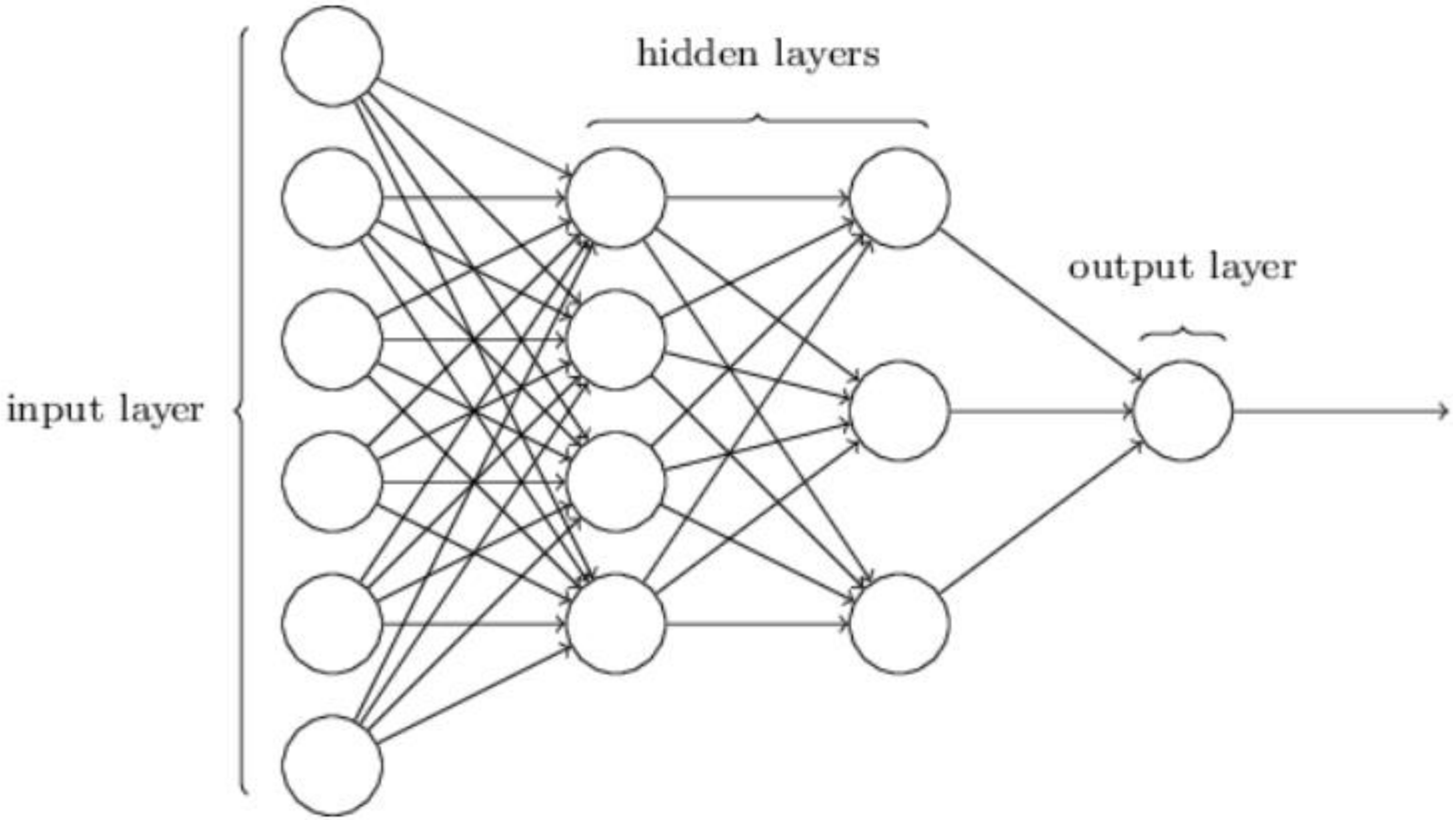
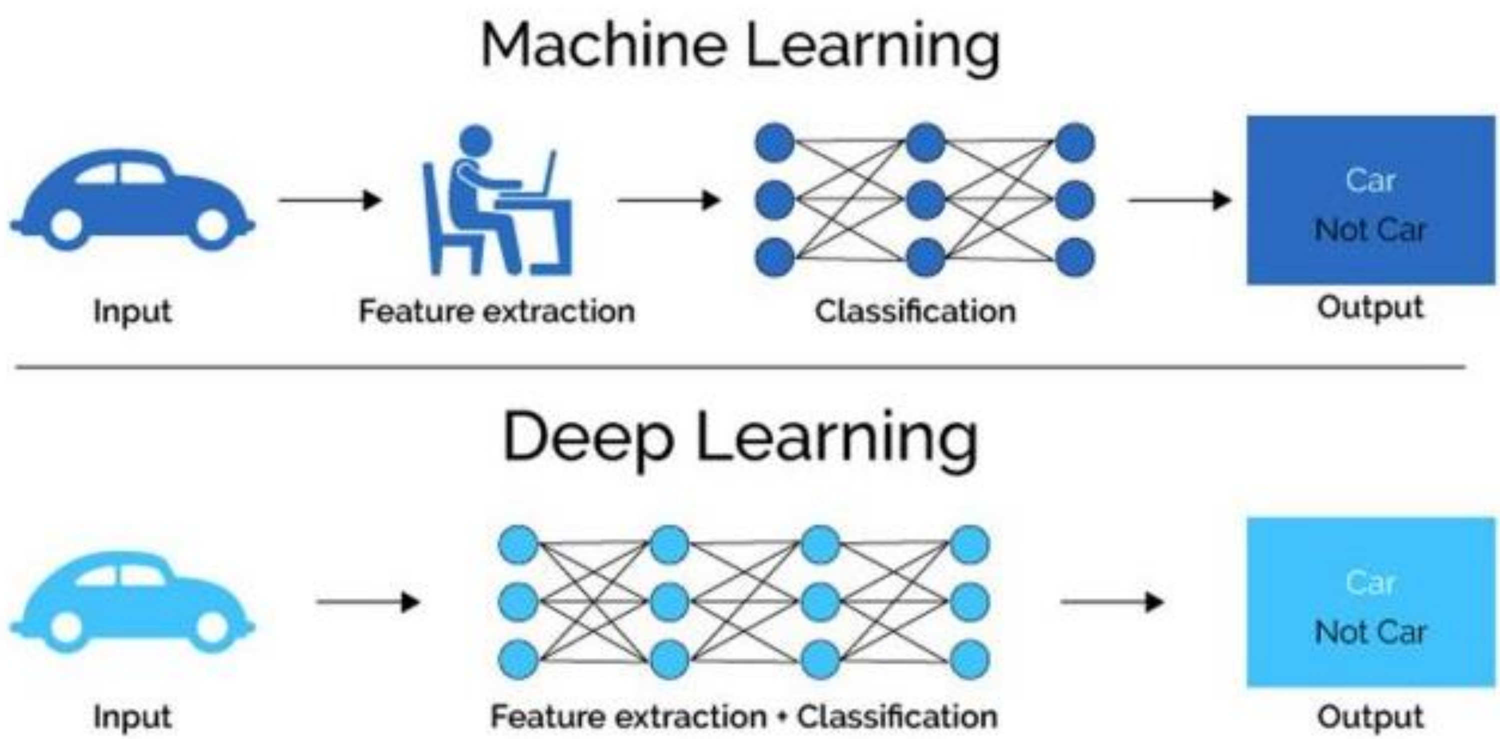
Convolutional Neural Network

