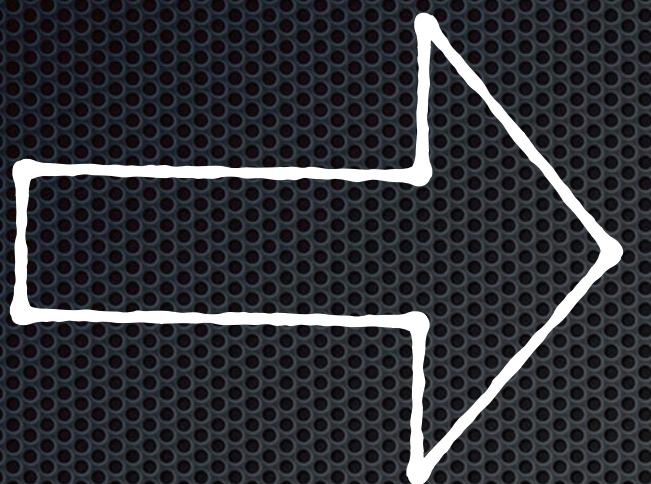
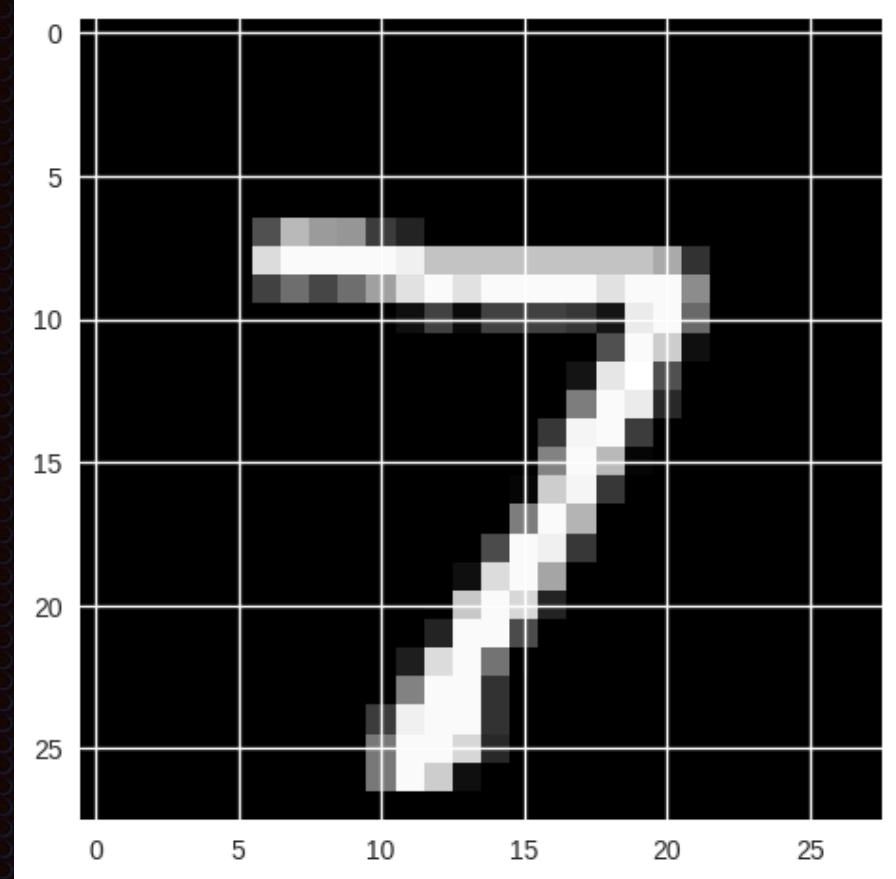
Introduction

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

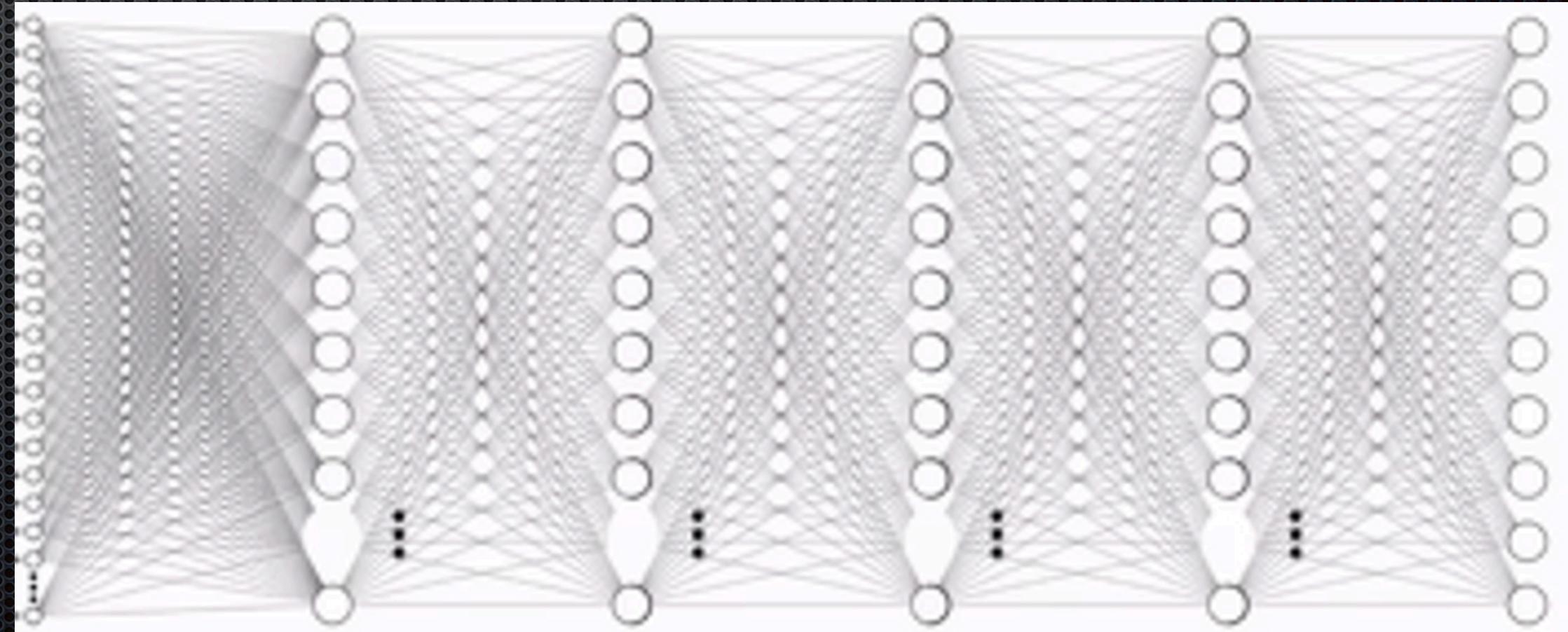
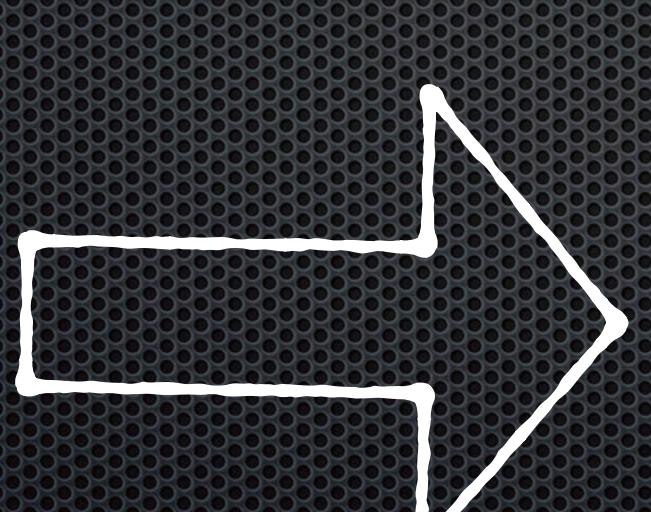
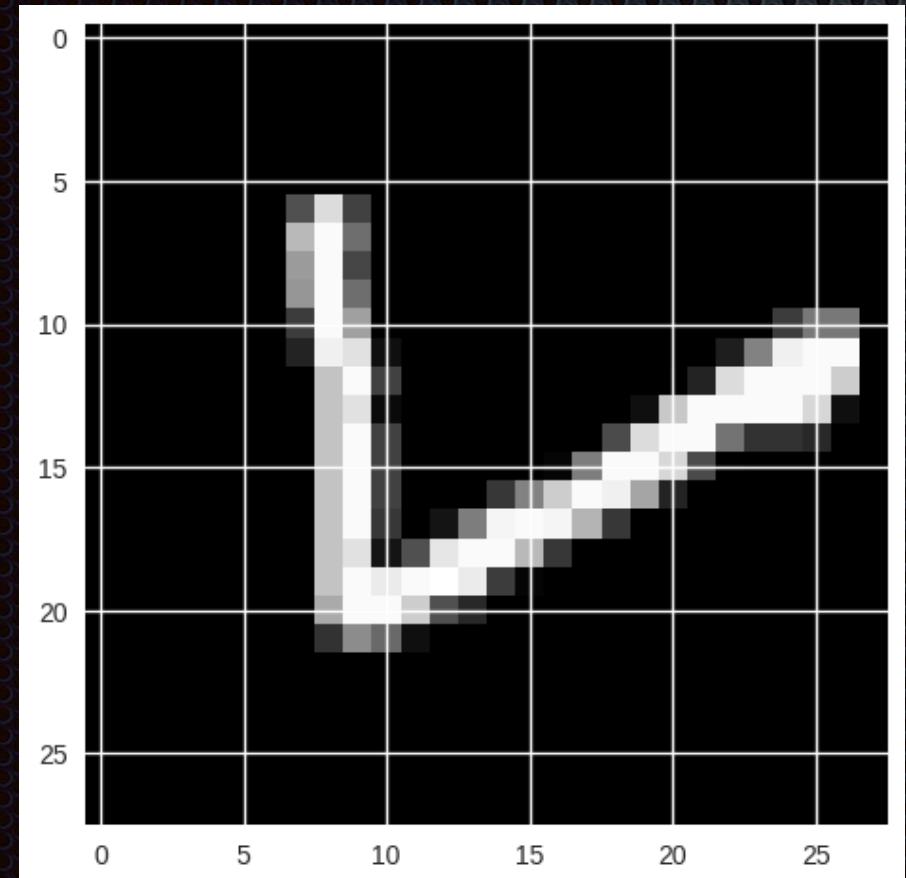
Introduction

Trained DNN from lecture 4

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



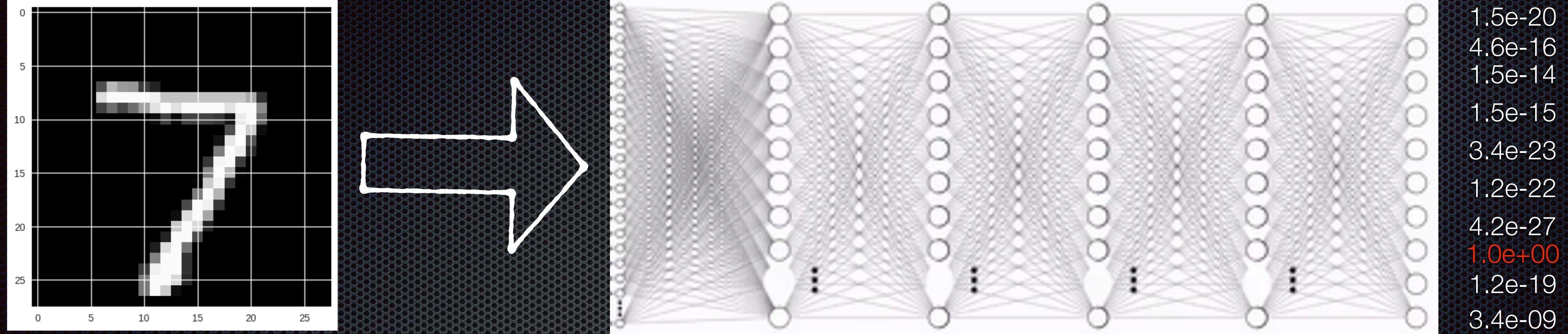
$1.5e-20$
 $4.6e-16$
 $1.5e-14$
 $1.5e-15$
 $3.4e-23$
 $1.2e-22$
 $4.2e-27$
 $1.0e+00$
 $1.2e-19$
 $3.4e-09$



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

Introduction

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



As a fully connected model

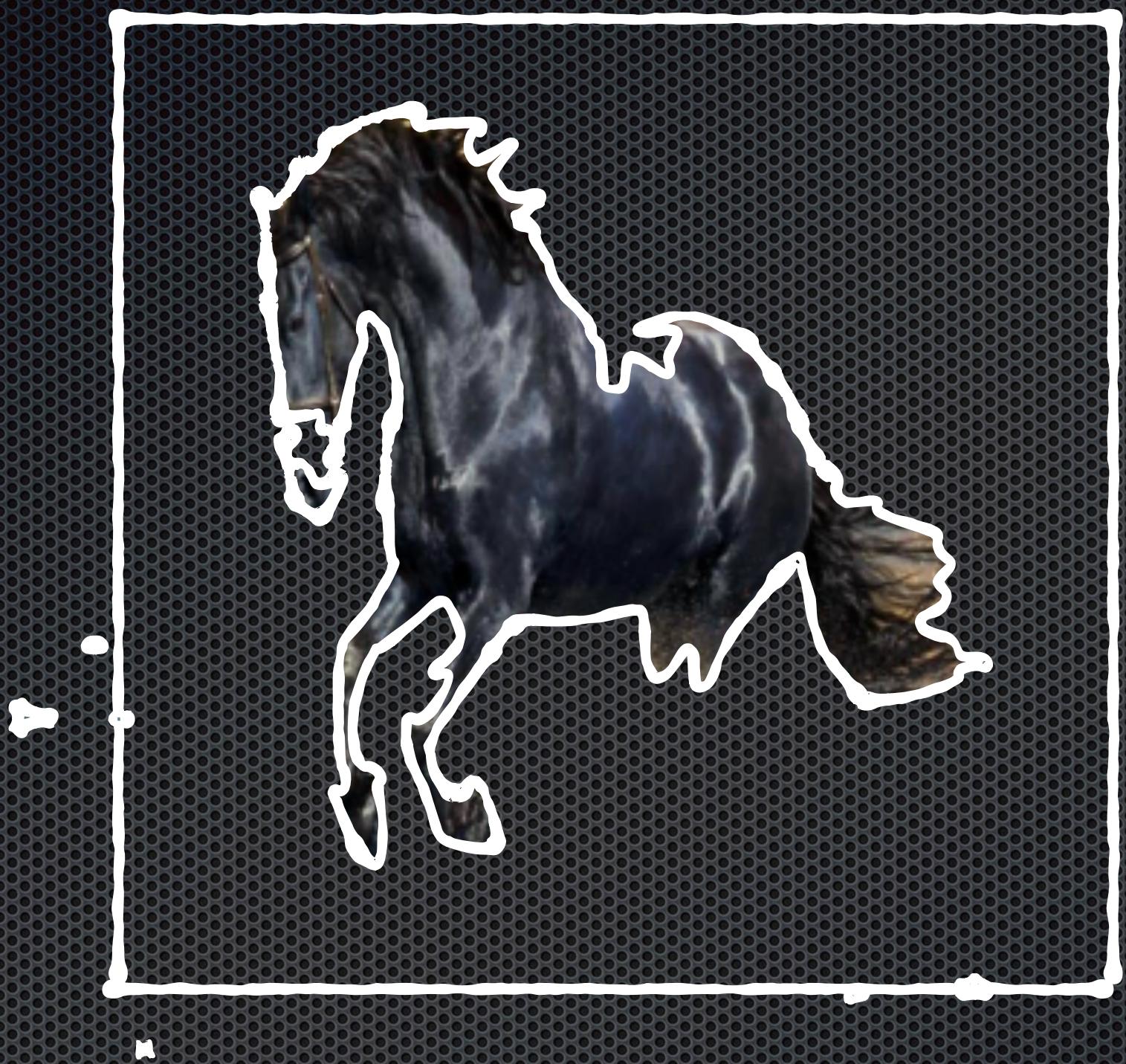
$$\text{Neuron Output} = \text{Activation function} \left(\sum_m \omega_m x_m + b \right)$$

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

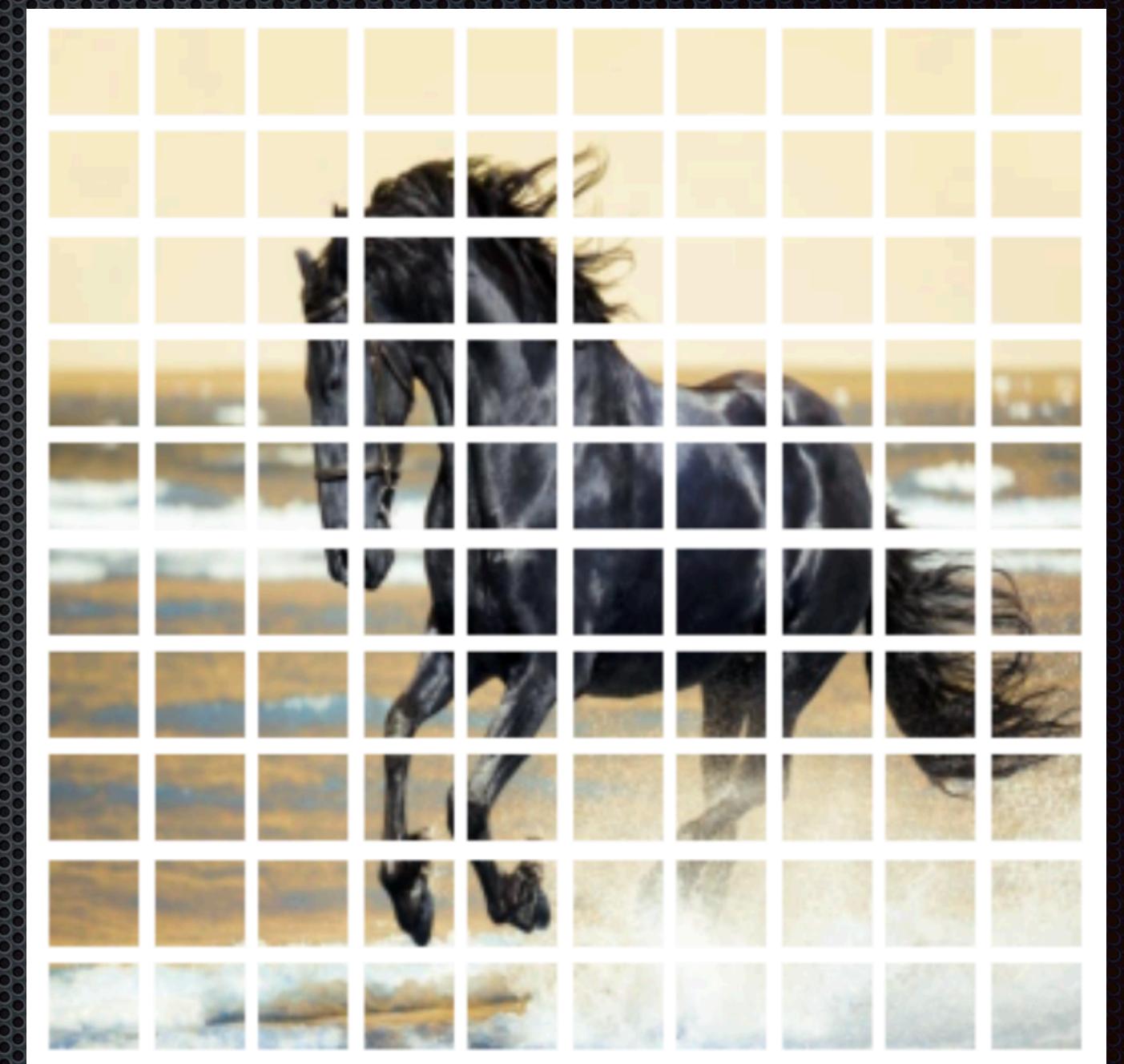
Introduction



Original image



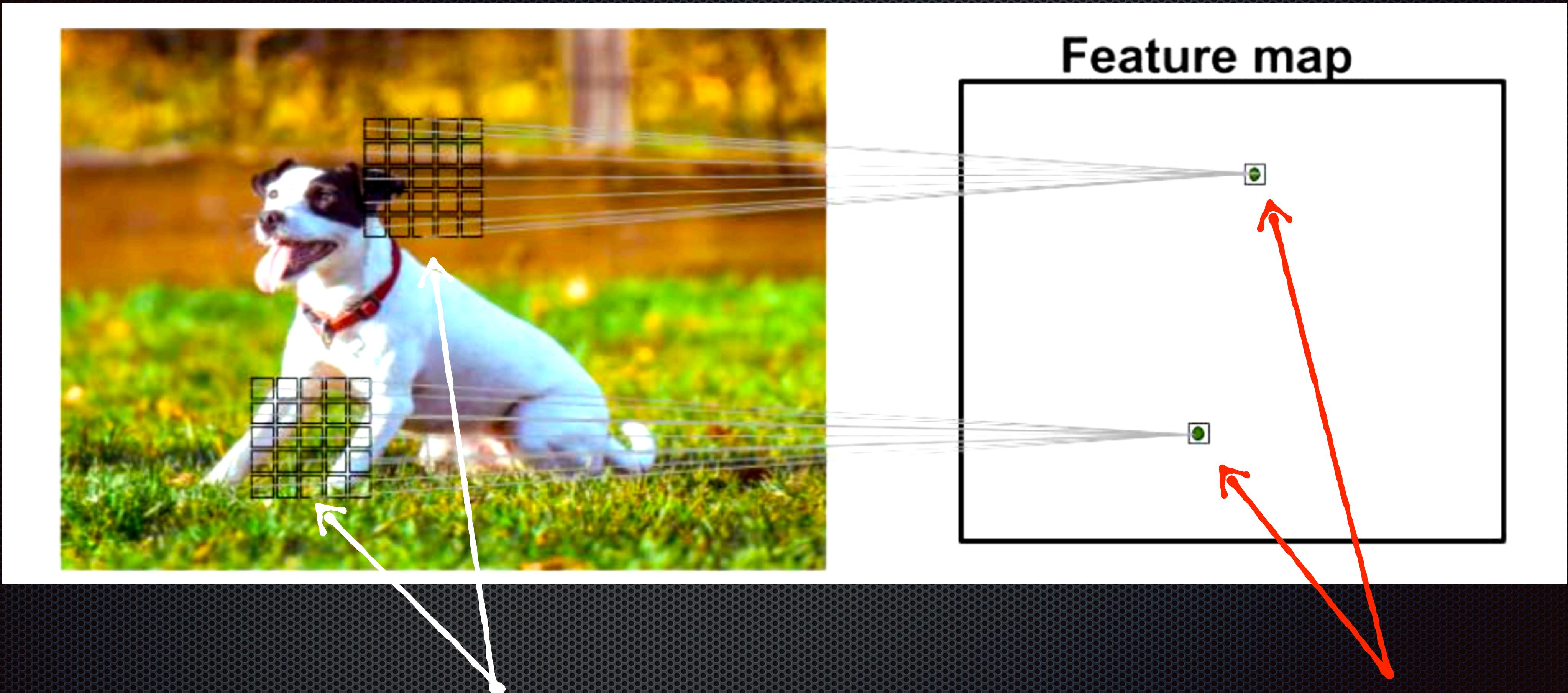
Learned global
information



Learned local
information

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

Introduction



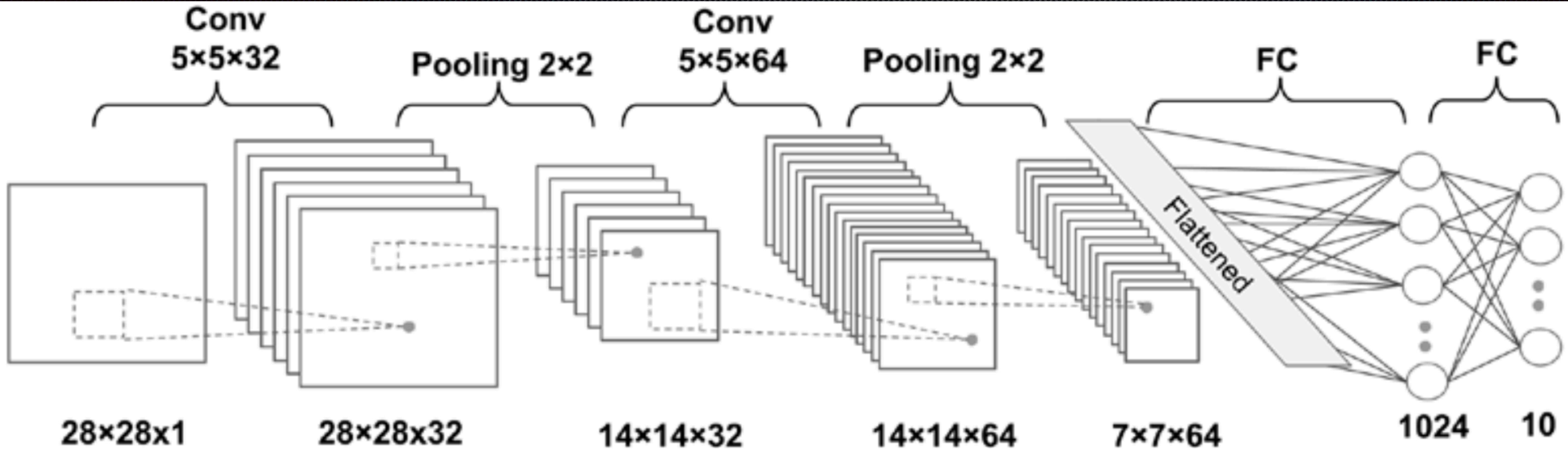
Filters to capture the **local features** of the images

Filters share their weights internally but not with other filters

Disconnected
Patches

Introduction

Example of deep CNN model



CNN = Convolution layers + Fully connected layers

How does the CNN work in detail ?