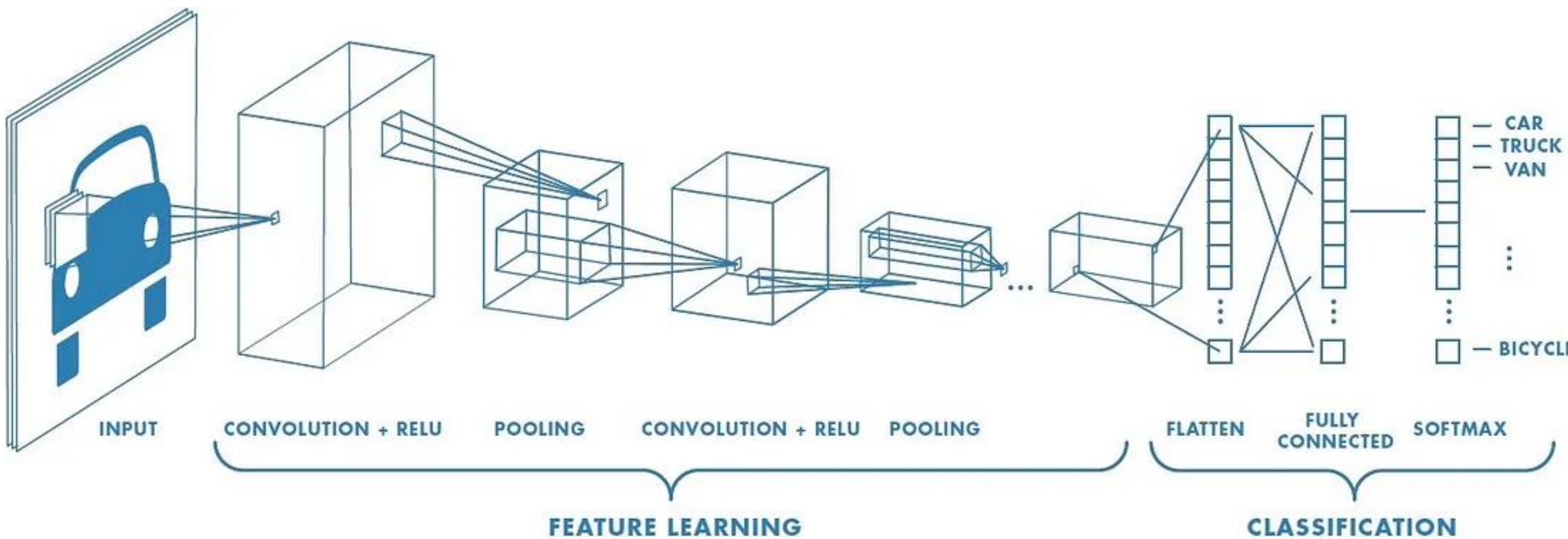


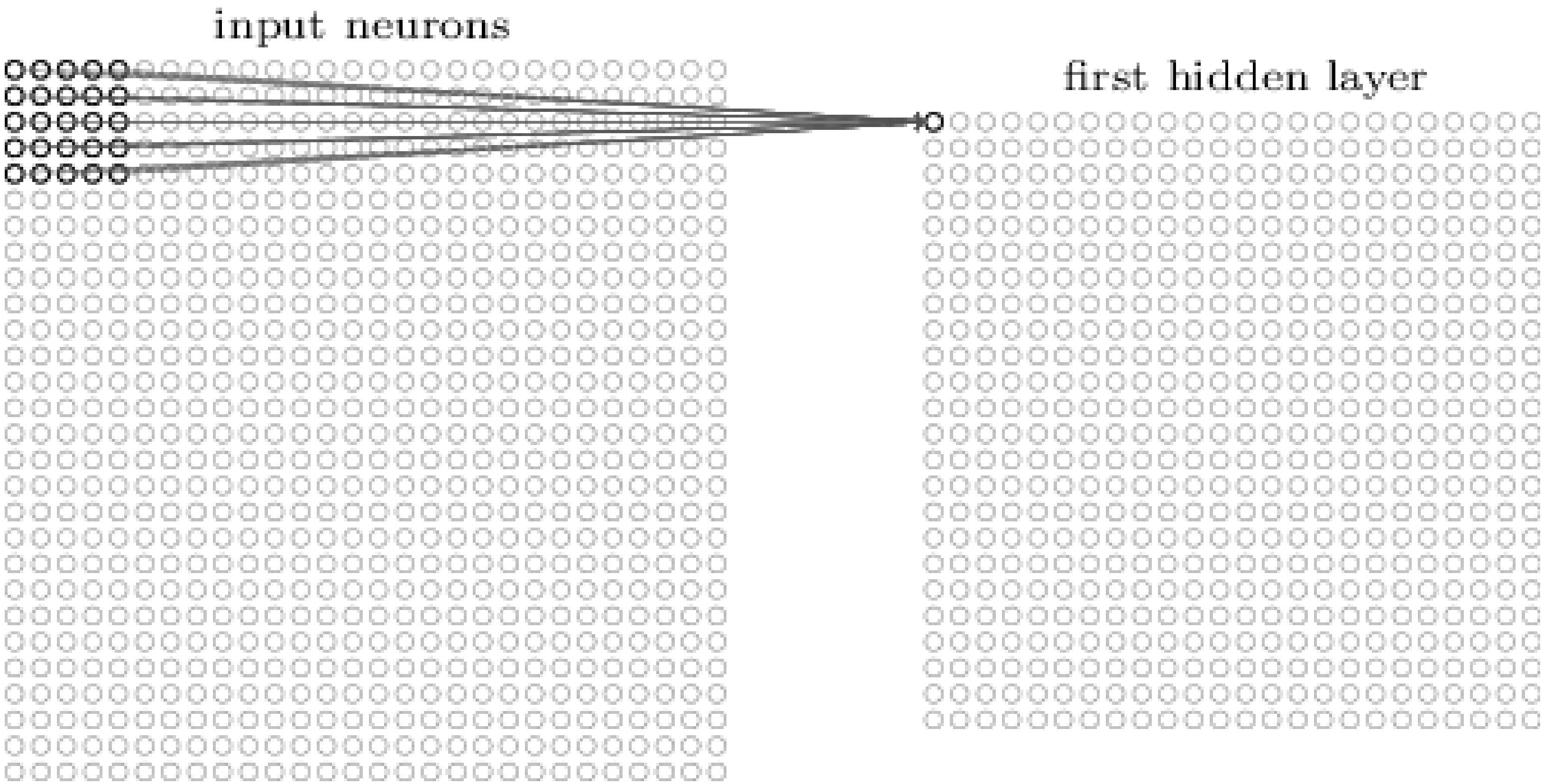
200*200*3 = 120 000 weights !



www.takethiscourse.net

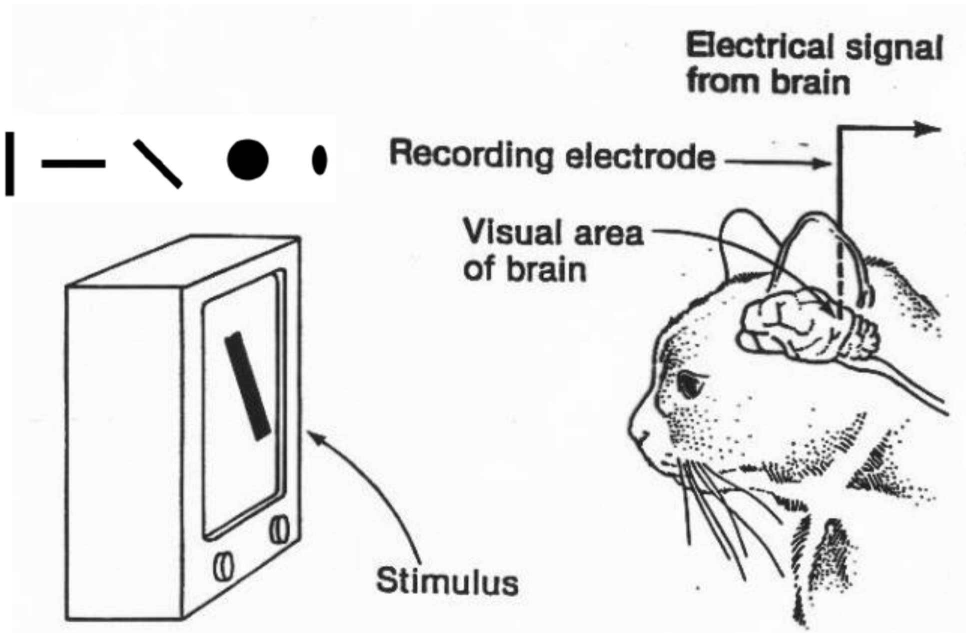
Convolutional Neural Network