Convolution kernels

1D convolution

$$y = x * \omega = y[i] = \sum_{k=0}^K x[i-k] \omega[k]$$

Input $x[i]$

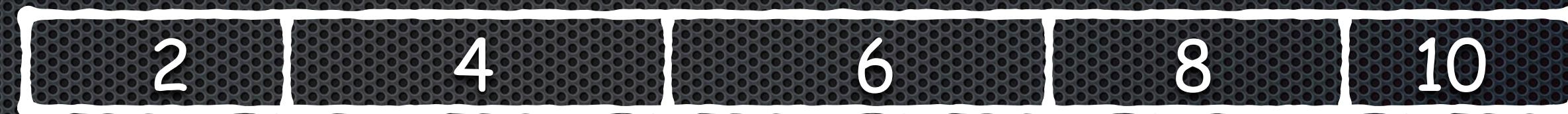


Dimension
(1,5)

Filter $\omega[k = 0]$

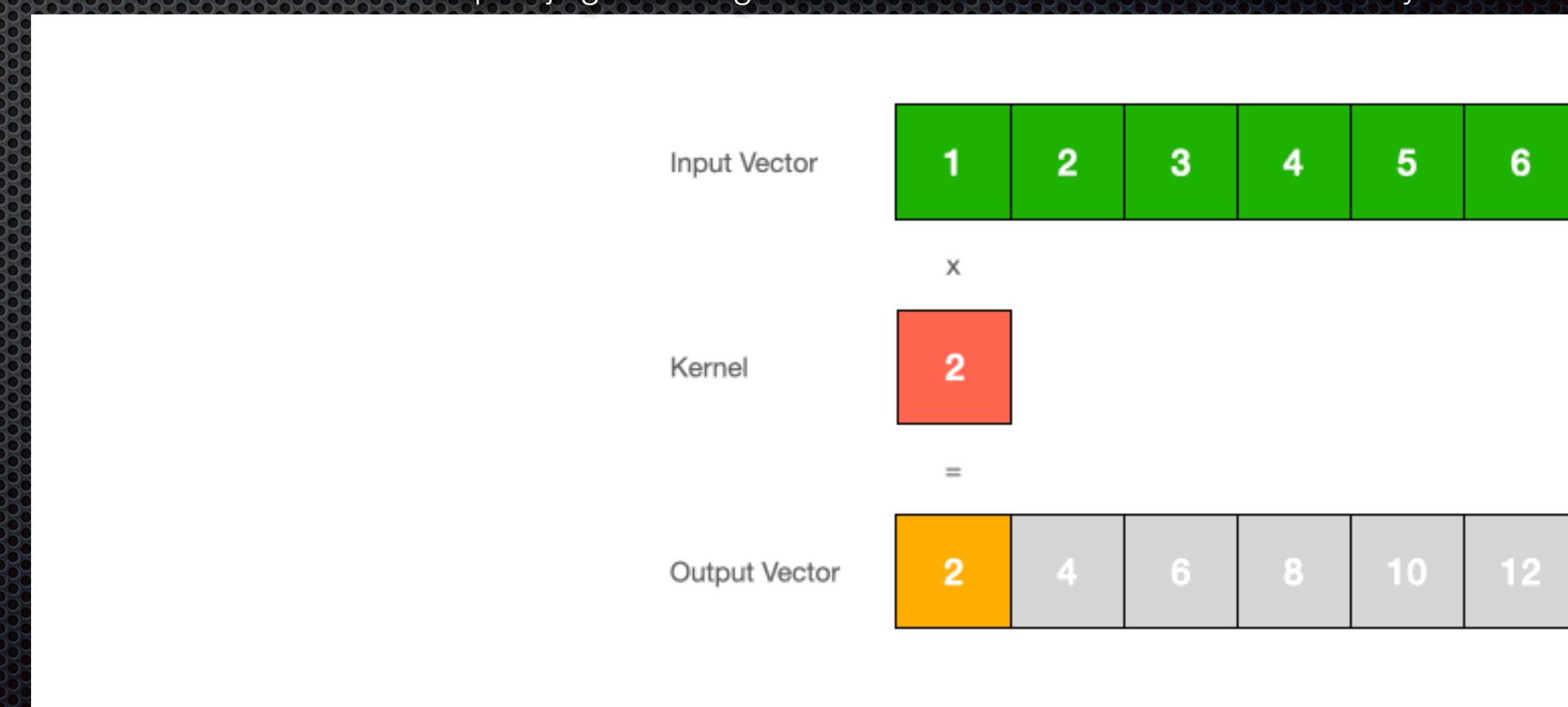


Feature map $y[i]$



Dimension
(1,5)

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

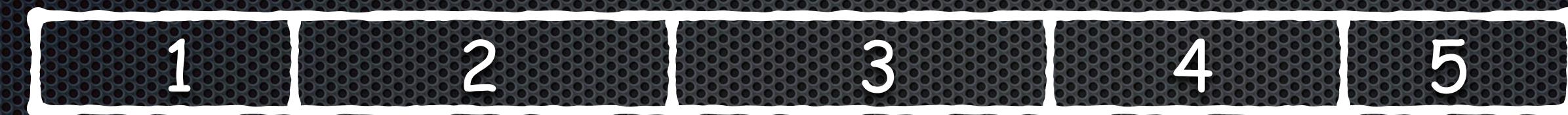


Convolution kernels

1D convolution

$$y = x * \omega = y[i] = \sum_{k=0}^K x[i-k] \omega[k]$$

Input $x[i]$

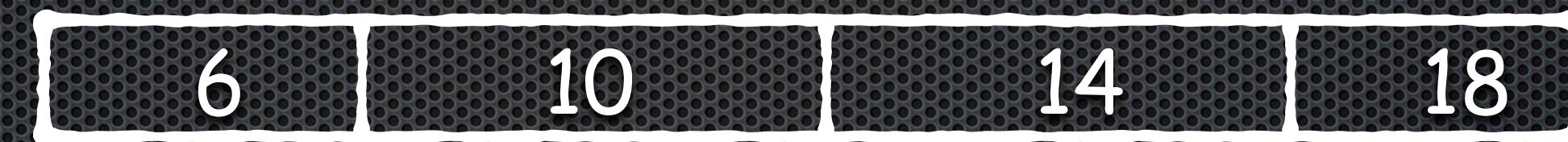


Dimension
(1,5)

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

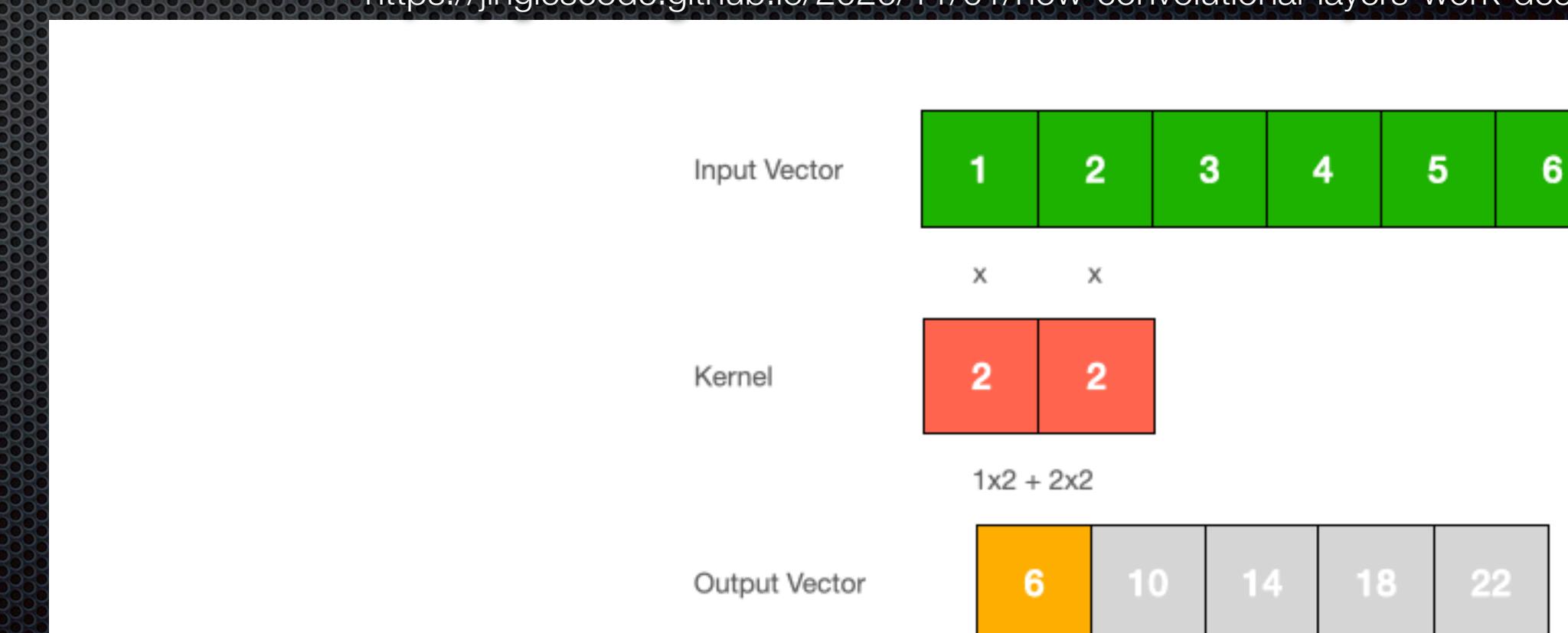


Feature map $y[i]$



Dimension
(1,4)

<https://jinglescode.github.io/2020/11/01/how-convolutional-layers-work-deep-learning-neural-networks/> 33



Convolution kernels

1D convolution

$$y = x * \omega = y[i] = \sum_{k=0}^K x[i-k] \omega[k]$$

Input $x[i]$



Dimension
(1,5)

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

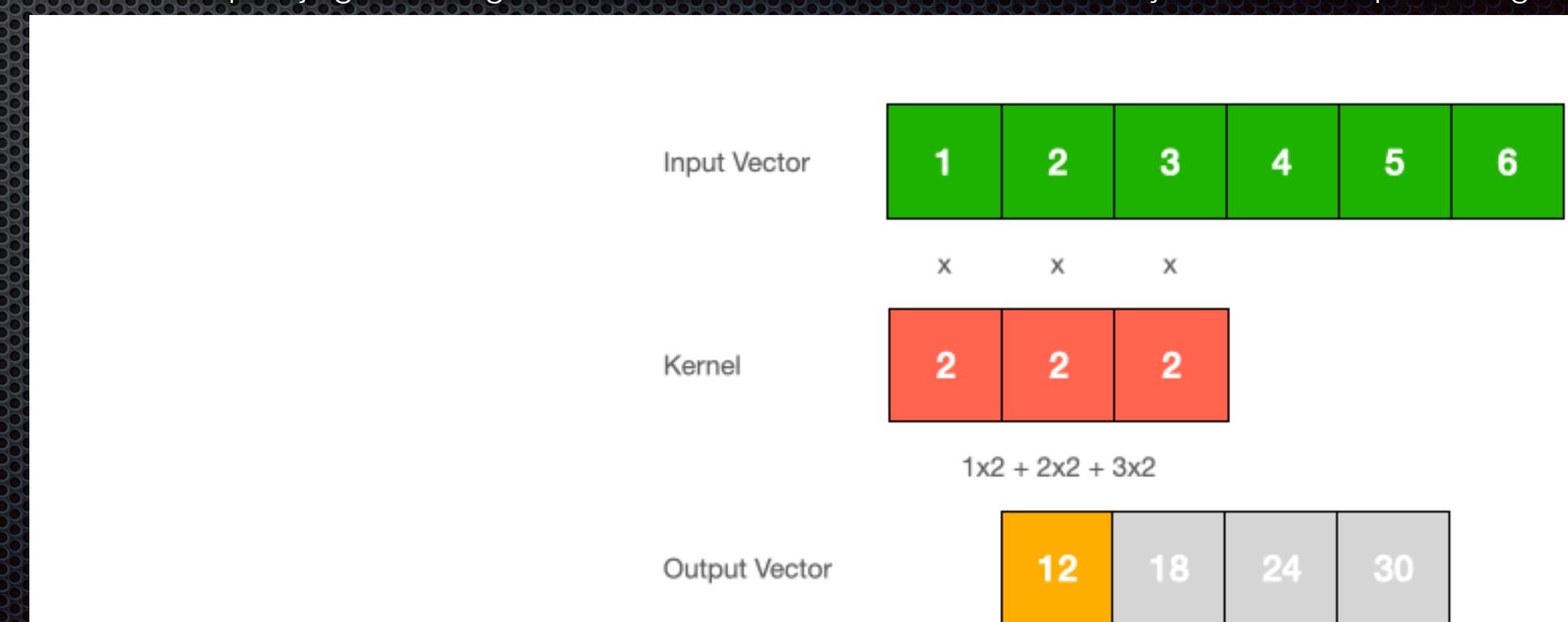


Feature map $y[i]$



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



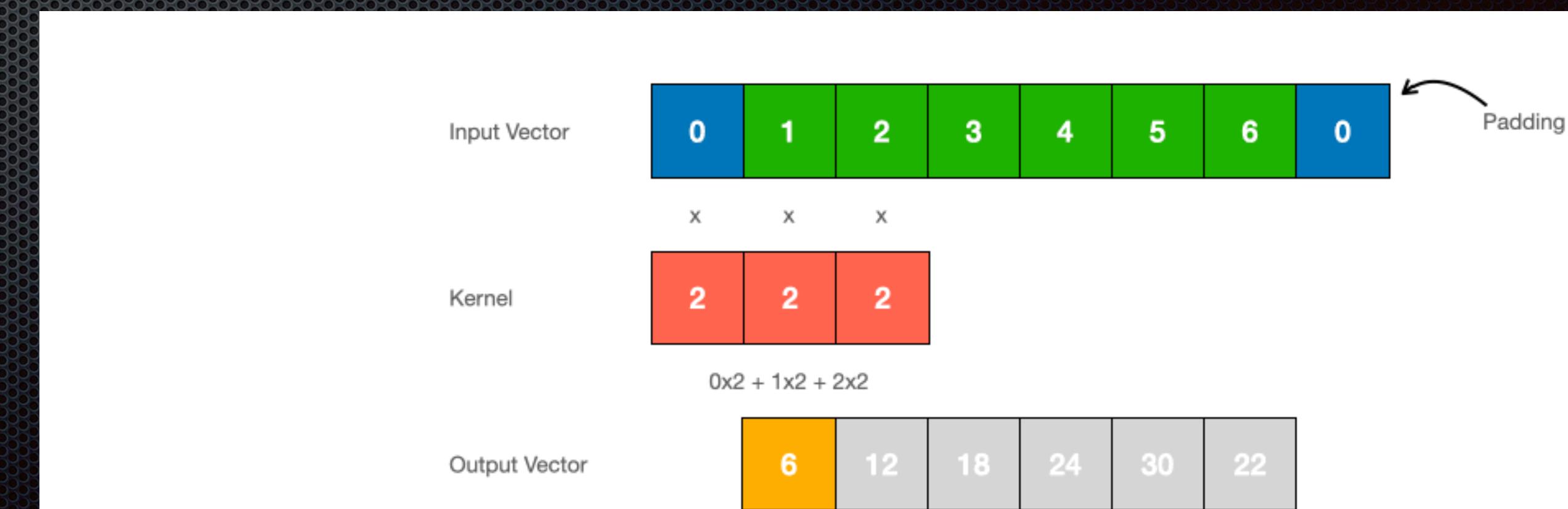
Feature map $y[i]$



Dimension
(1,5)

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

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



Shifts of the filter (strides)

Stride: the step size of the shifting filter

Input $x[i]$



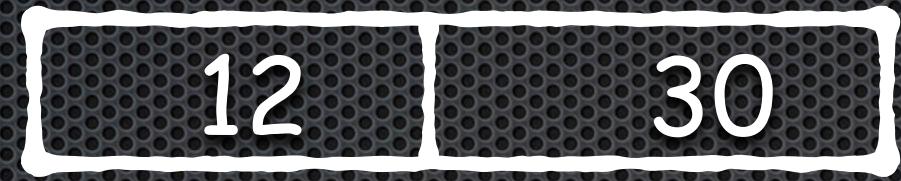
Dimension
(1,5)

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



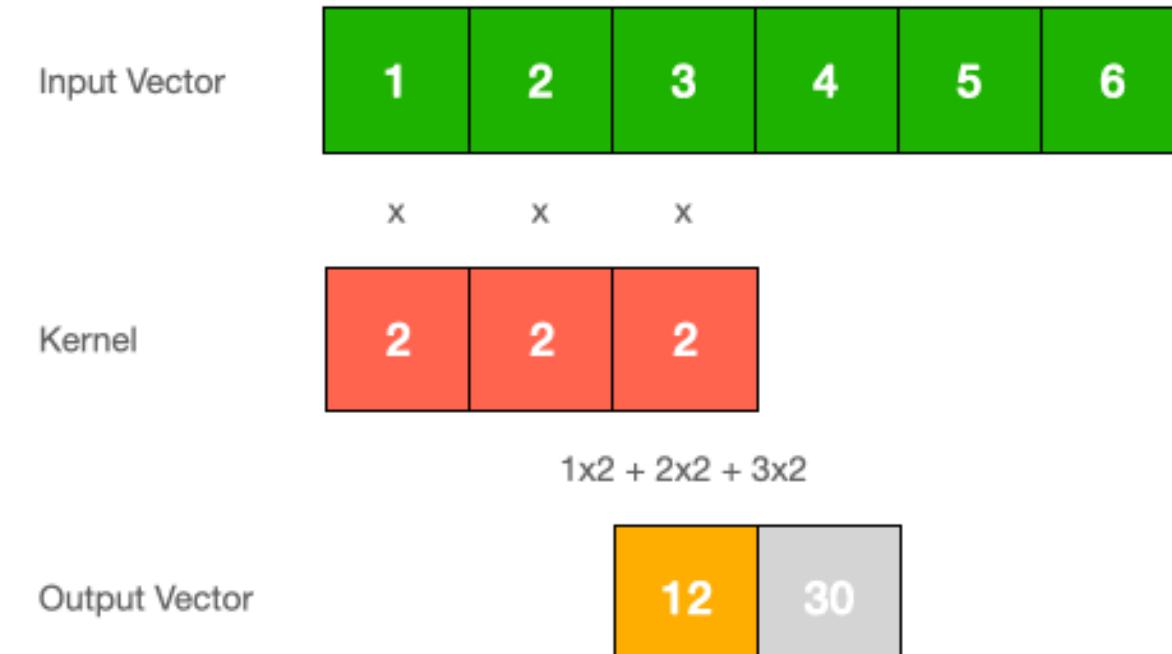
Stride = 3

Feature map $y[i]$



Dimension
(1,3)

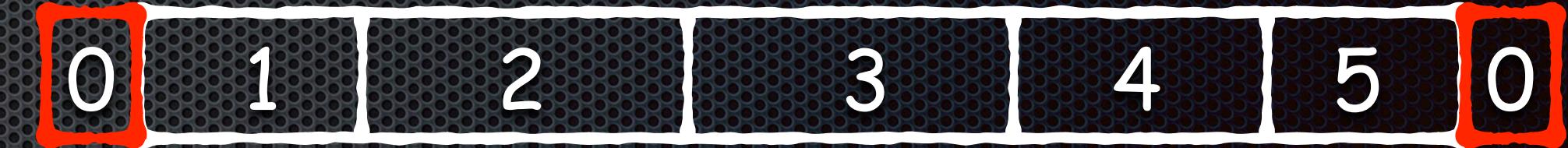
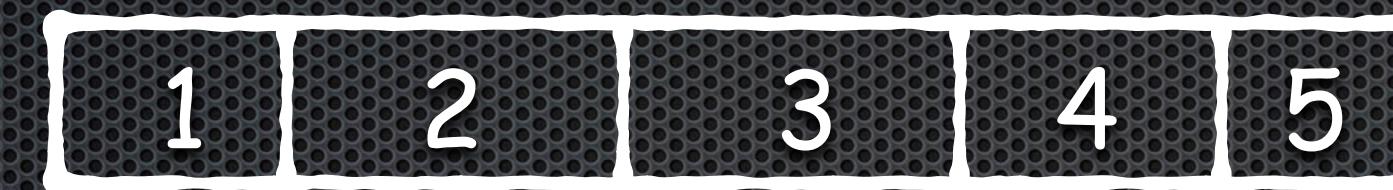
<https://jinglescode.github.io/2020/11/01/how-convolutional-layers-work-deep-learning-neural-networks/>



Size of the convolution output

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

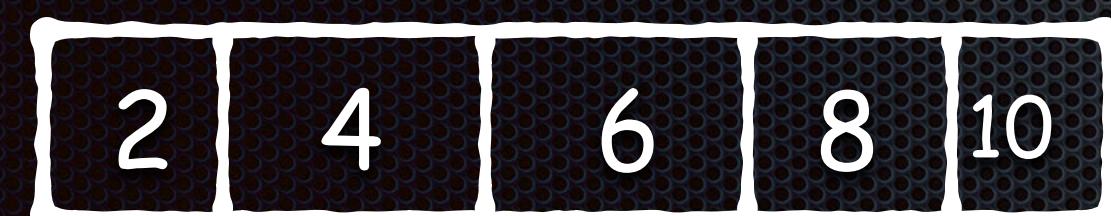
Please confirm....