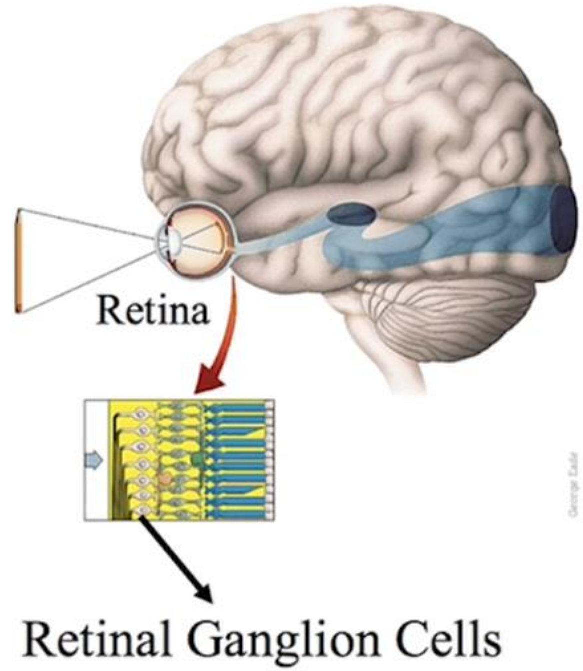




http://neuralnetworksanddeeplearning.com/chap6.html

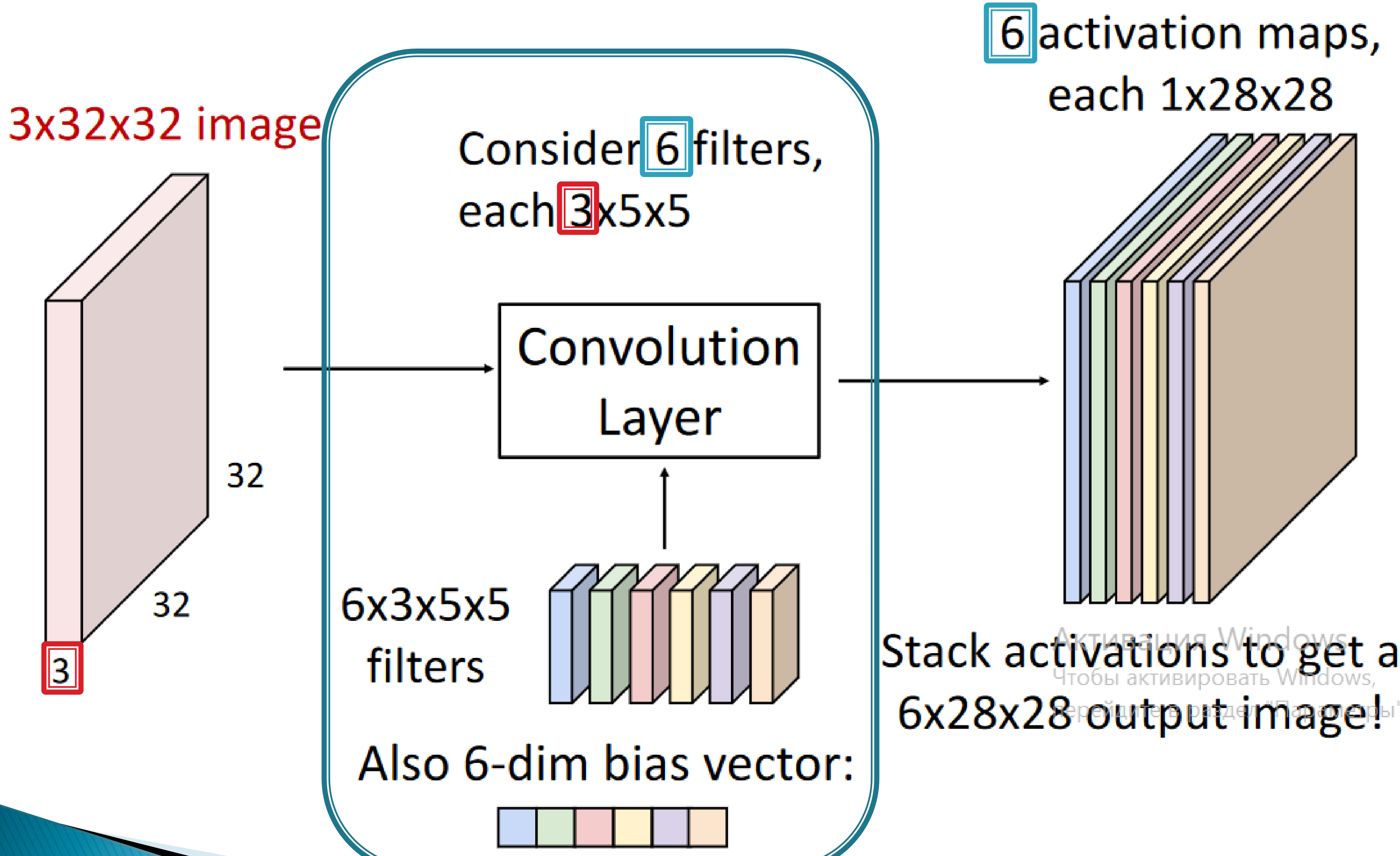
Hubel & Wiesel Experiment



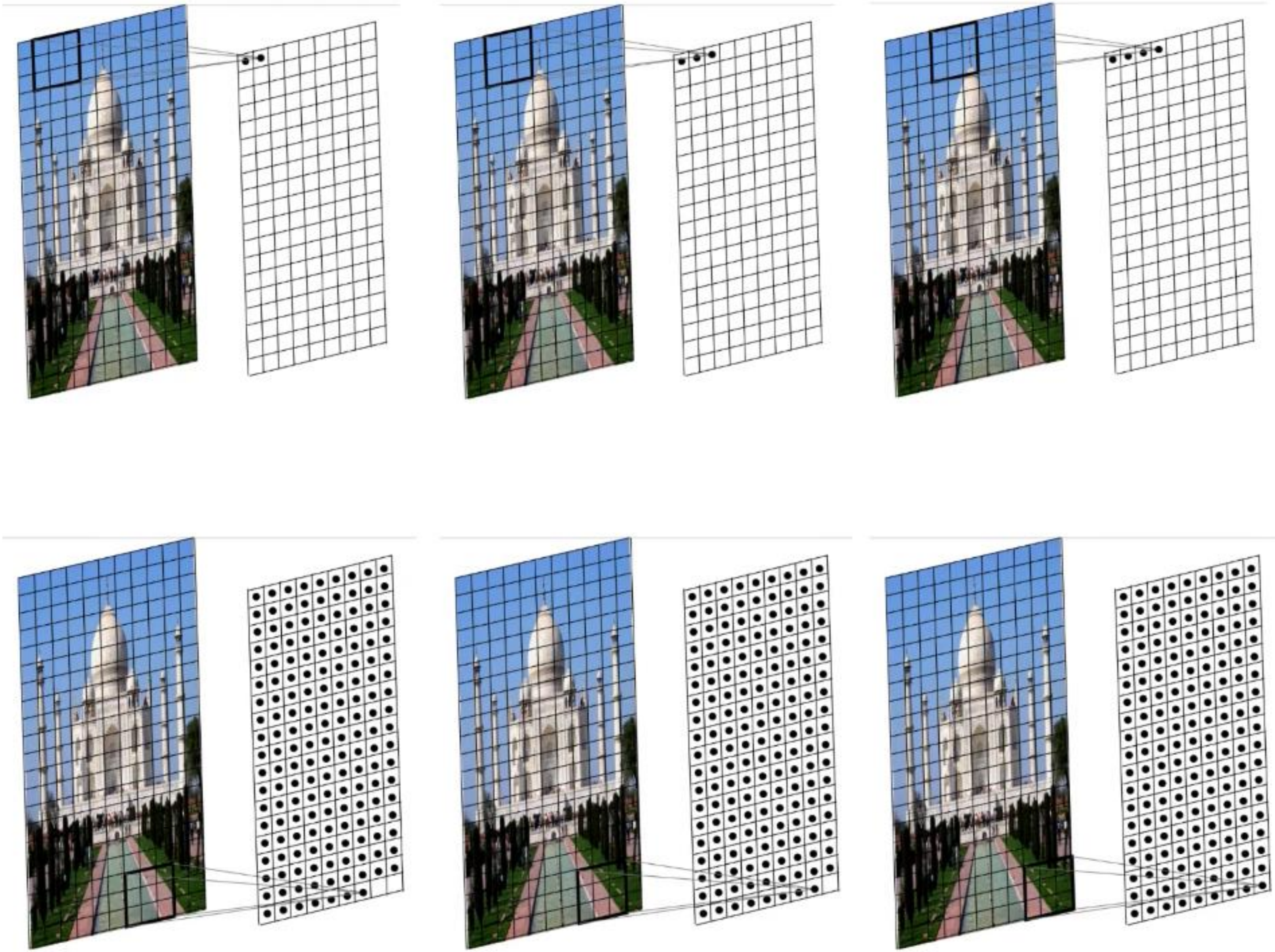