2

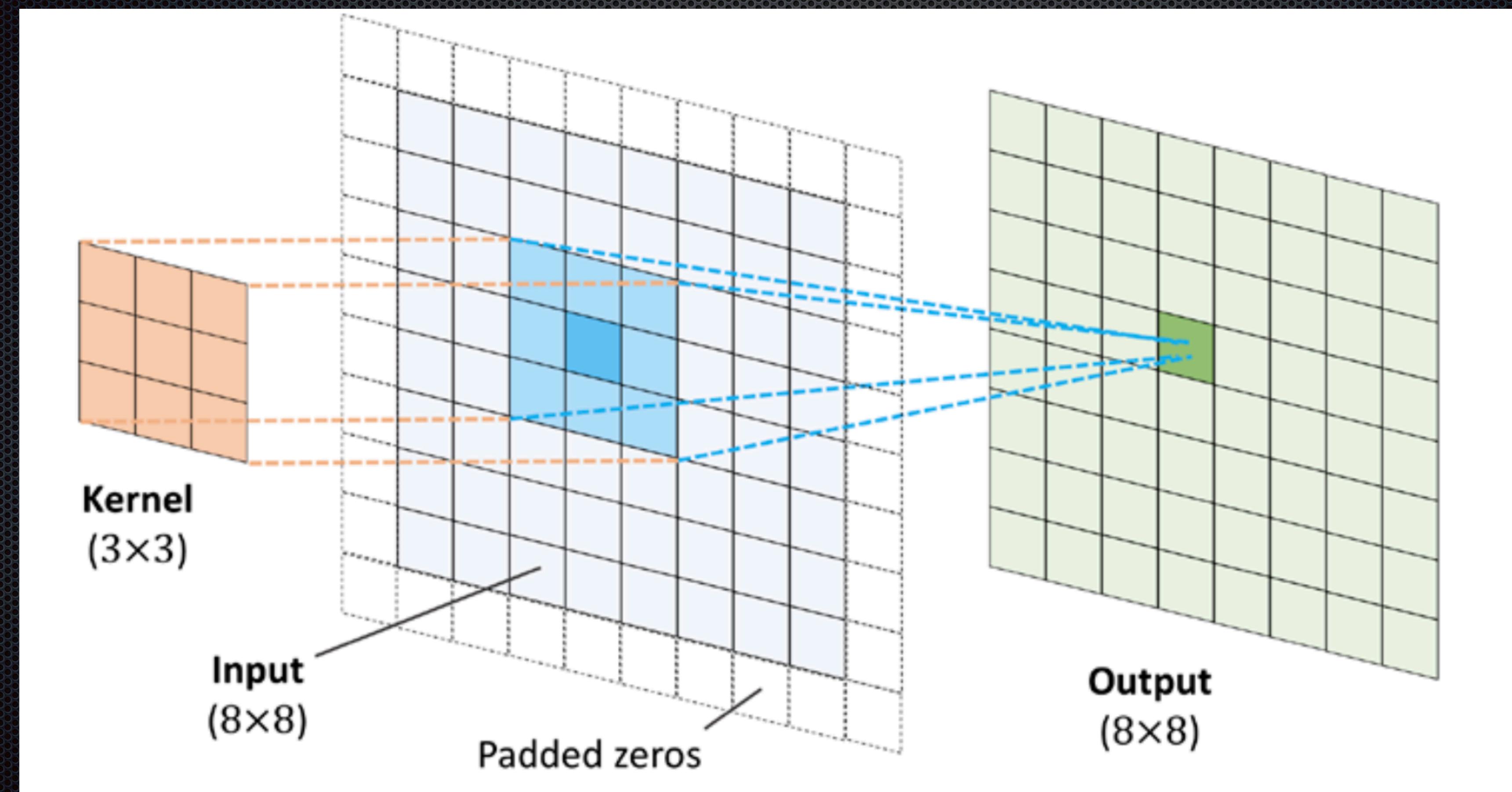
22

222



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

0 ₂	0 ₀	0 ₁	0	0	0	0
0 ₁	2 ₀	2 ₀	3	3	3	0
0 ₀	0 ₁	1 ₁	3	0	3	0
0	2	3	0	1	3	0
0	3	3	2	1	2	0
0	3	3	0	2	3	0
0	0	0	0	0	0	0

1	6	5
7	10	9
7	10	8

Local weights sharing

Convolution output =

Size of input vector + 2*Padding - filter size

Stride

Convolution output =

(5,5) + 2*(1,1) - (3,3)

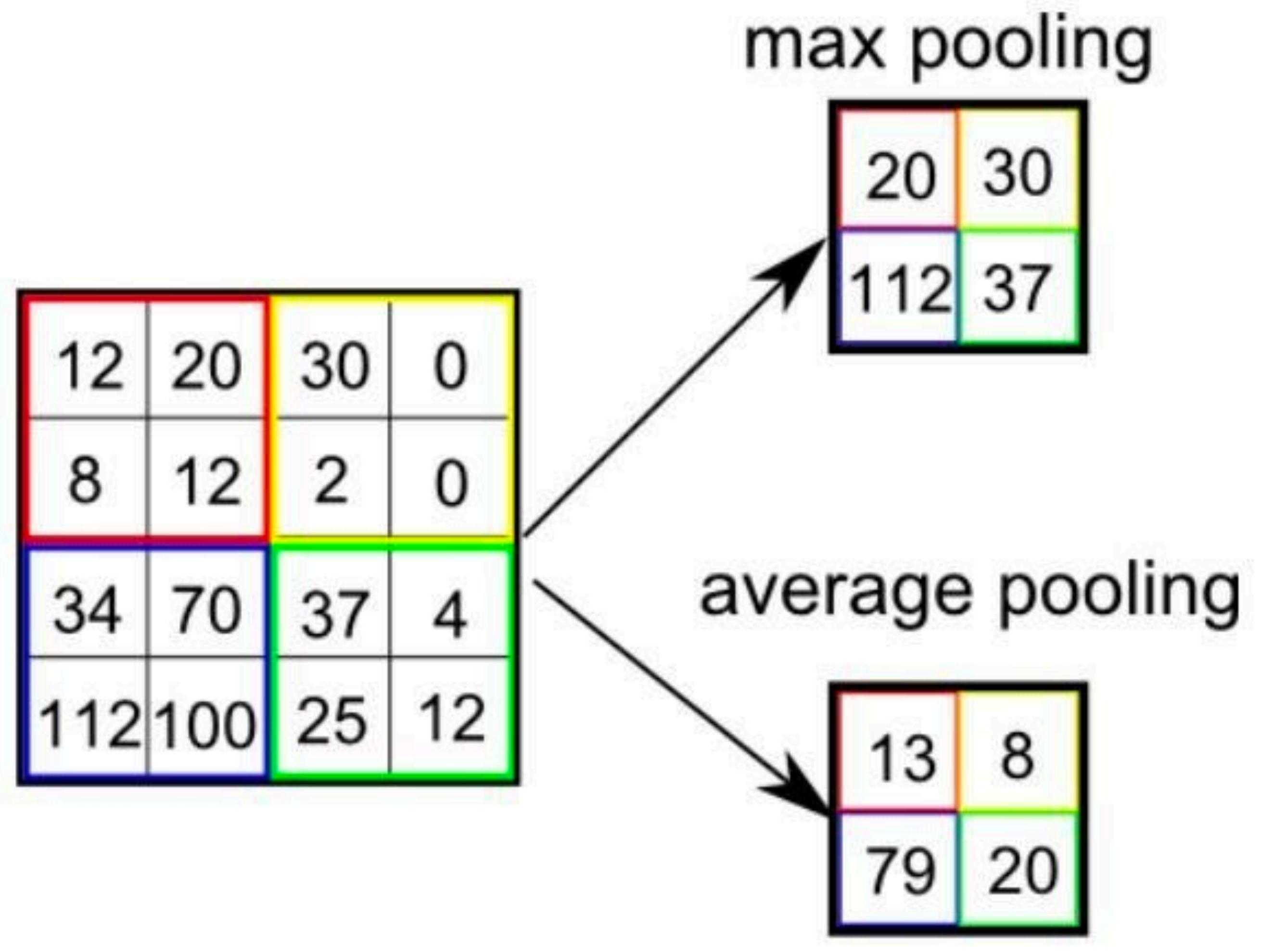
(2,2)

+1 = (3,3)

+1

Pooling

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



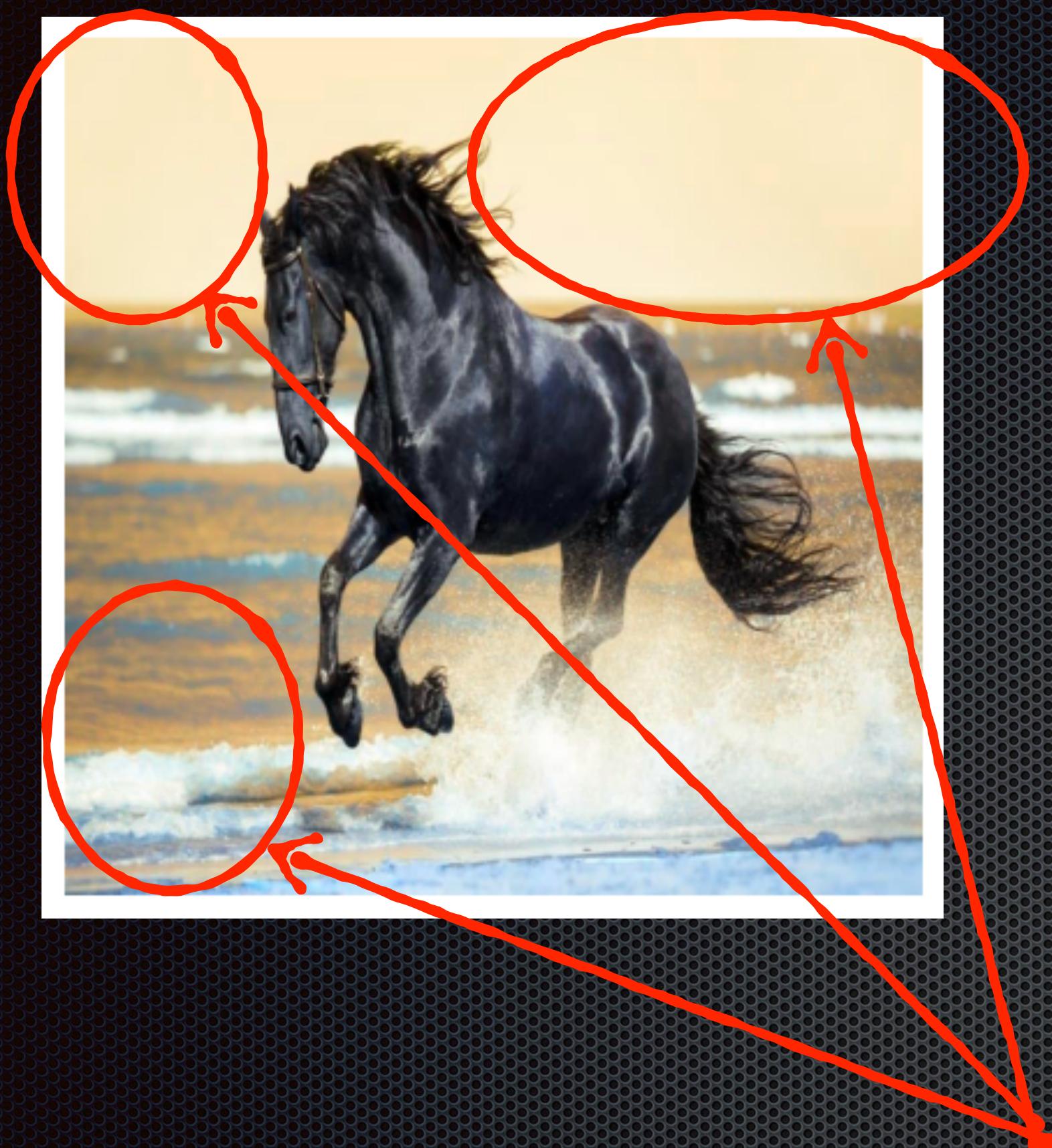
Back propagation

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

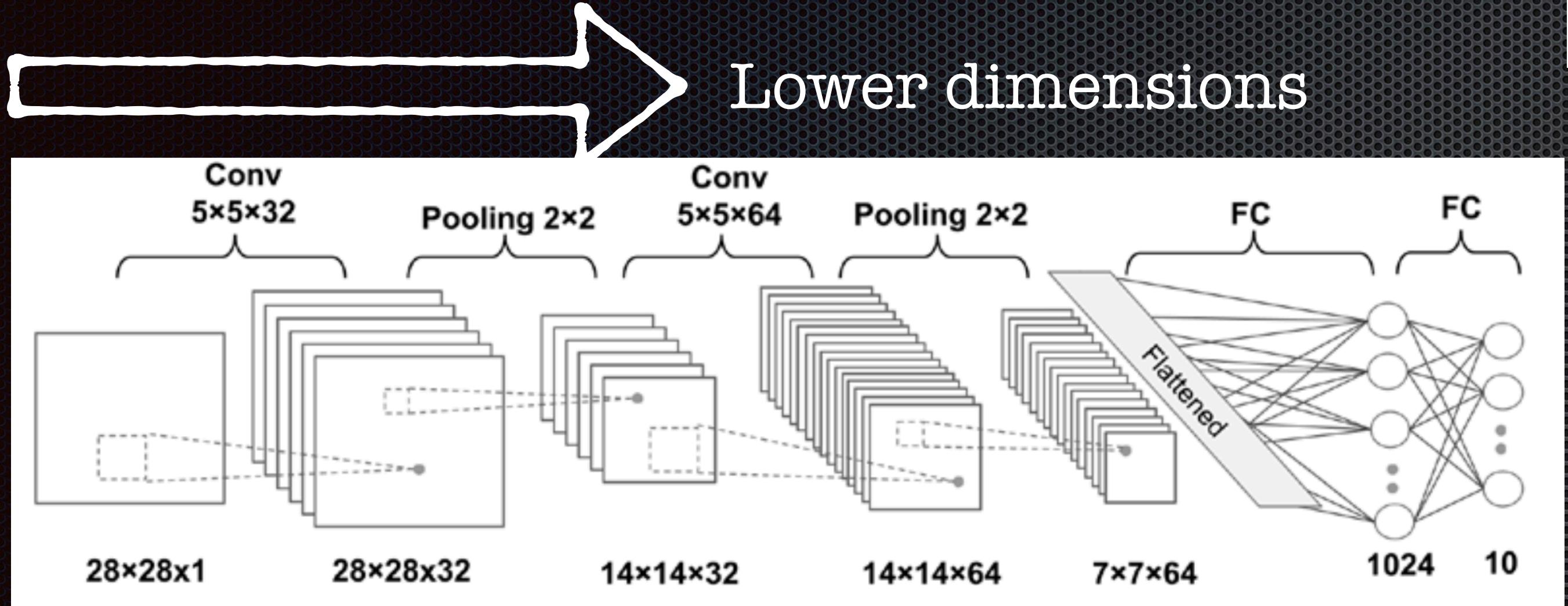
Please try it yourself and don't wait for the solution

Pooling

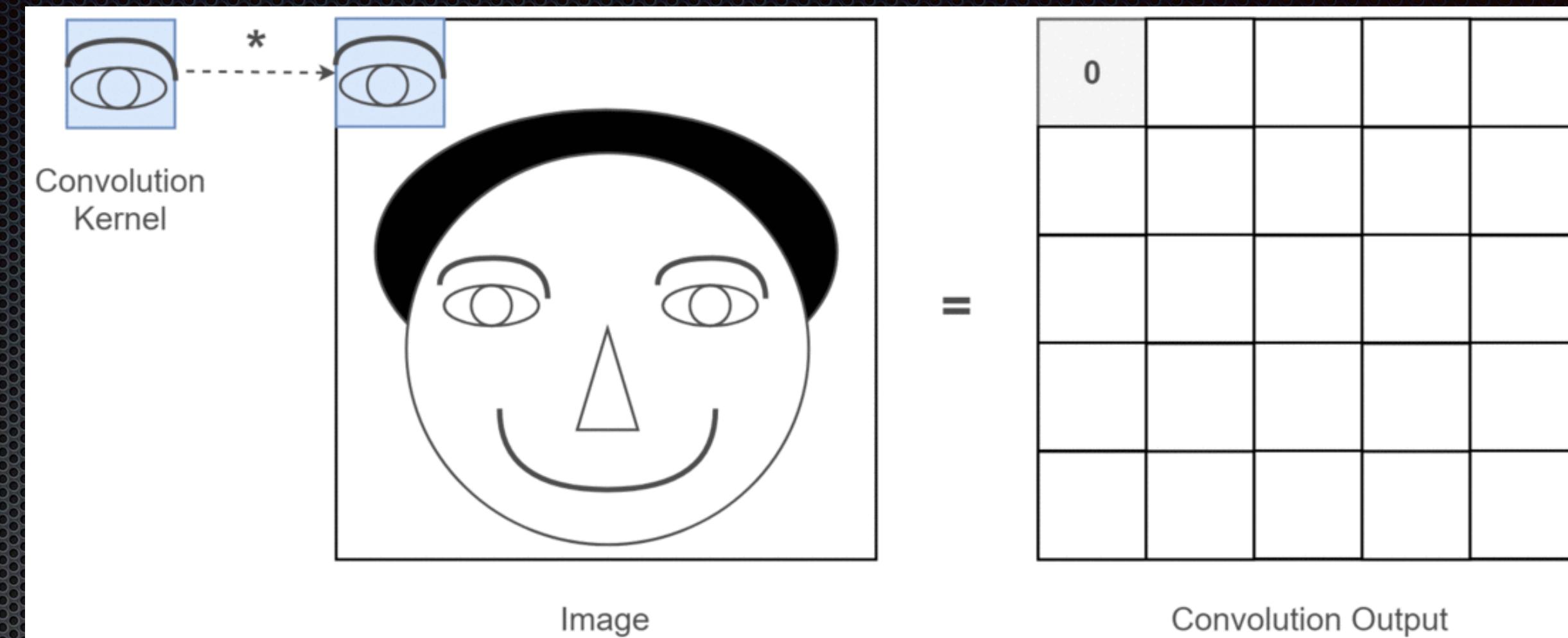


Redundant information which
can be diluted using pooling layers

All together



Lower dimensions



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

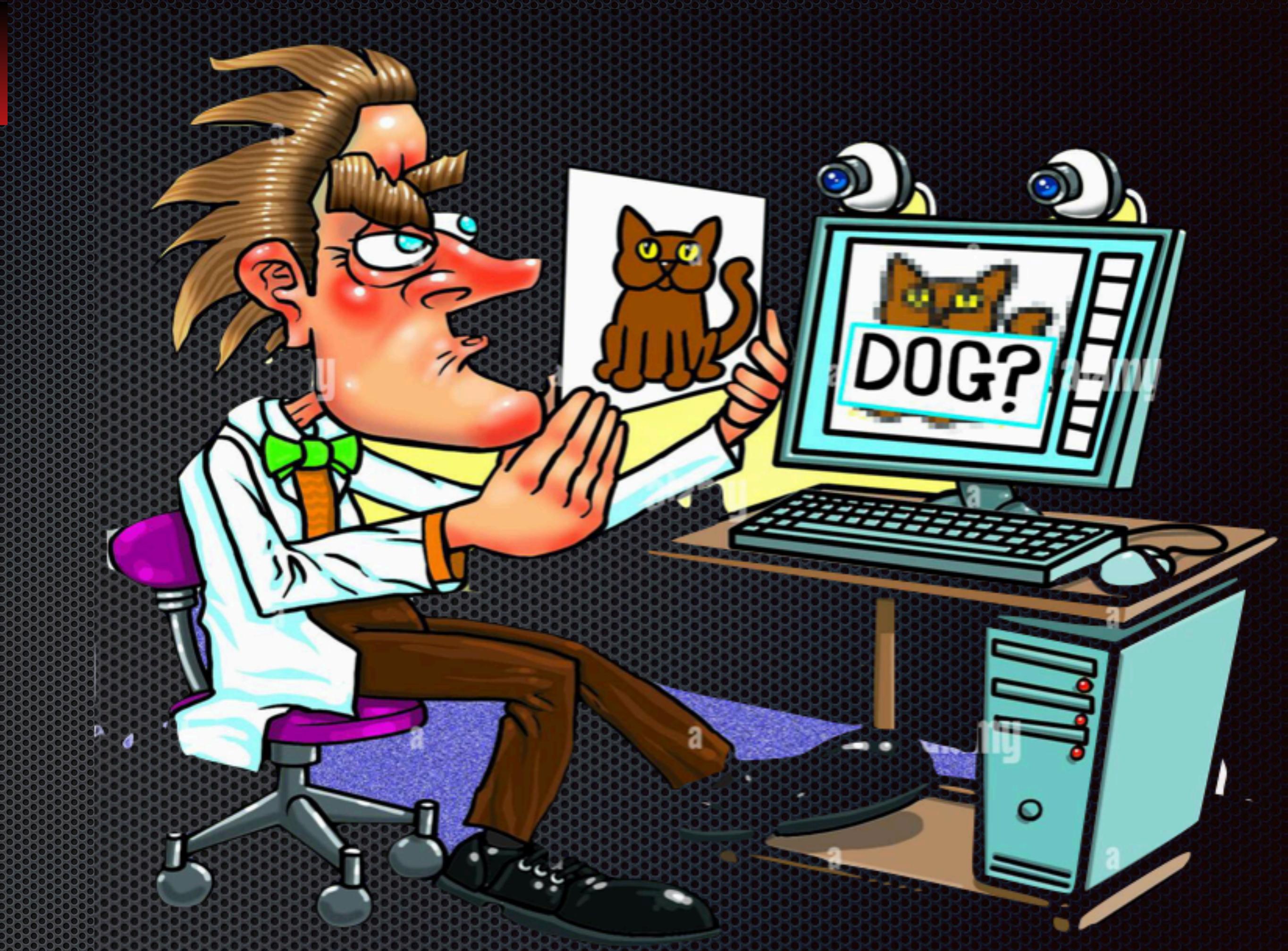
After the captured local information is mapped onto the decomposed latent space we use a fully connected layers to analyze these information and learn a global information about the input image