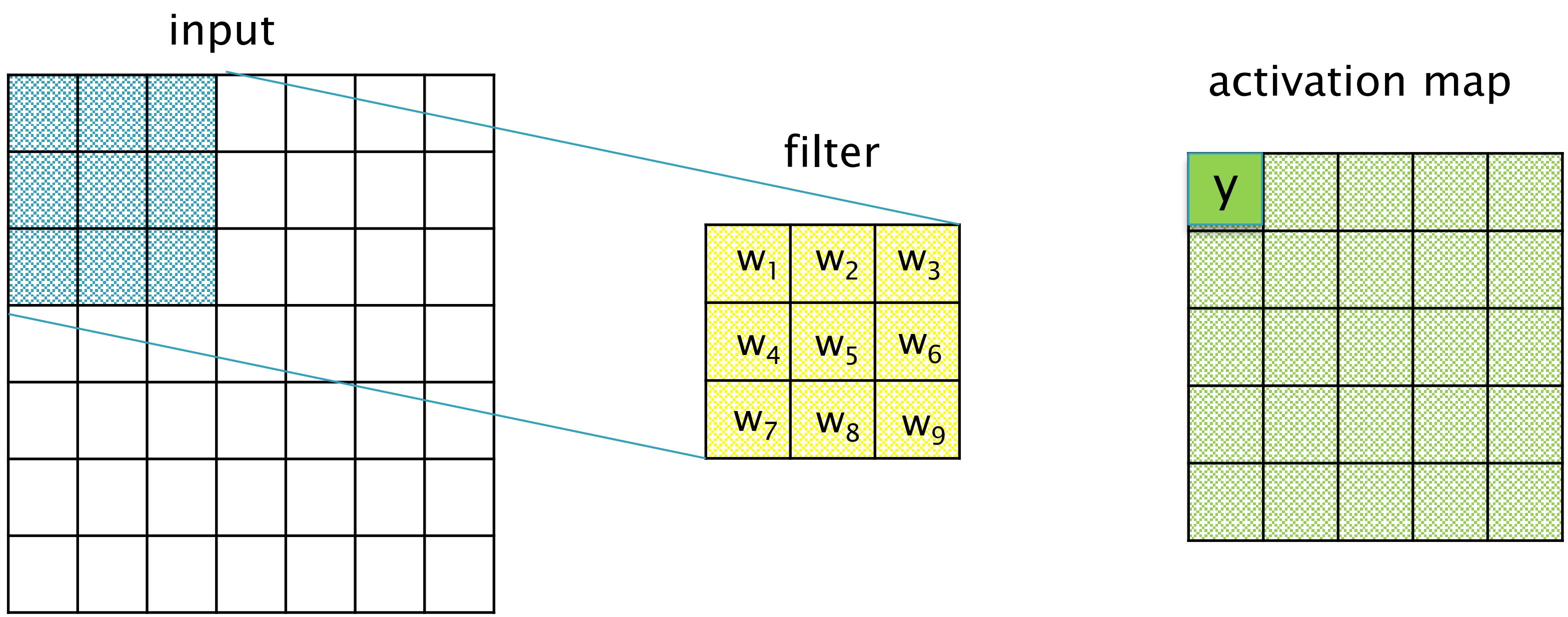
https://www.ncbi.nlm.nih.gov/pmc/articles/PMC1