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



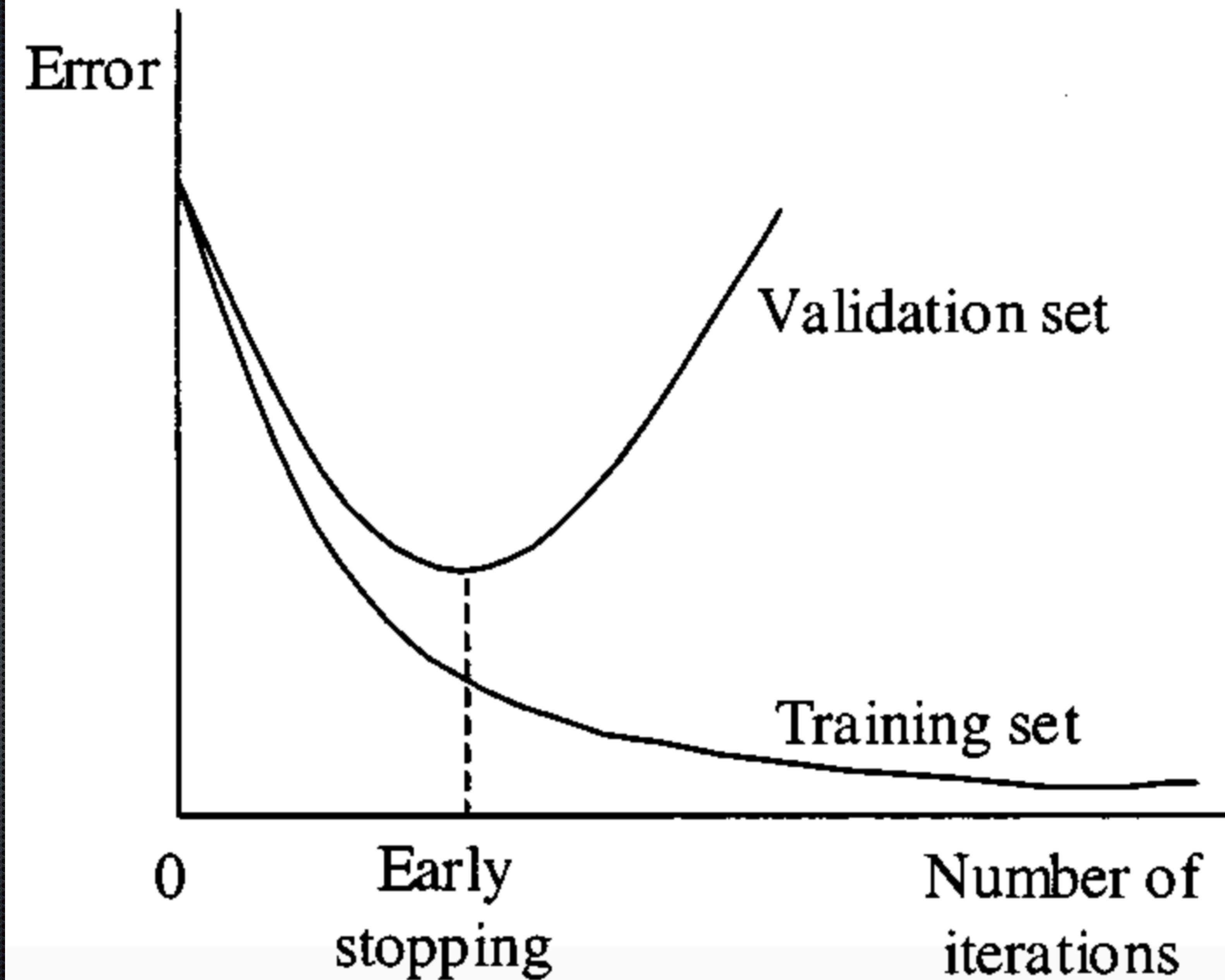
Overfitting in CNN models

• Early stopping

Terminate your model training
to the best fitting value

Implementation in tensorflow

```
tensorflow.keras.callbacks.EarlyStopping()
```

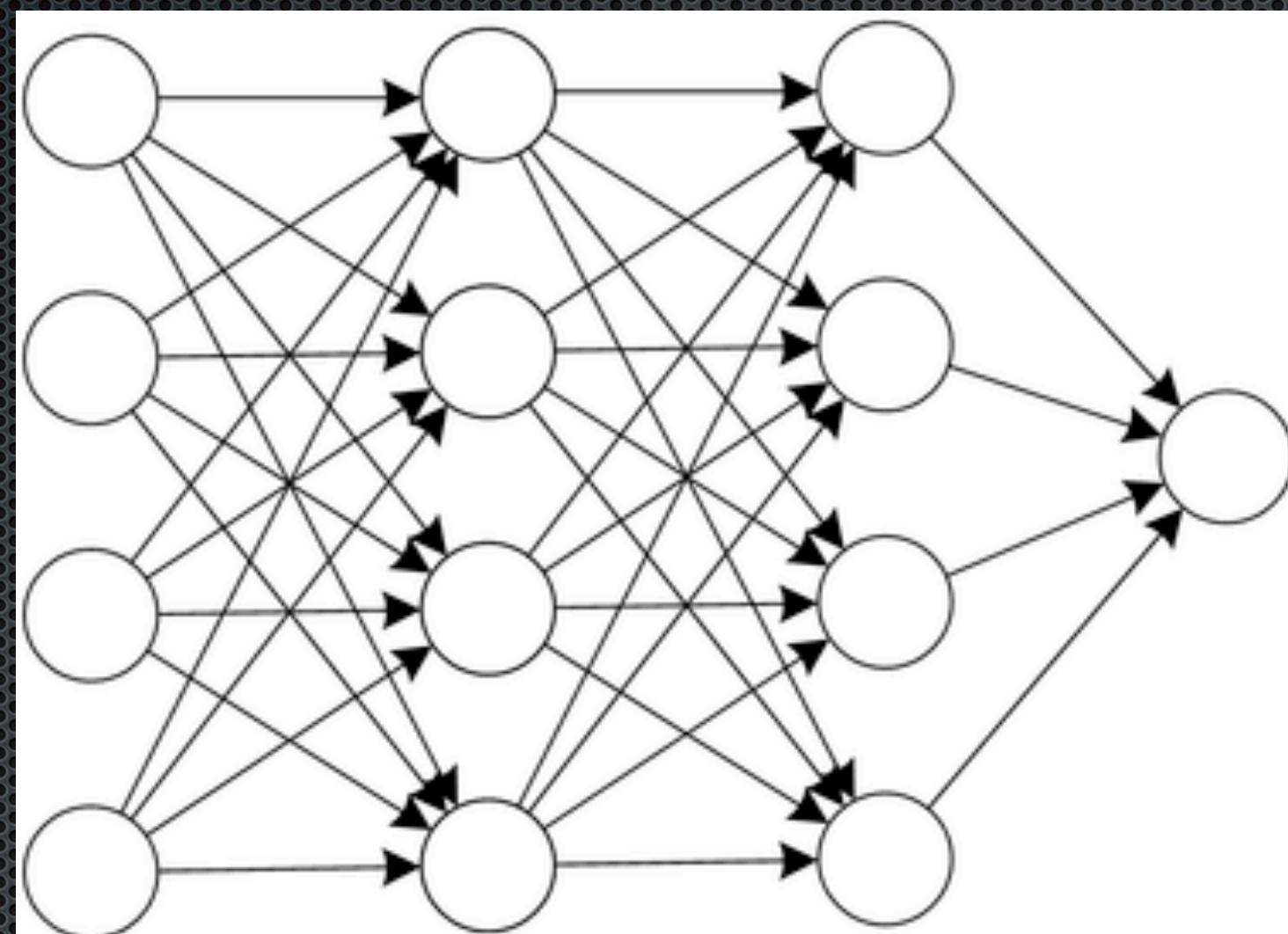




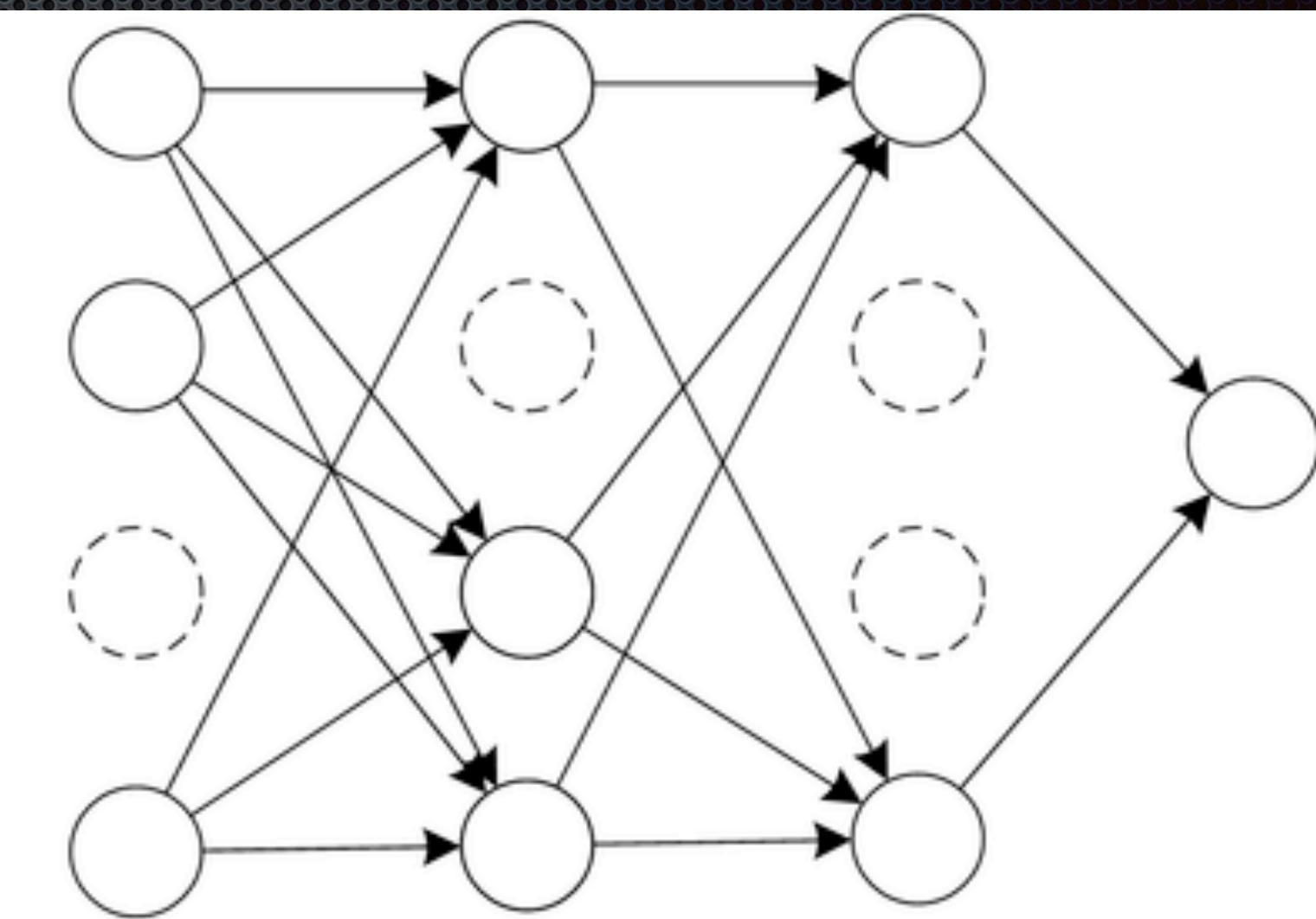
Random dropout

Implementation in tensorflow

```
tensorflow.keras.layers.dropout(rate)
```



(a) Standard Neural Network



(b) Network after Dropout

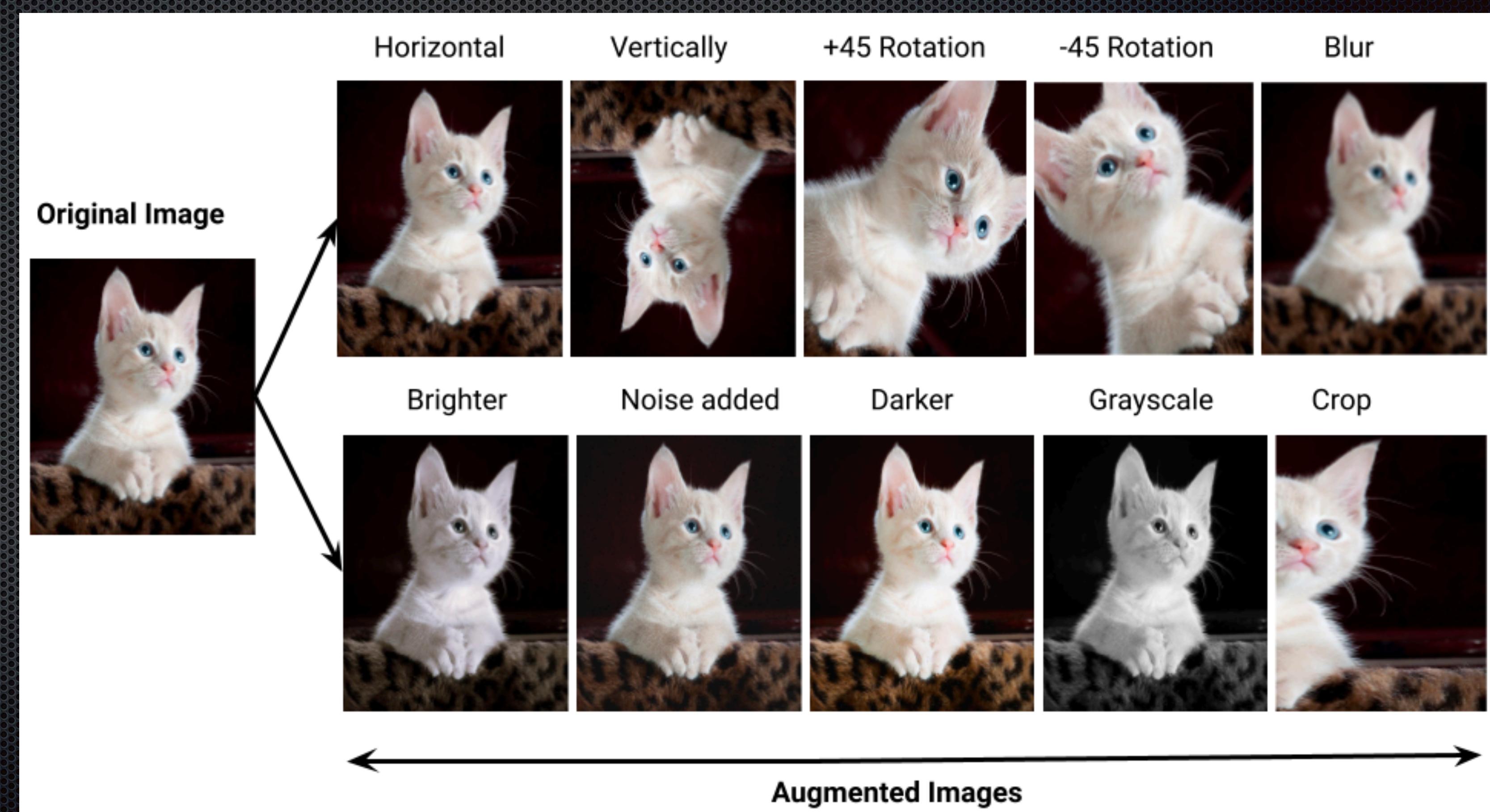
There is no theoretical proof on how the random dropout can avoid the overfitting but empirically it works fine!!

Overfitting in CNN models

④ 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)



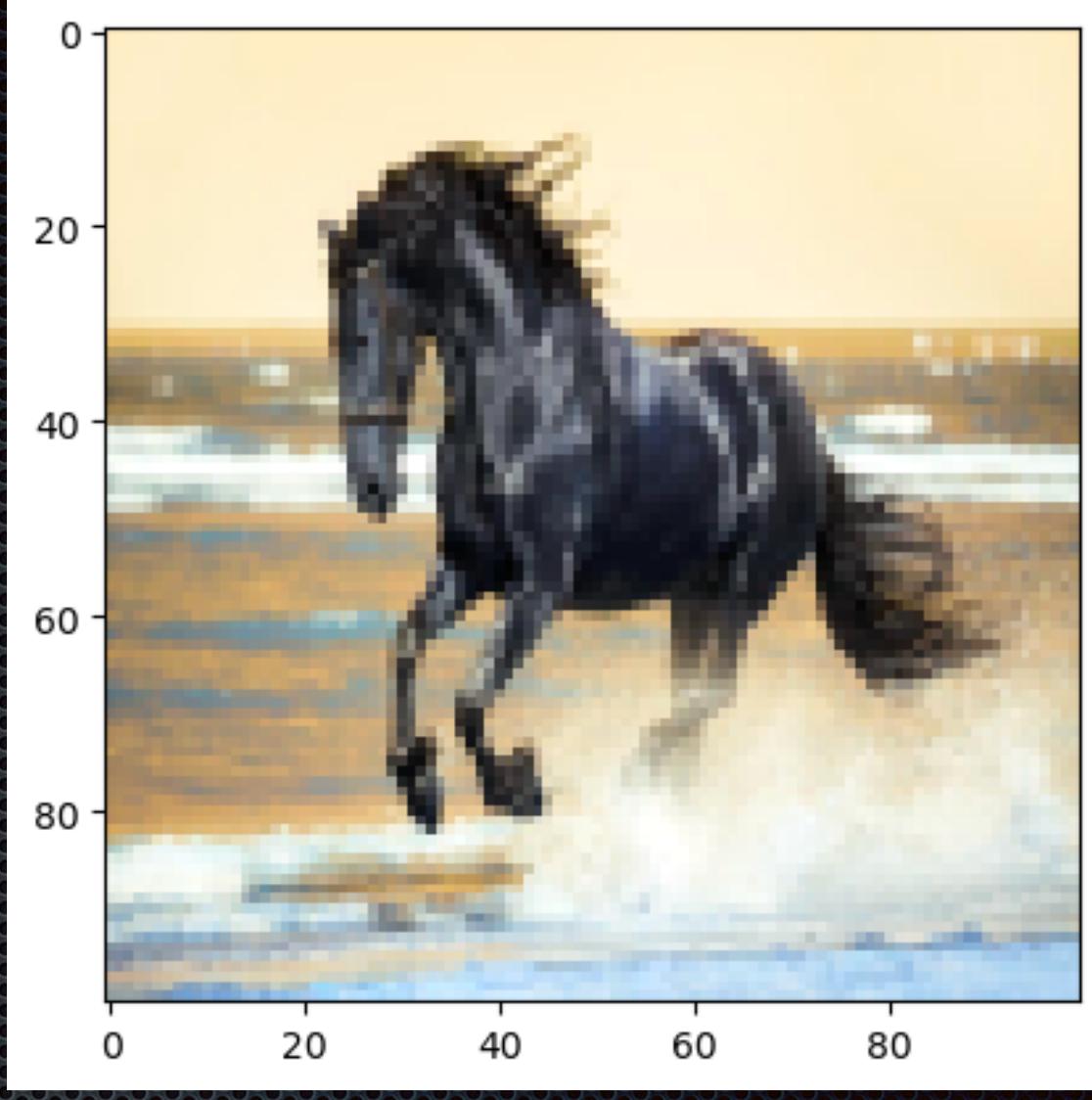
Binary image
Dimensions (length, width, 1)

Dealing with colored images

original image
nension: (100,100,3)

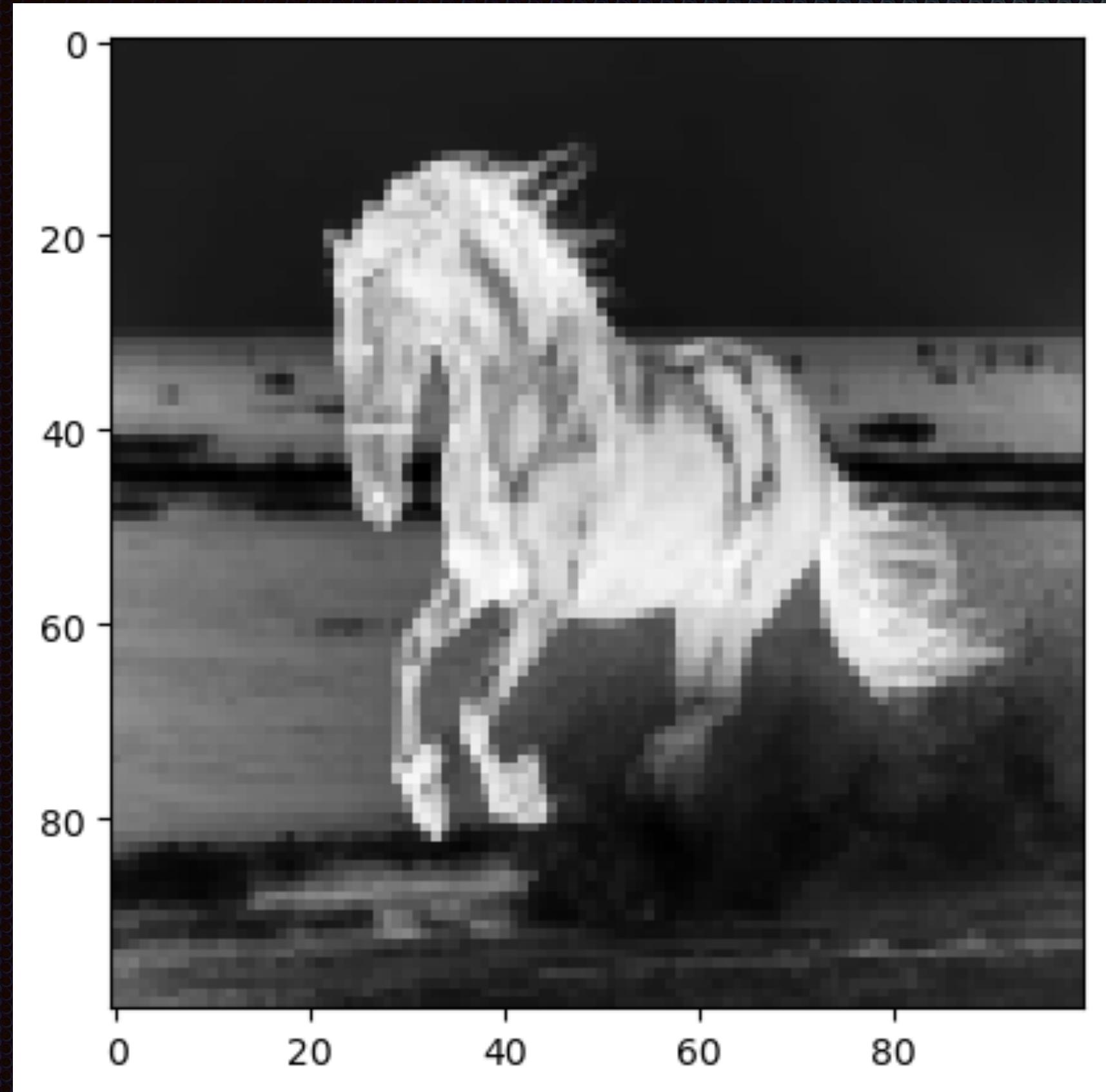
Blue dimension
Dimension: (100,100,

Red dimension
Dimension: (100,100,1)



Dealing with colored images

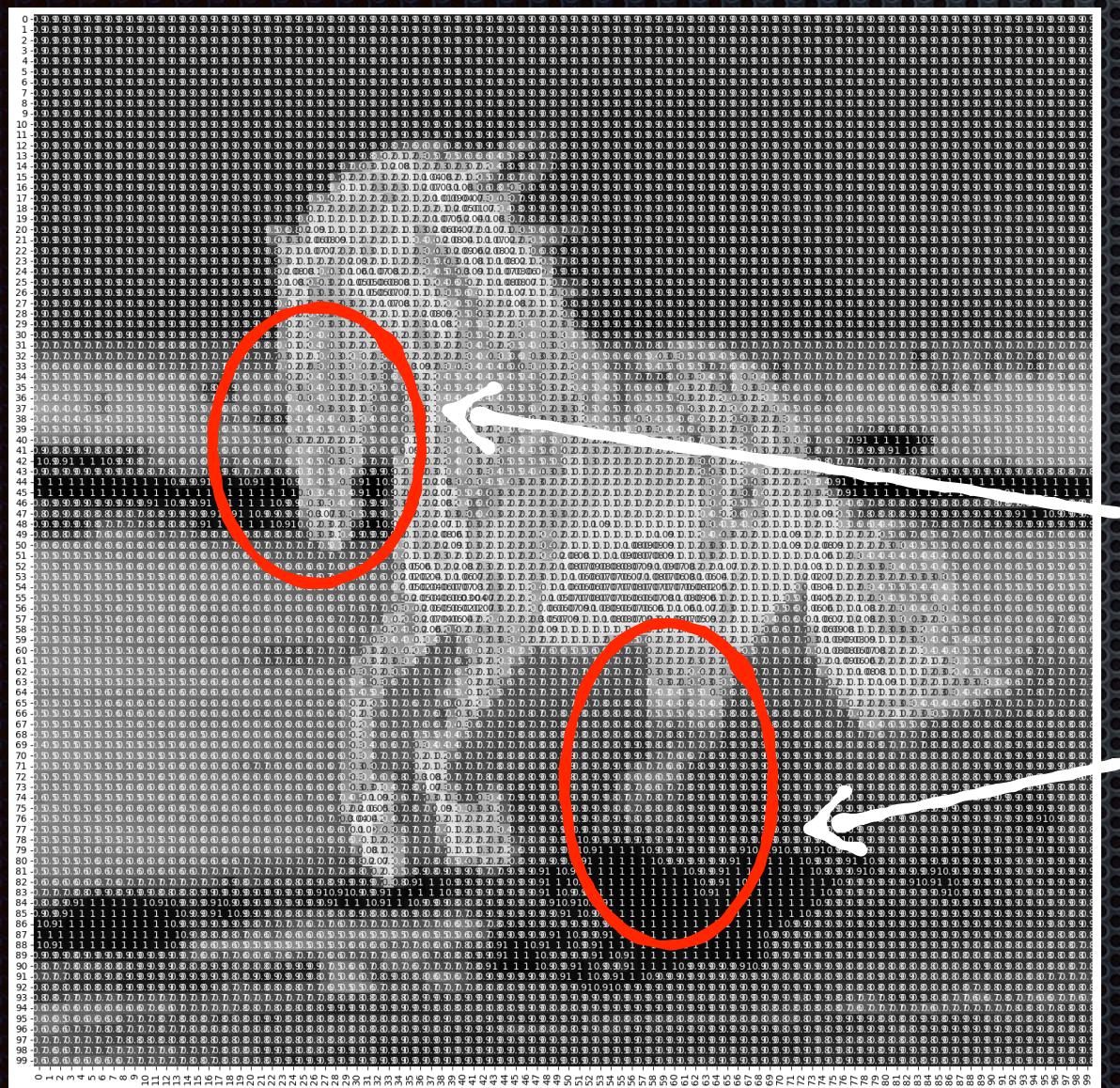
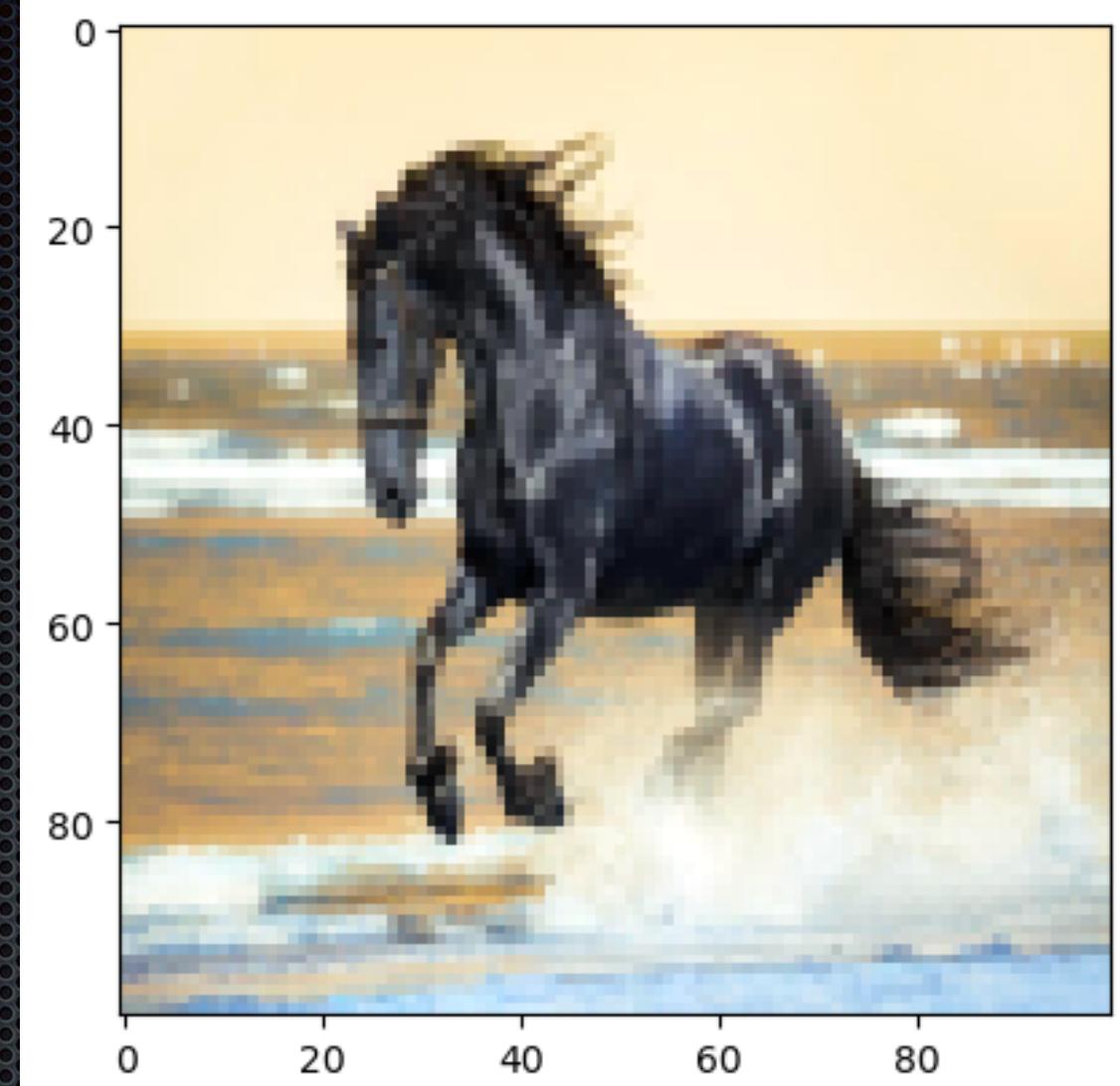
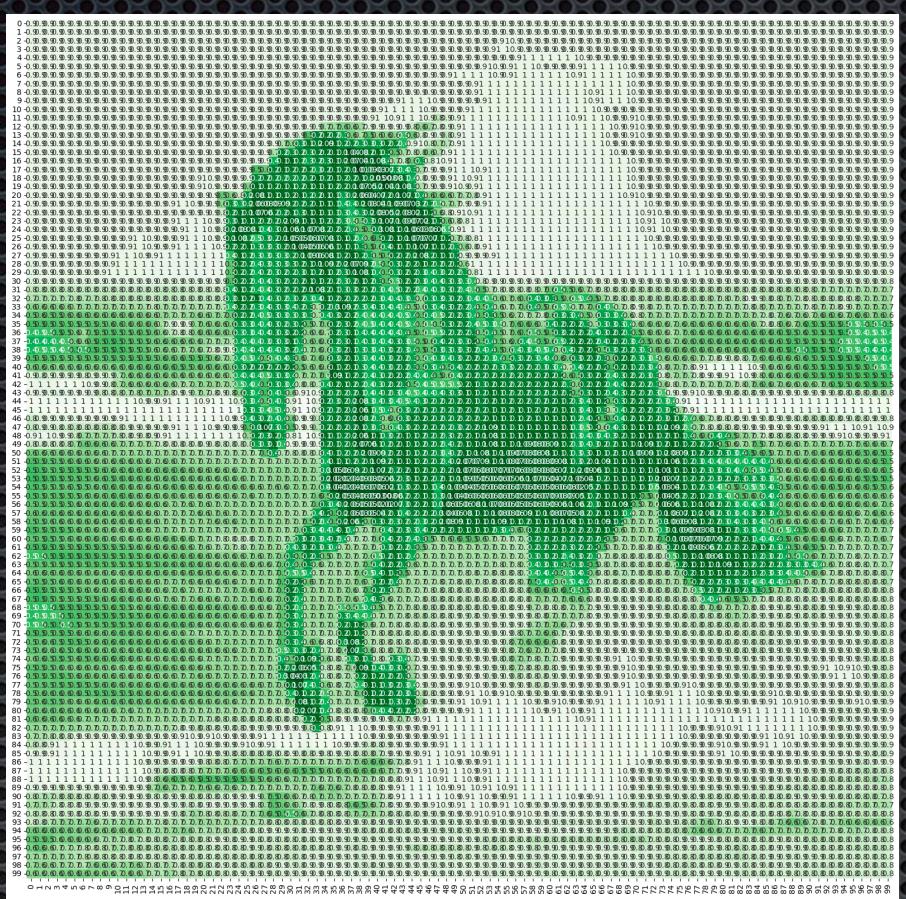
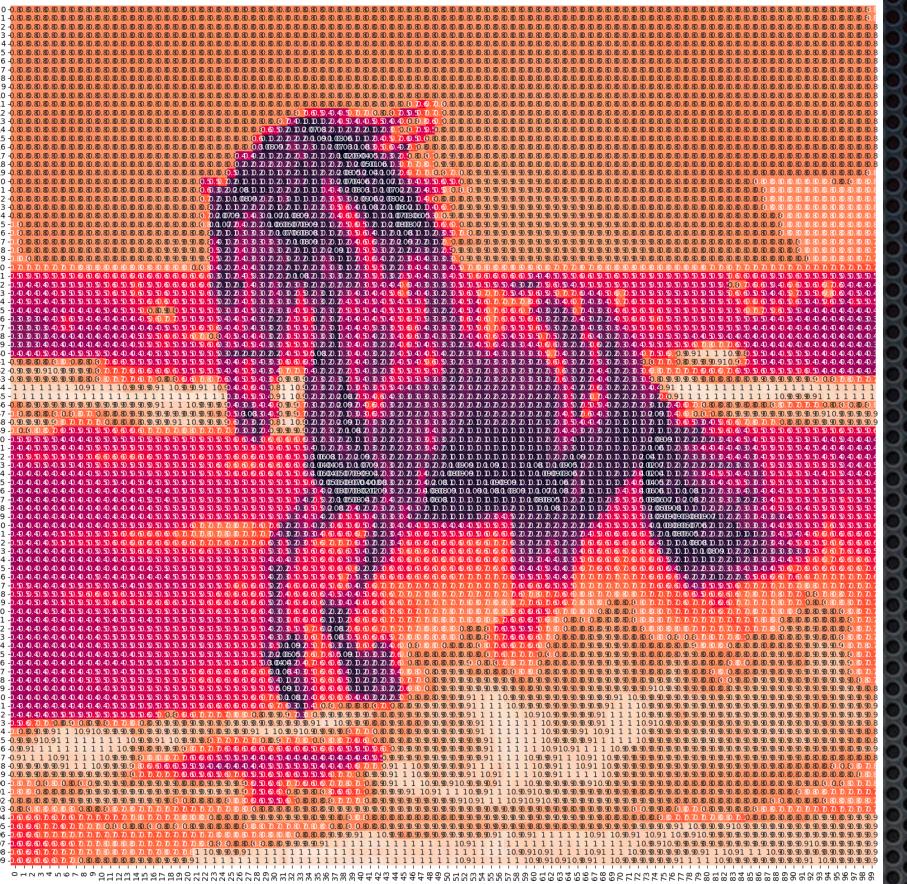
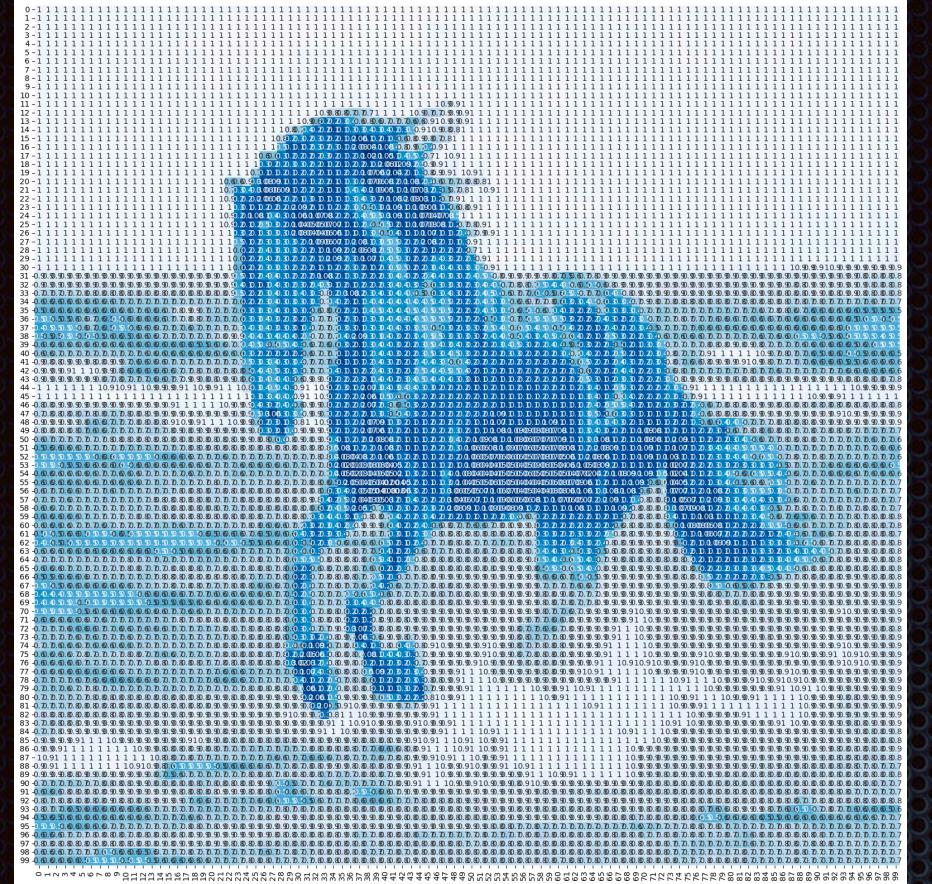
Original image
Dimension: (100,100,1)



Only binary channel

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99

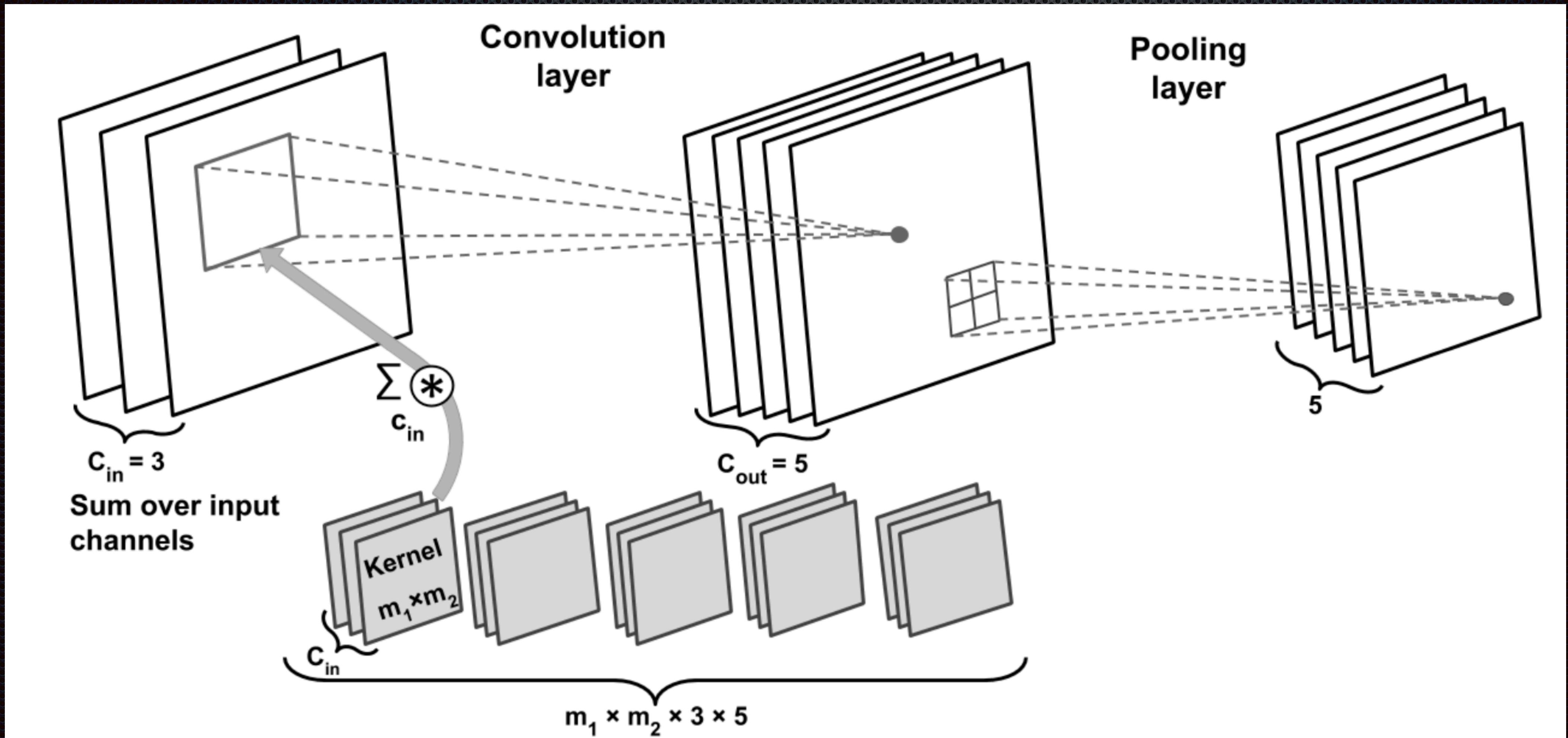
Dealing with colored images



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 2D)	(None, 25, 25, 16)	0
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 2D)	(None, 10, 10, 32)	0
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 2D)	(None, 4, 4, 64)	0

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

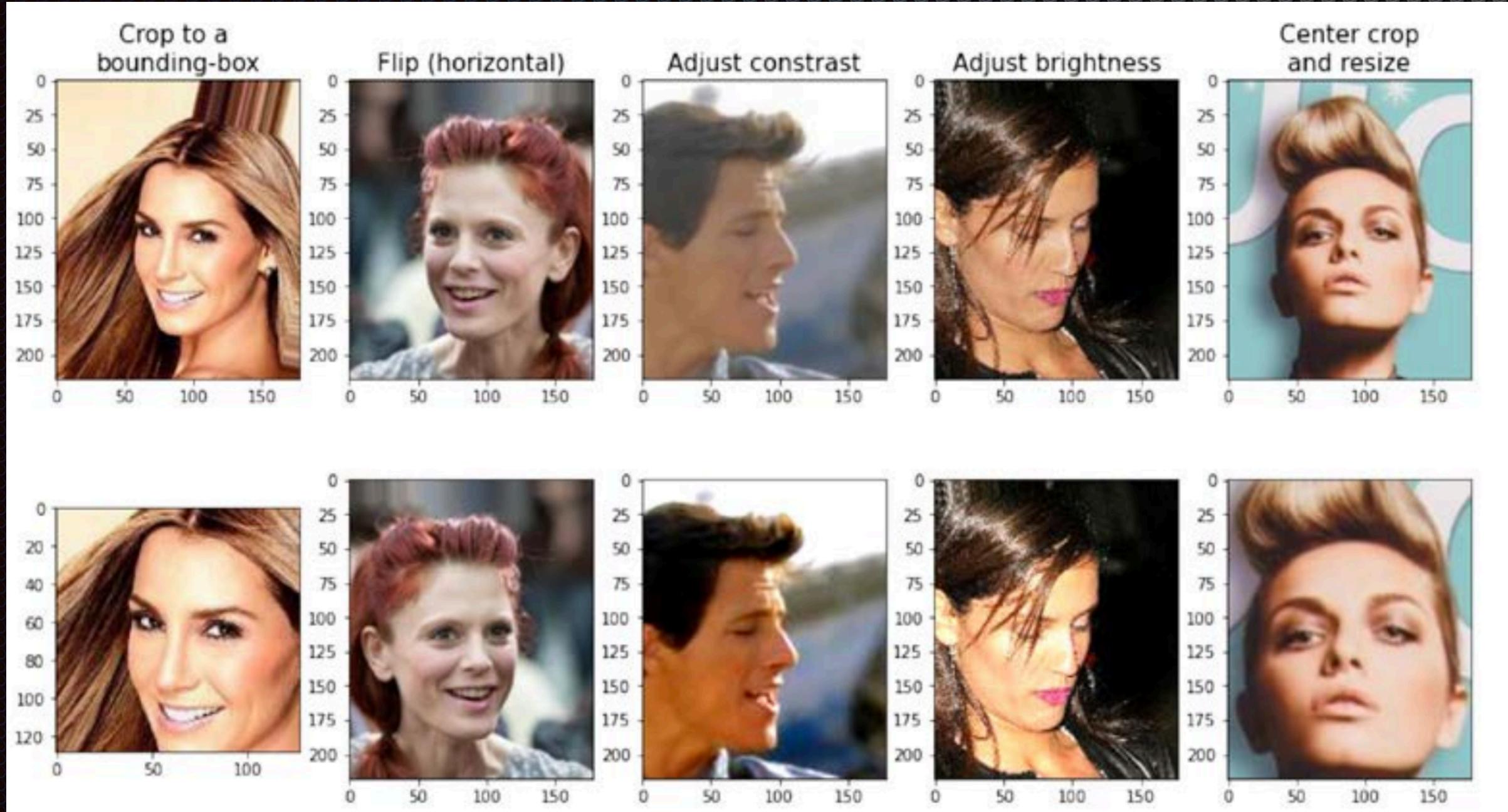
Total params: 206,723
Trainable params: 206,723
Non-trainable params: 0