359523/pdf/jphysiol01247-0121.pdf

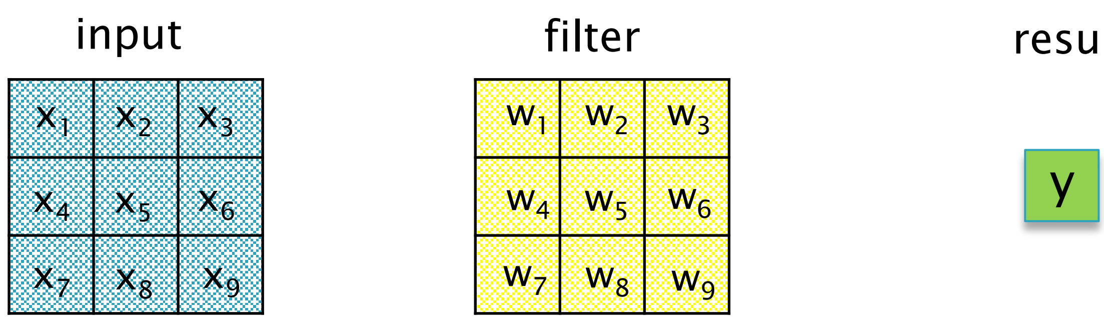
Convolutional layer