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

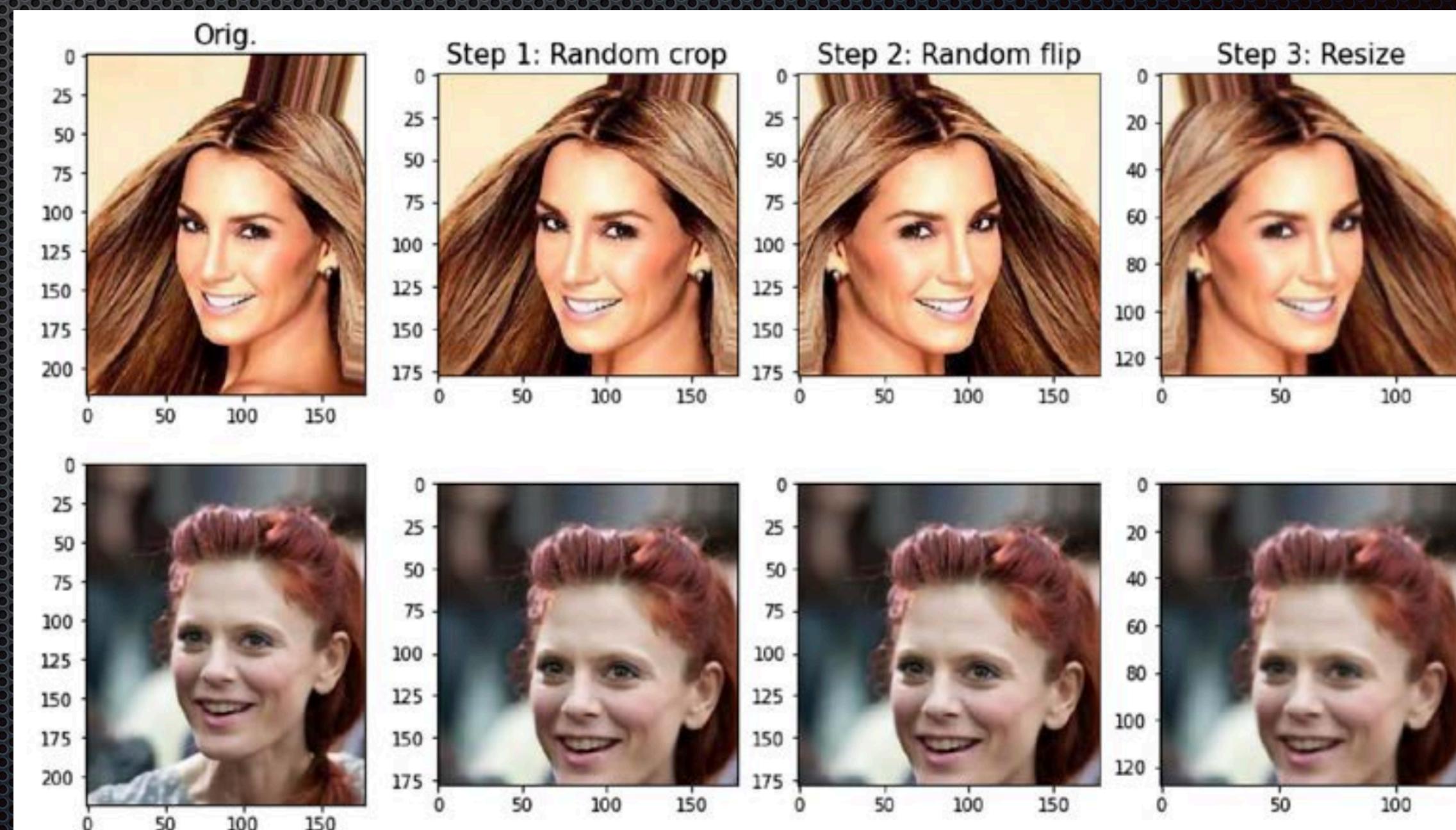
Example: Smily faces

Task: Construct a CNN model to classify the smily faces

Image preprocessing:

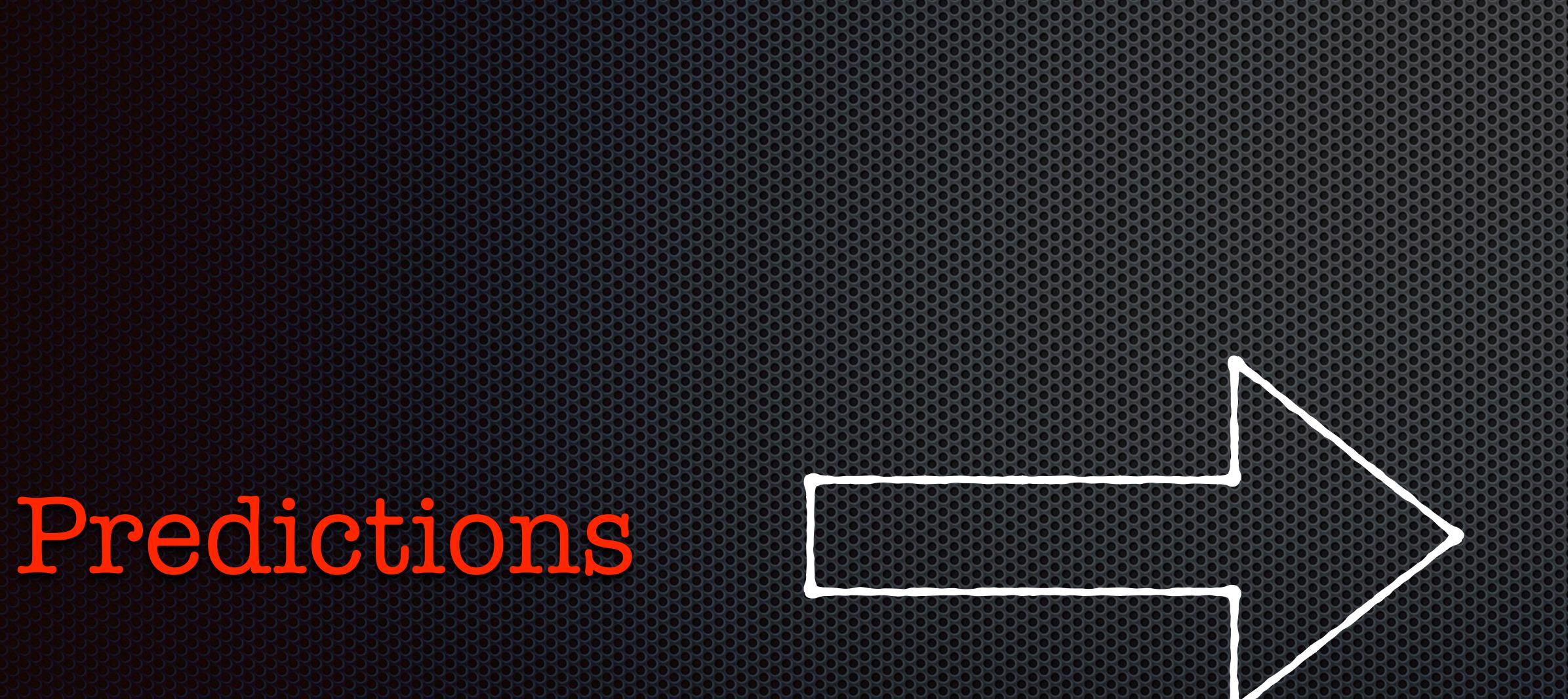
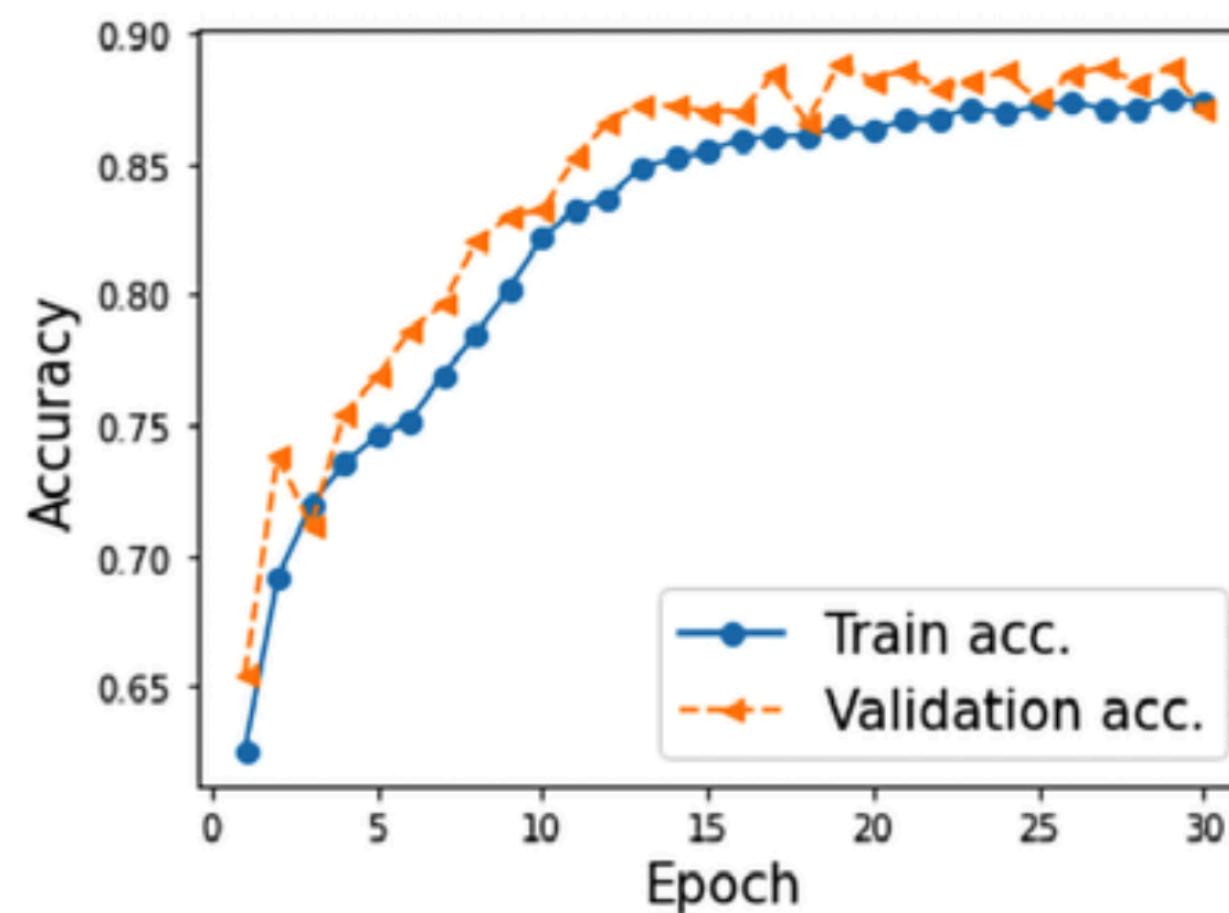
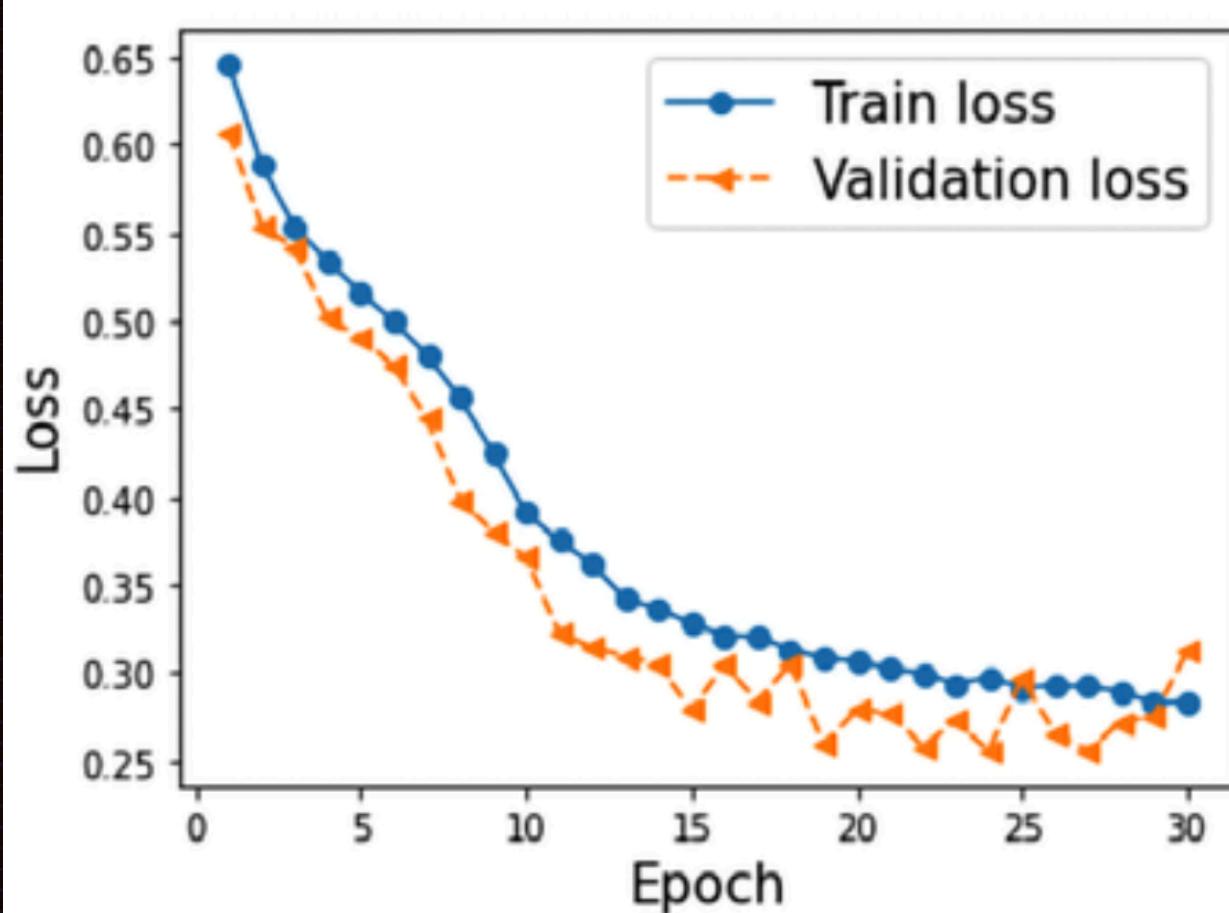


Data augmentation

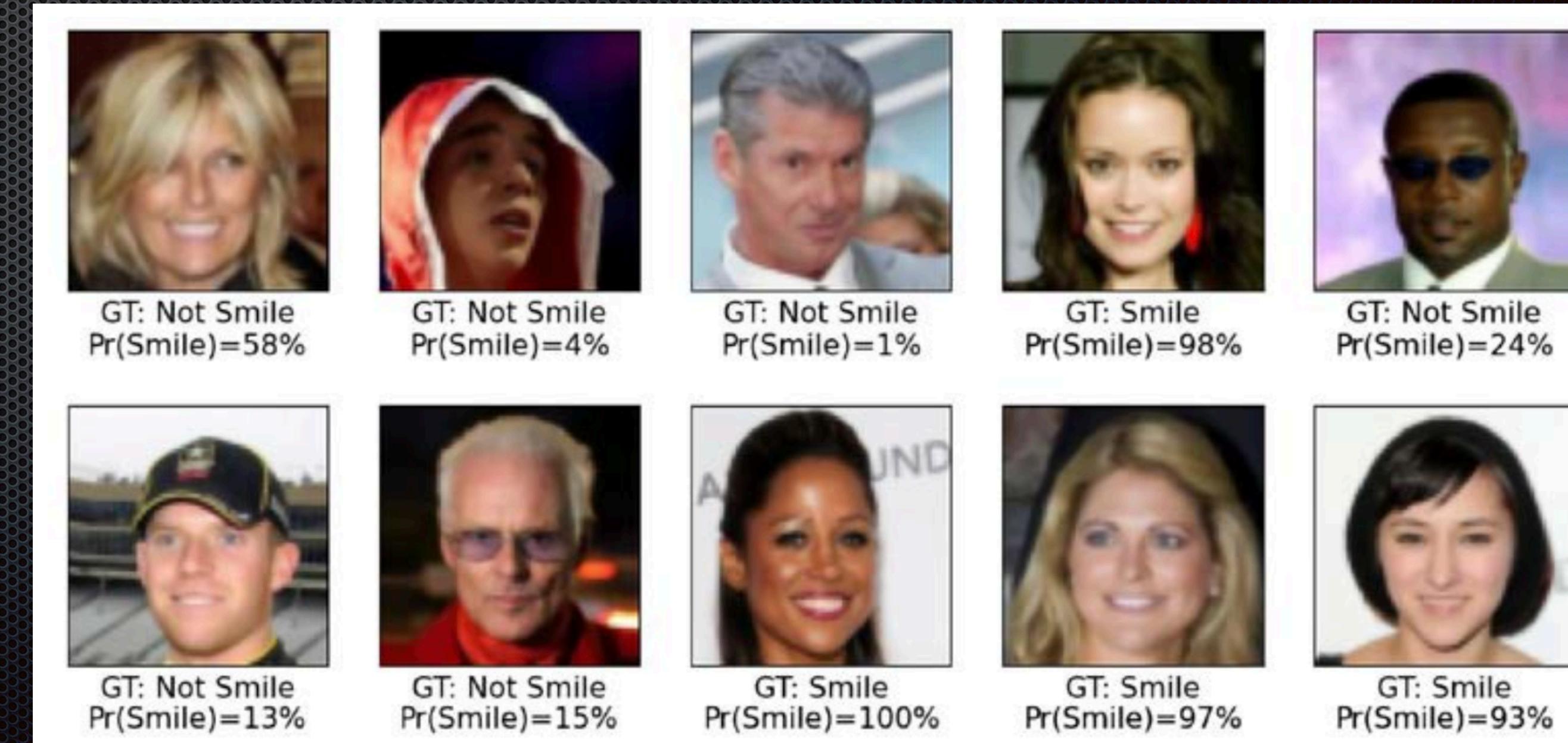


Example: Smiley faces

Task: Construct a CNN model to classify the smiley 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 ?

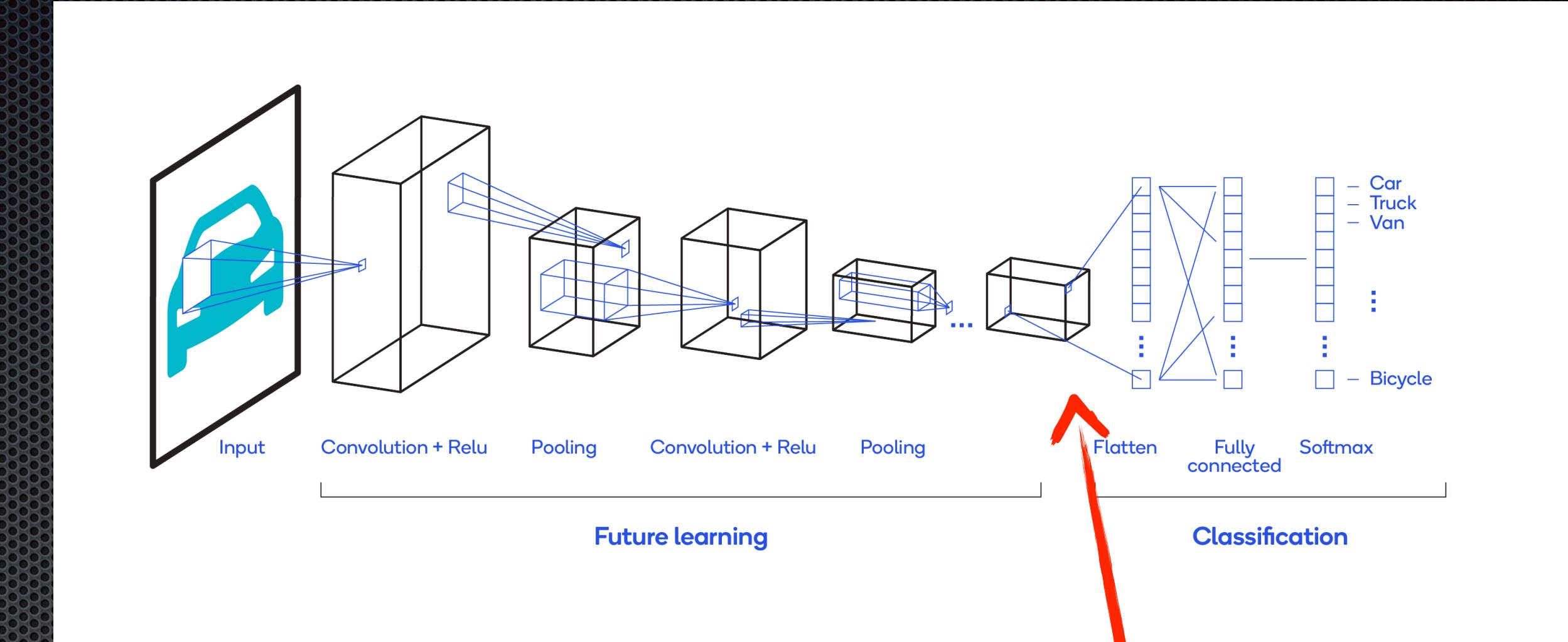
One can use the pretreated models on different images sizes by replacing the input and the output layer of the model with trainable layers and freeze all other layers. This process is called “**transfer learning**”

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 ?



Any convolution based model works as the following:

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

- 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

Shared
Weights/Activations
Loss Function

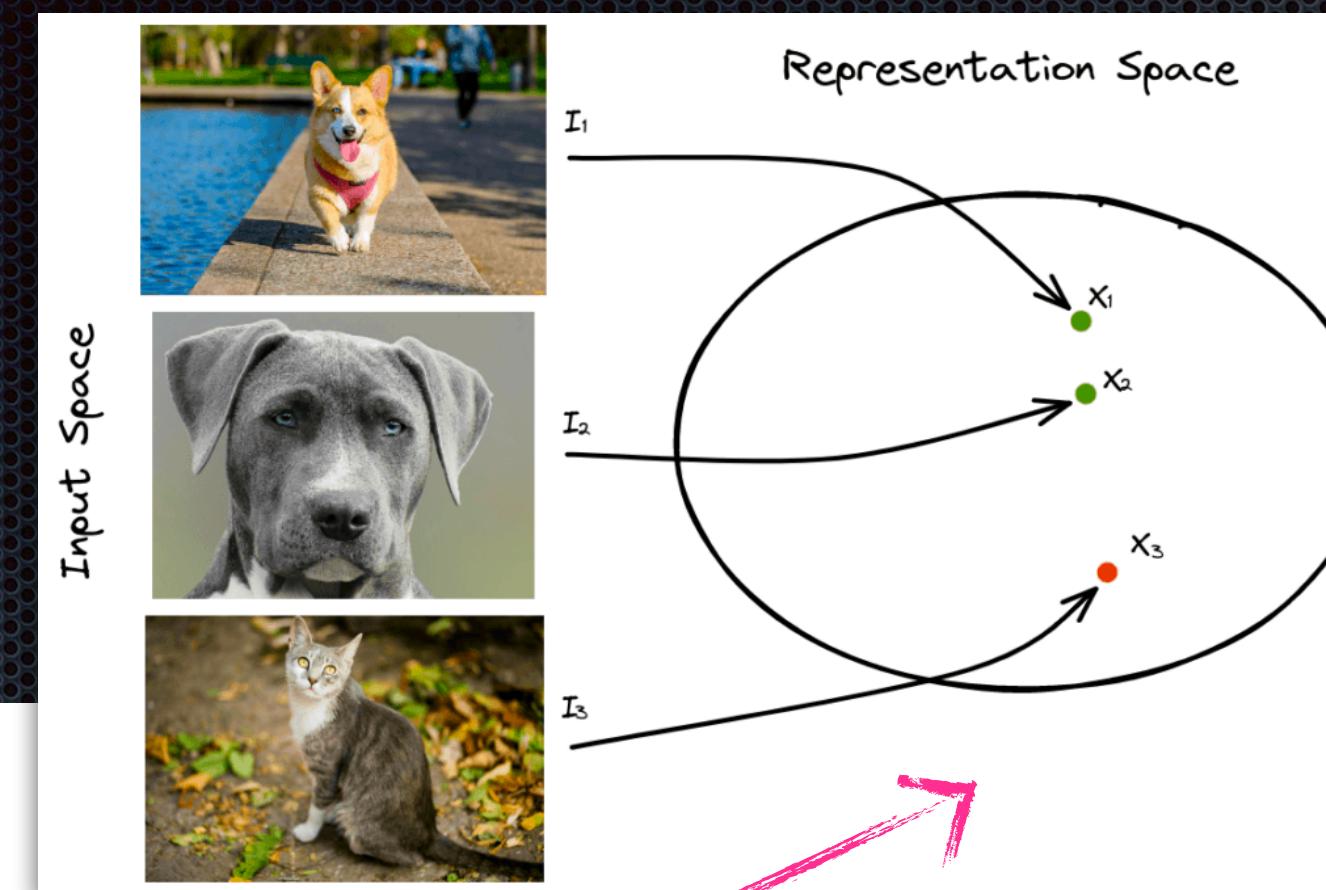
Stage 2

“Dog”

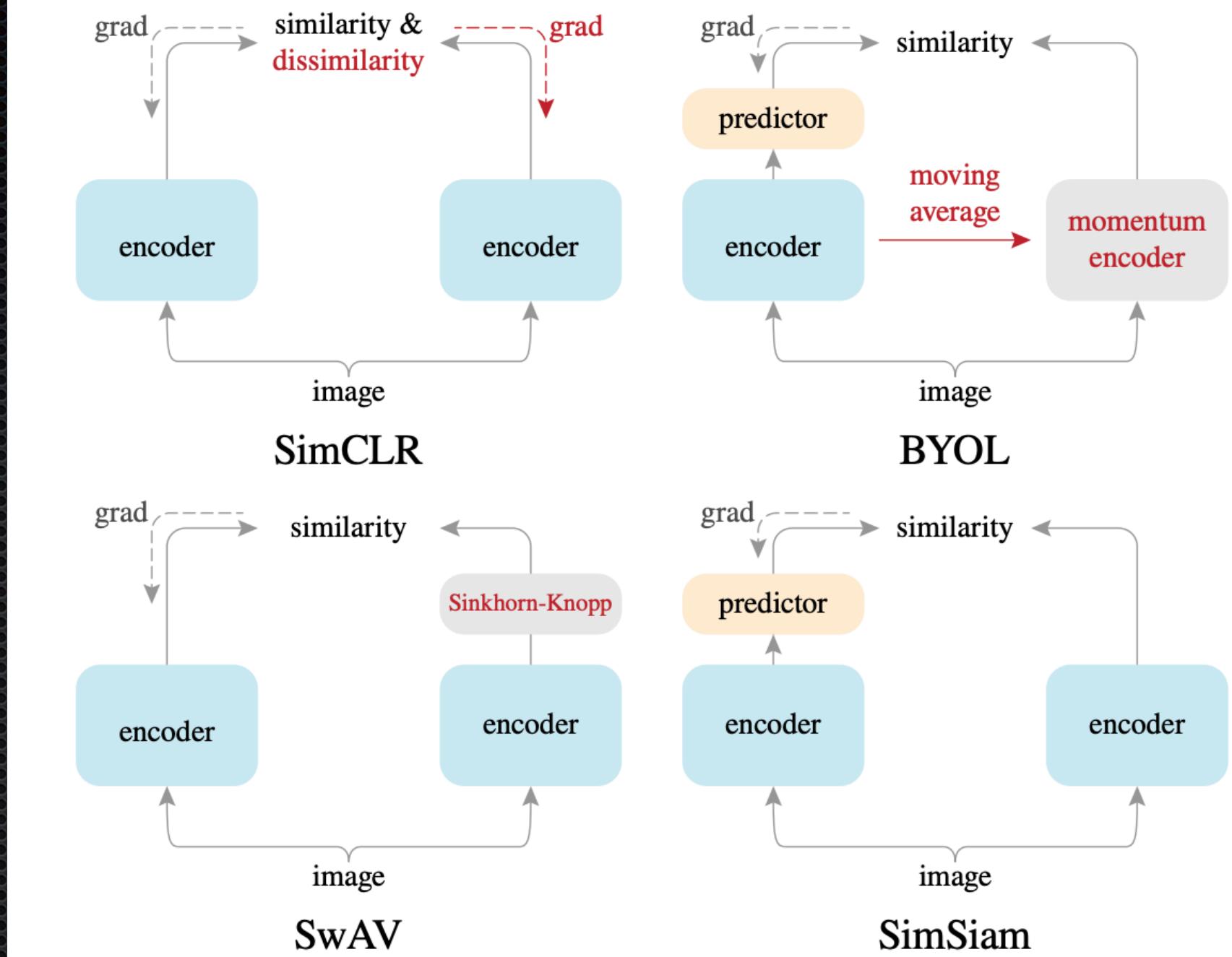
Softmax
1000-D
2048-D

128-D
2048-D

Contrastive

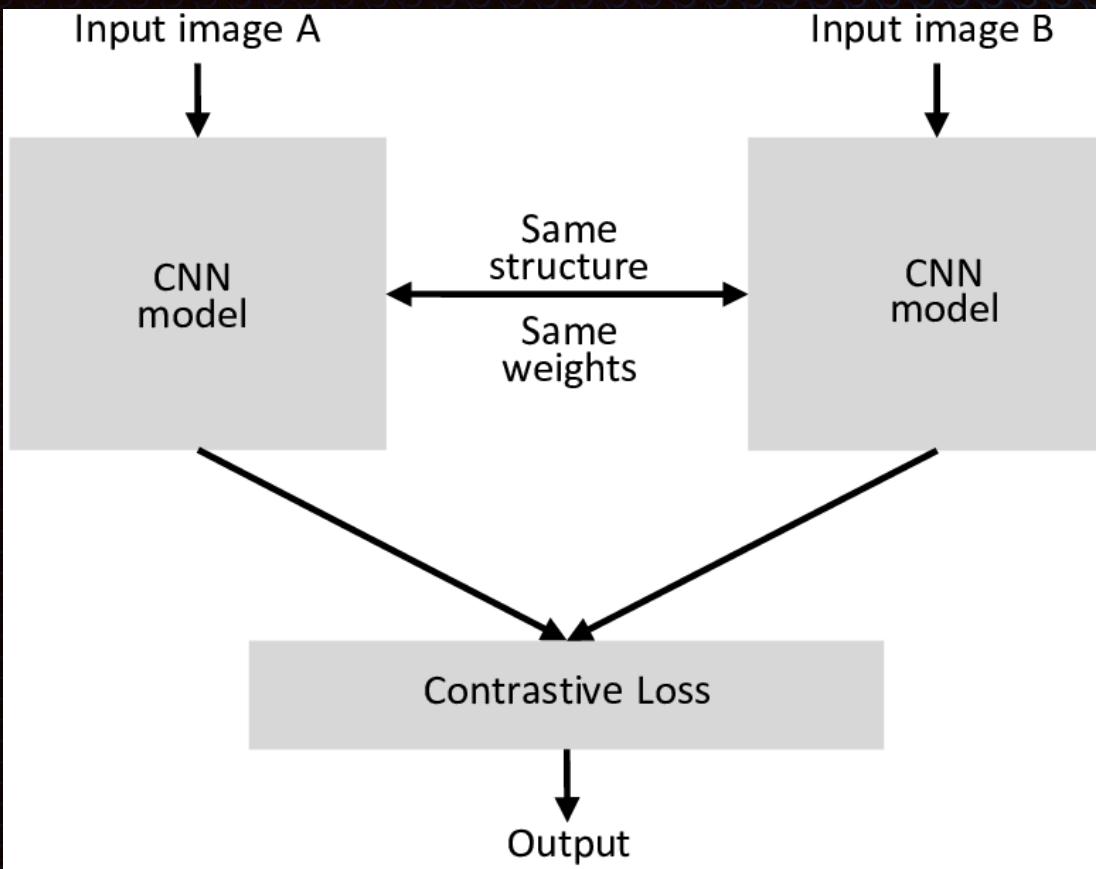


Twins encoder with shared weights, two steps training
with data augmentation

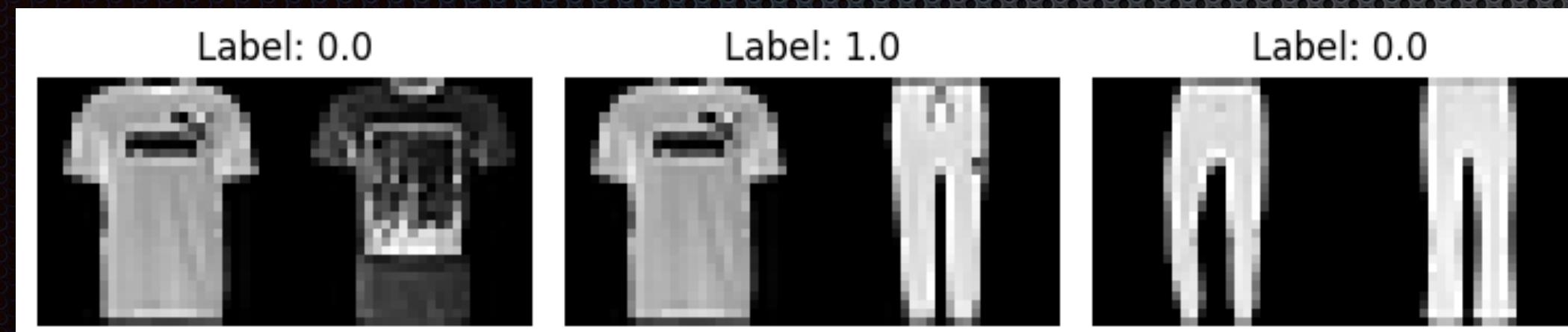


Wide list of contrastive models

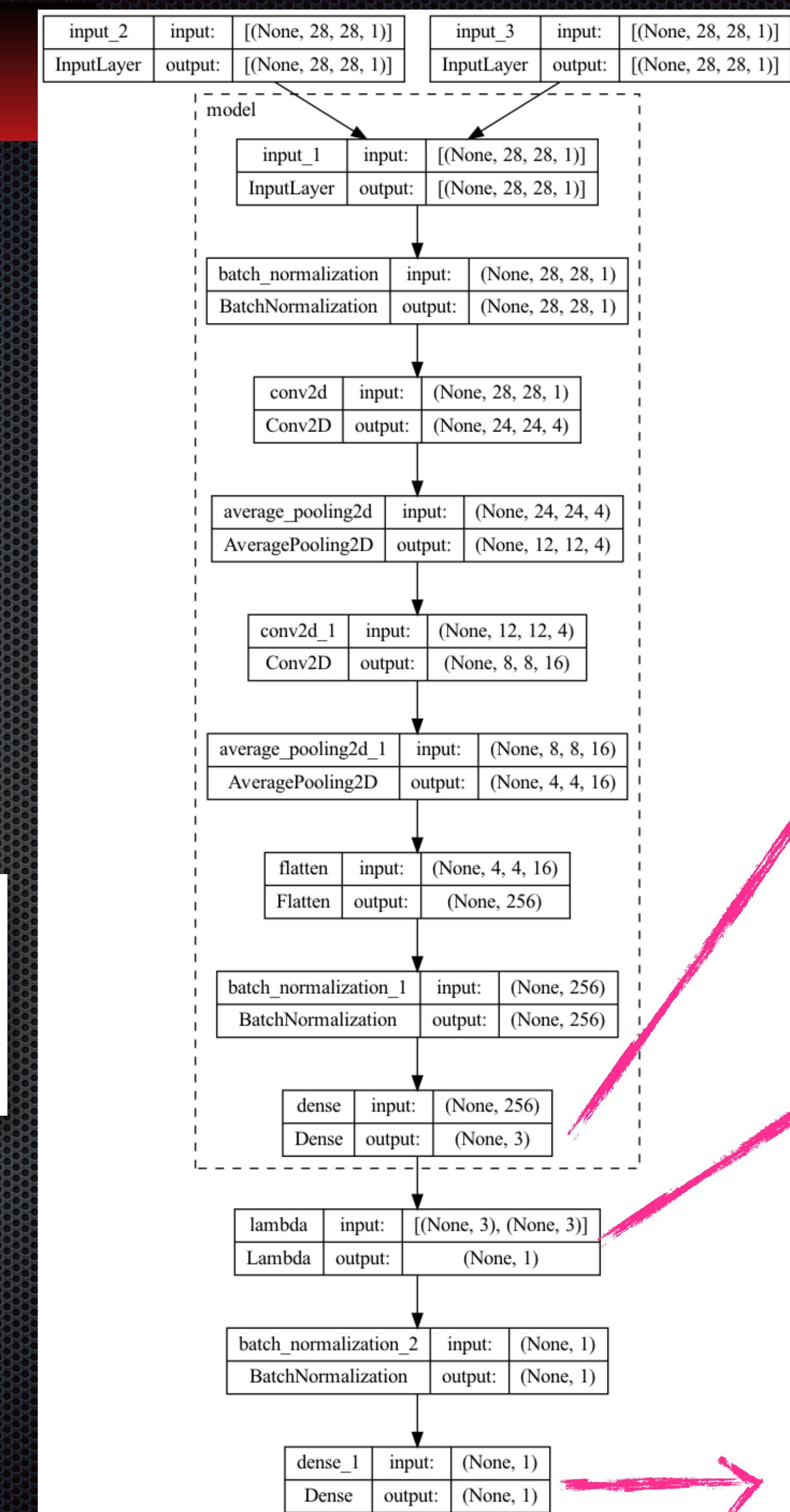
Contrastive learning



Supervised twins encoder with shared weights.

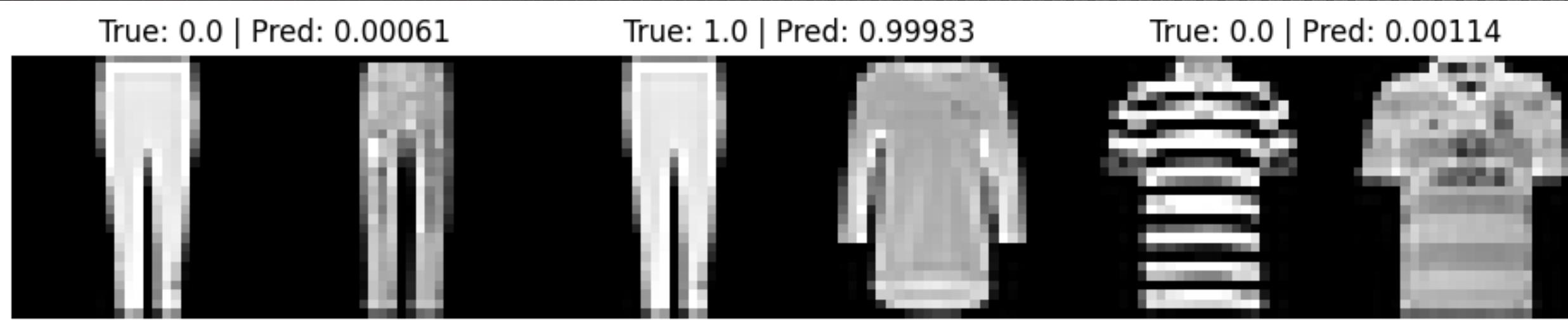


Contrastive Loss to. Minimize the Euclidean distance between similar objects and maximize the distance between different objects



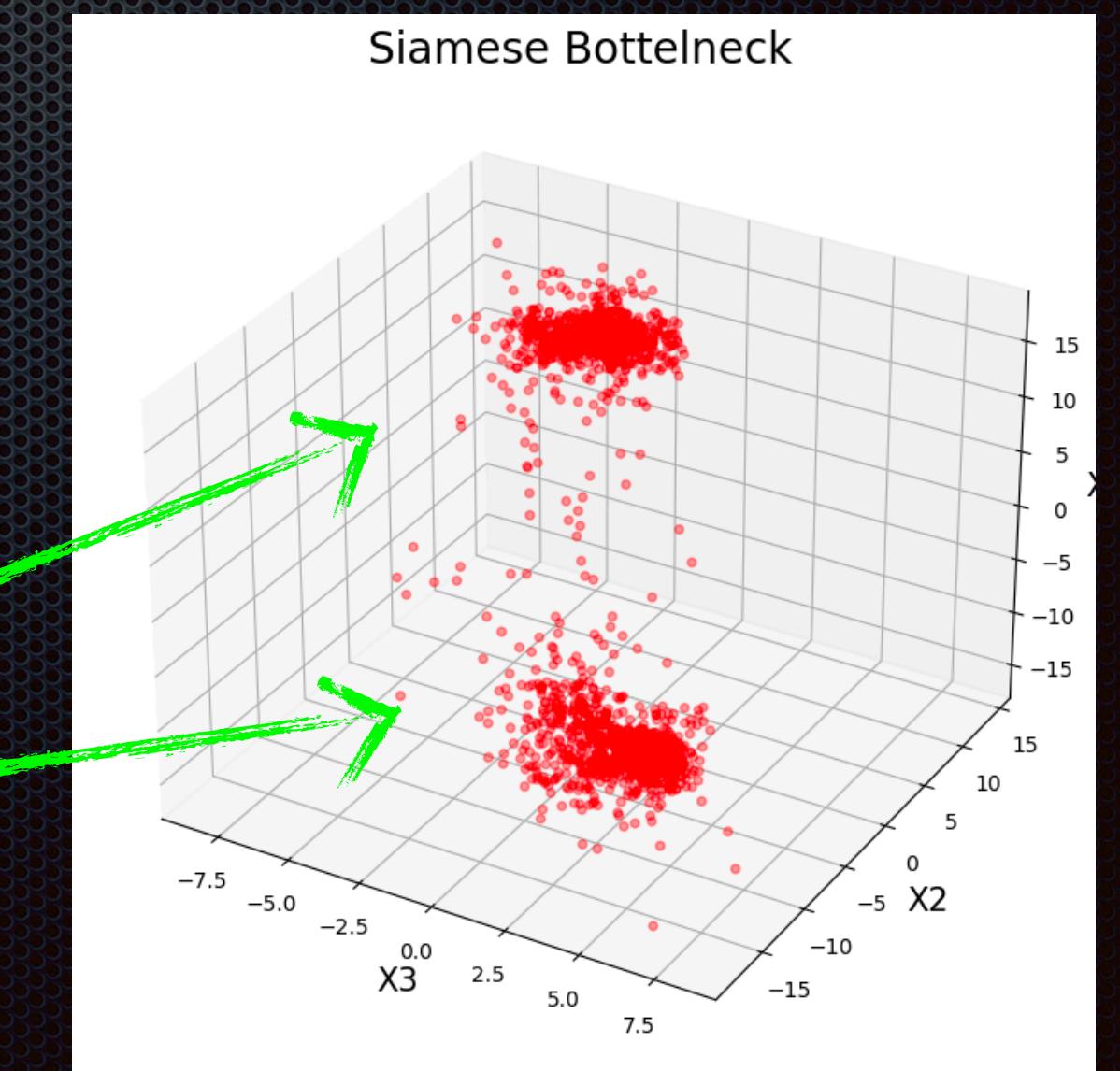
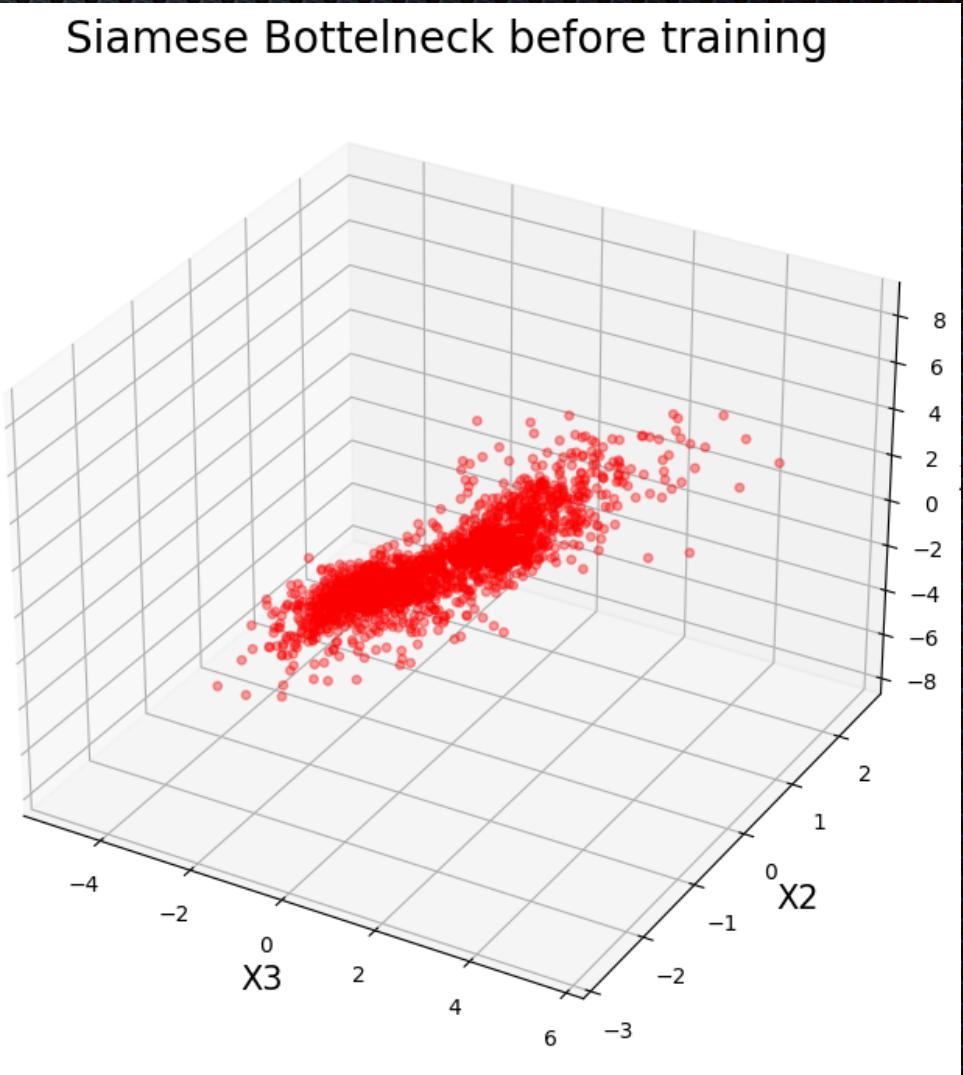
Contrastive learning

Siamese predictions

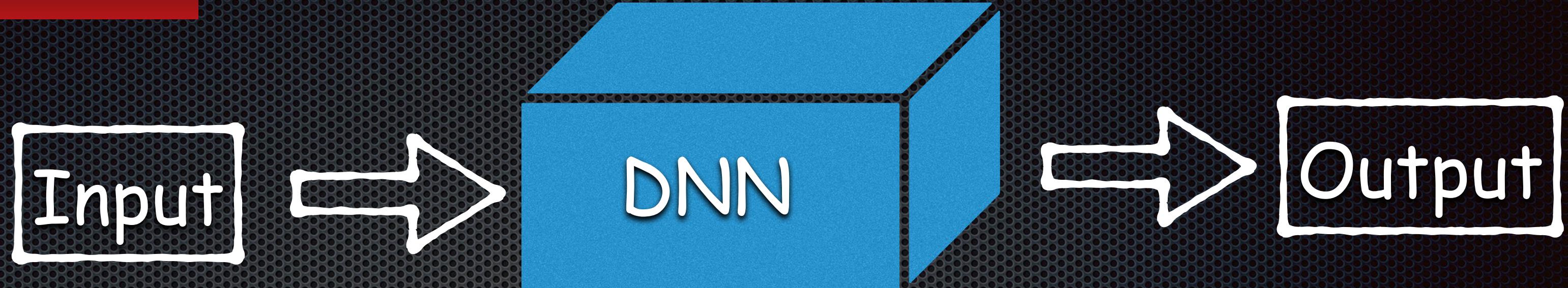


Siamese with one step training,
cannot be used for classification !!

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.

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

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

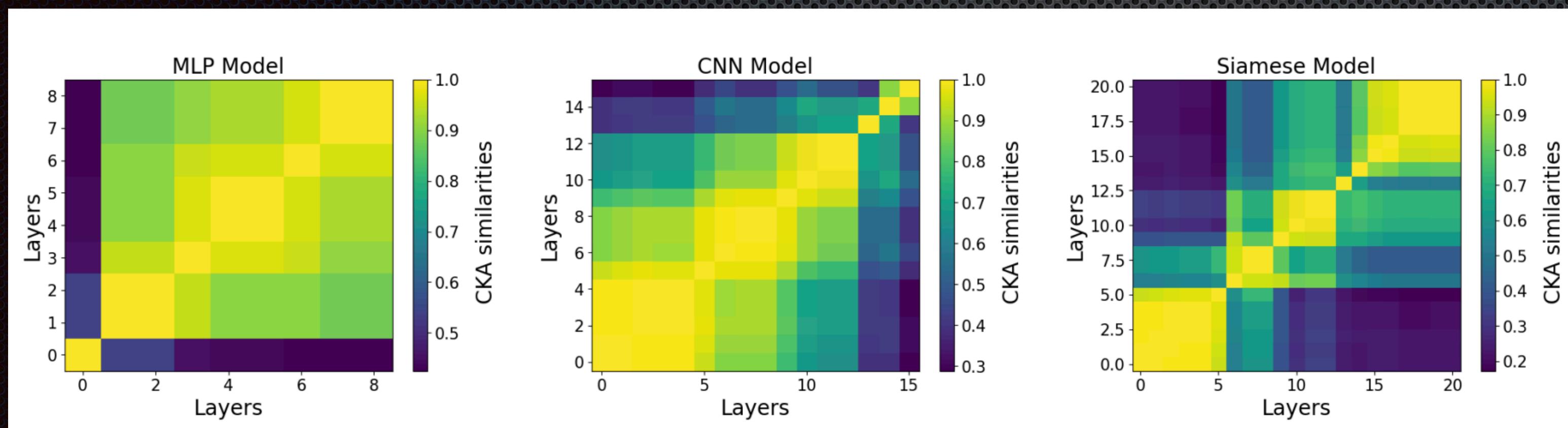
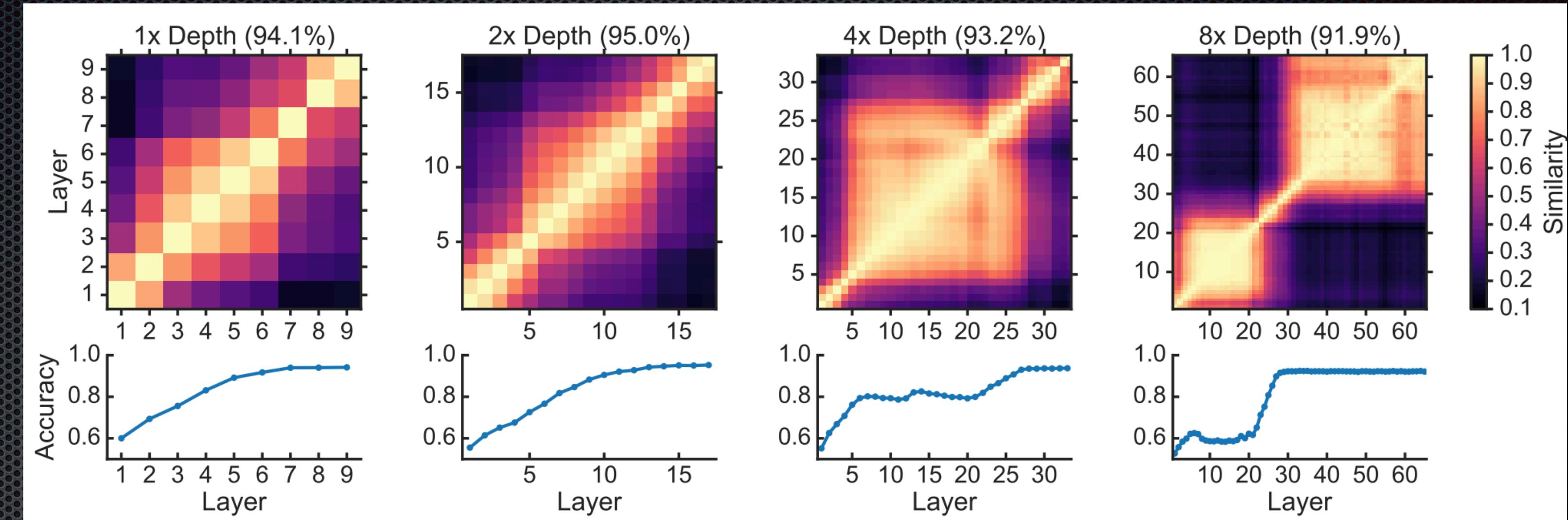
$$M = XX^\top N = YY^\top$$

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

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

CKA similarity

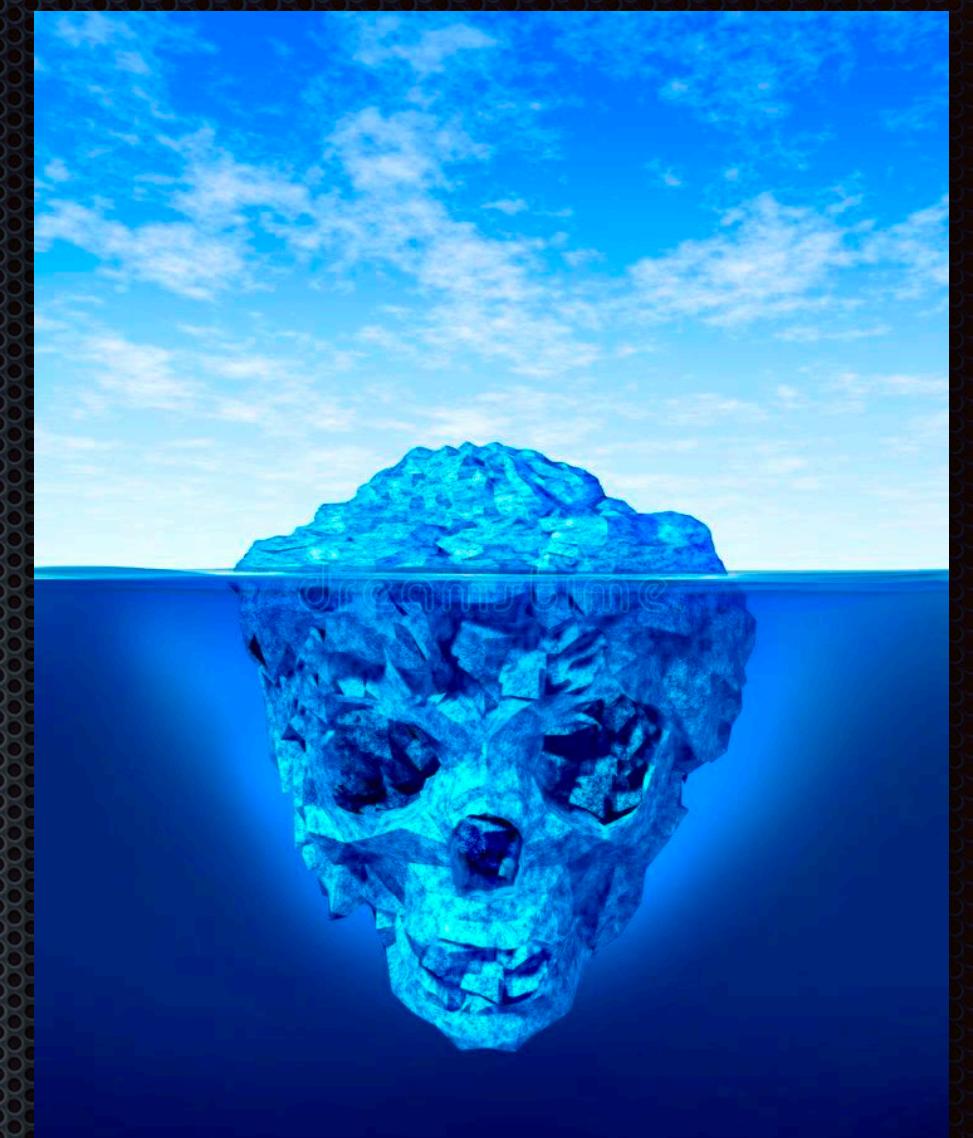
arxiv:1905.00414



All models trained on
CIFAR10 data sets

Image sparsity (hidden danger)

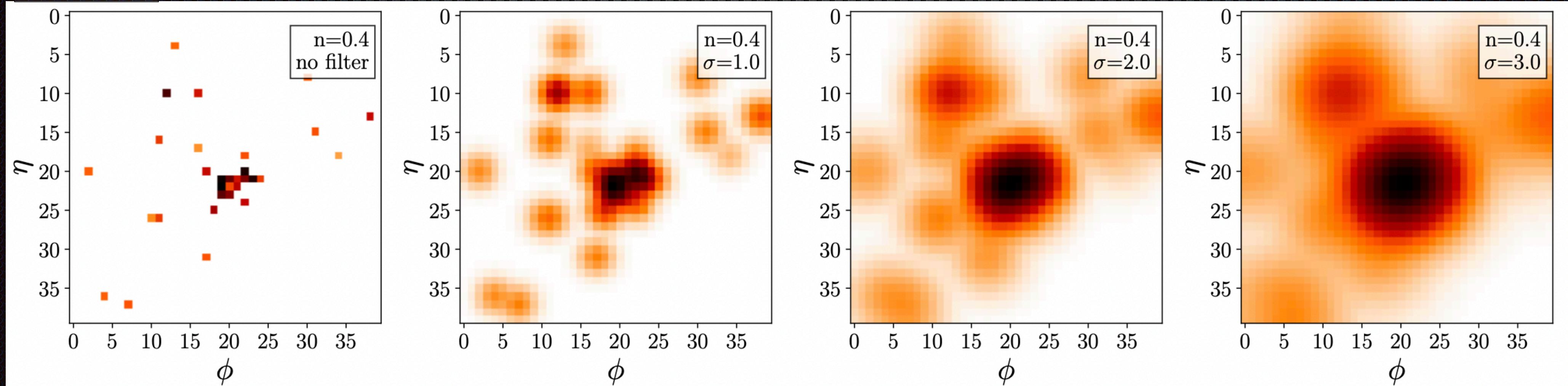
Sparsity is large number of zero pixels which are randomly distributed



- 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, et al
arXiv:2202.00686

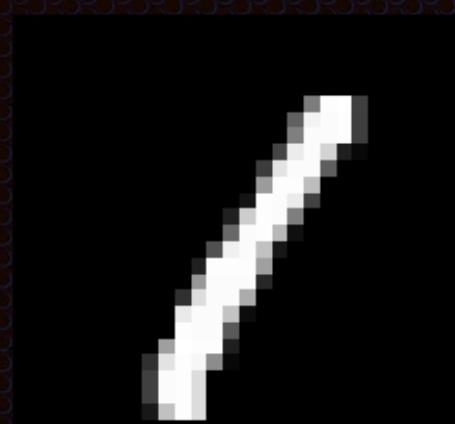


Sparse CNN

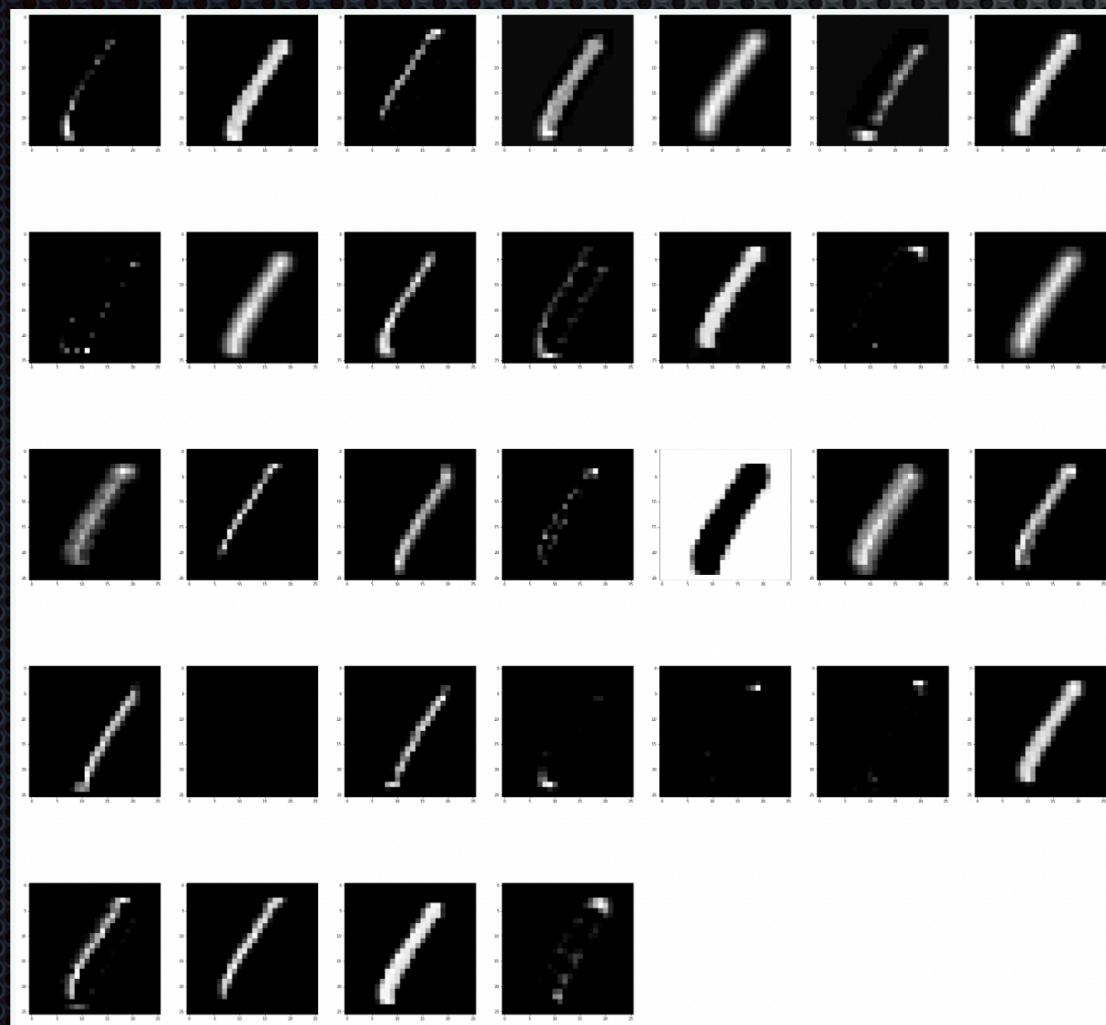
Regularize the layers activity : turning off kernels that capture low level information

Example:

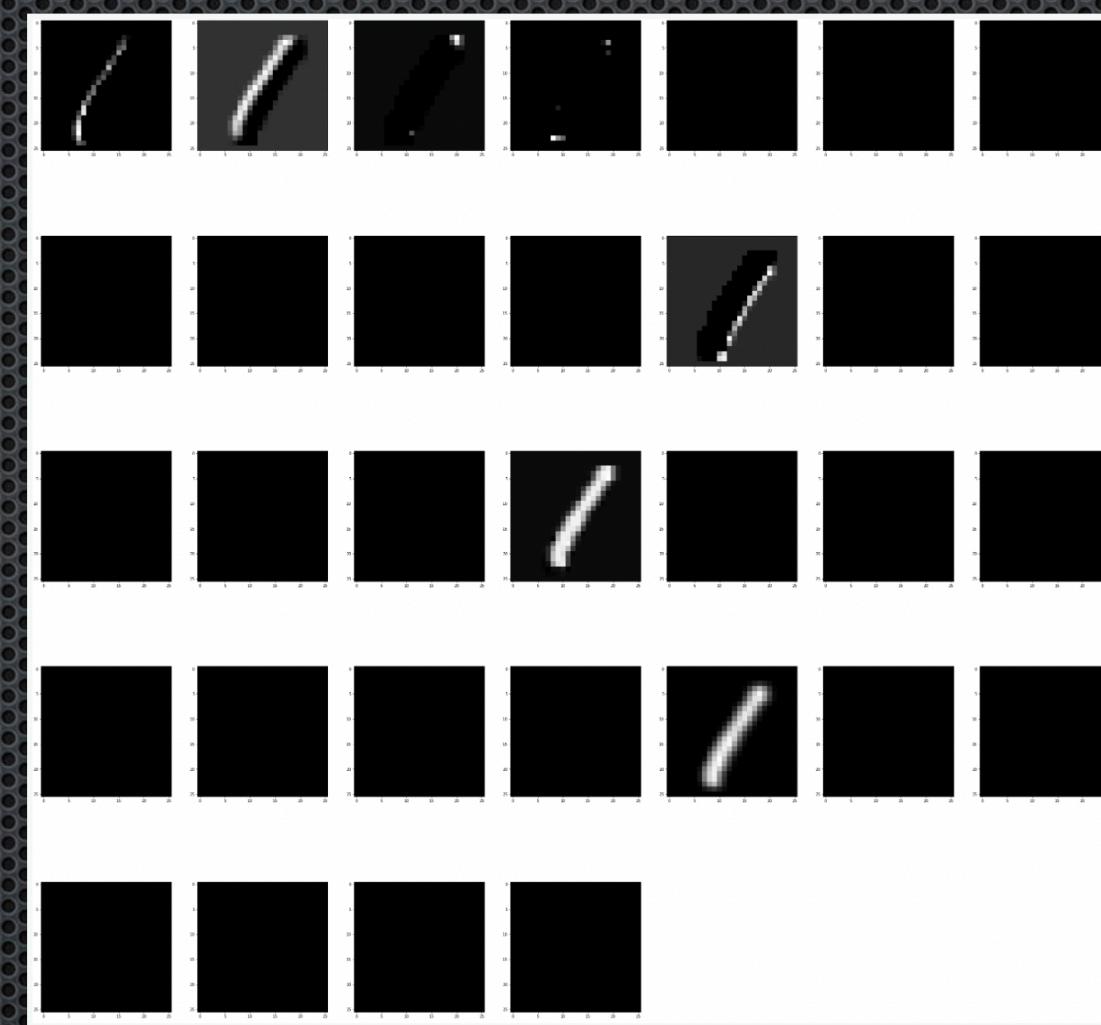
Output of the first Conv. Layer with 32 kernels after training on MNIST data set.



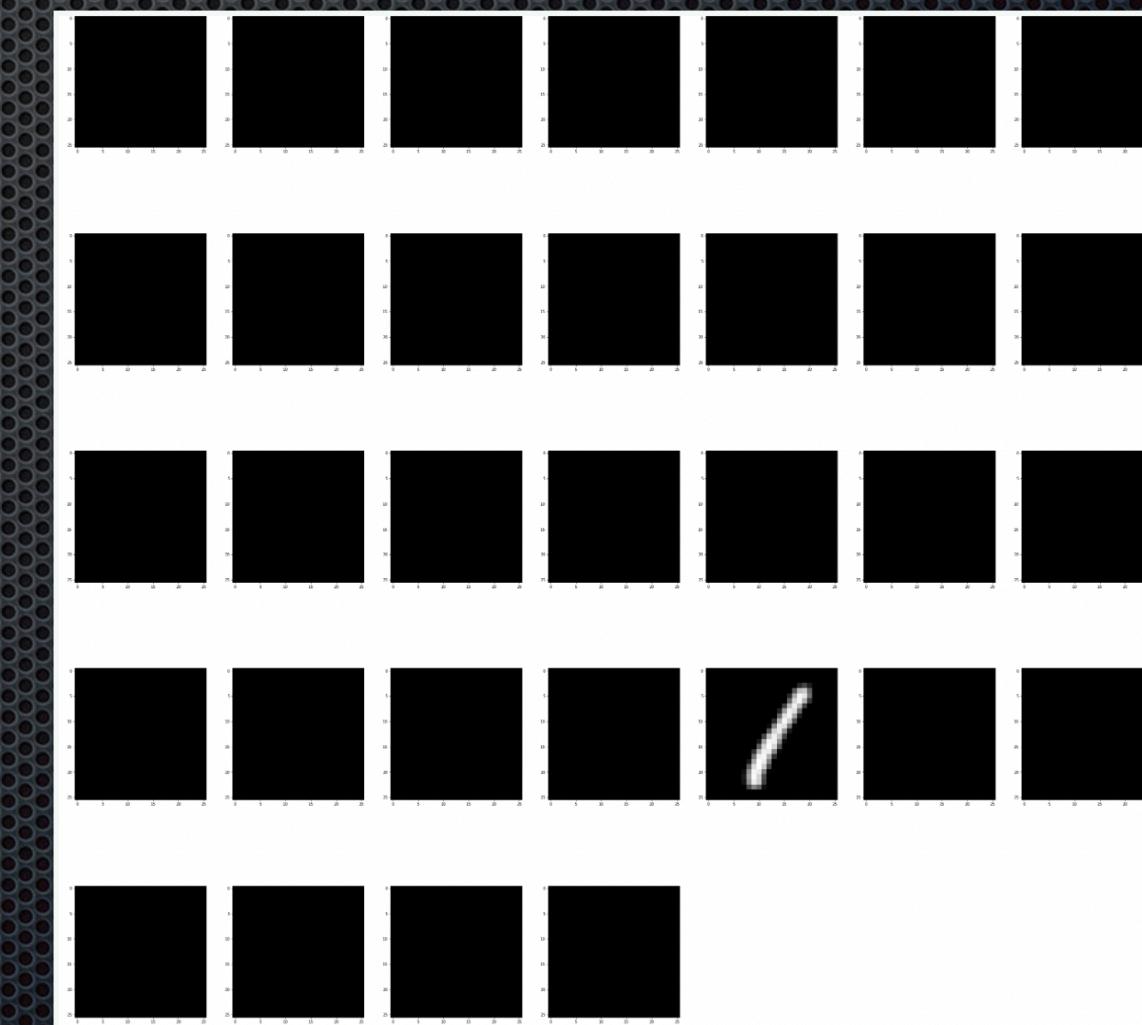
$$\alpha = 0$$



$$\alpha = 0.01$$



$$\alpha = 0.5$$



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...