http://cs231n.stanford.edu/slides/2022/lecture_5_ruohan.pdf



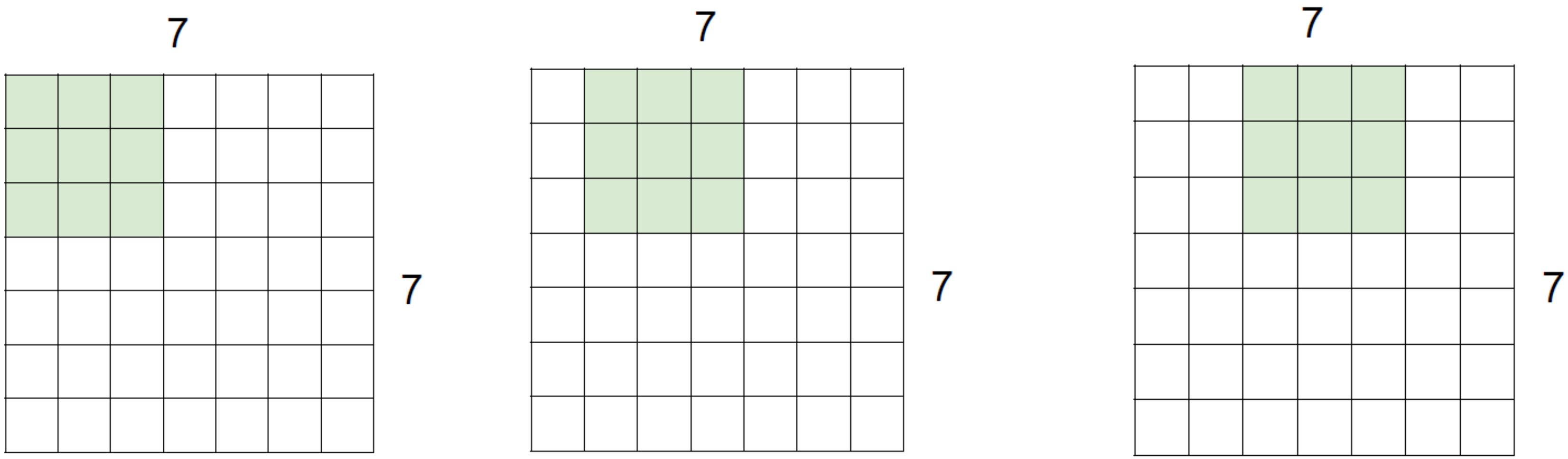




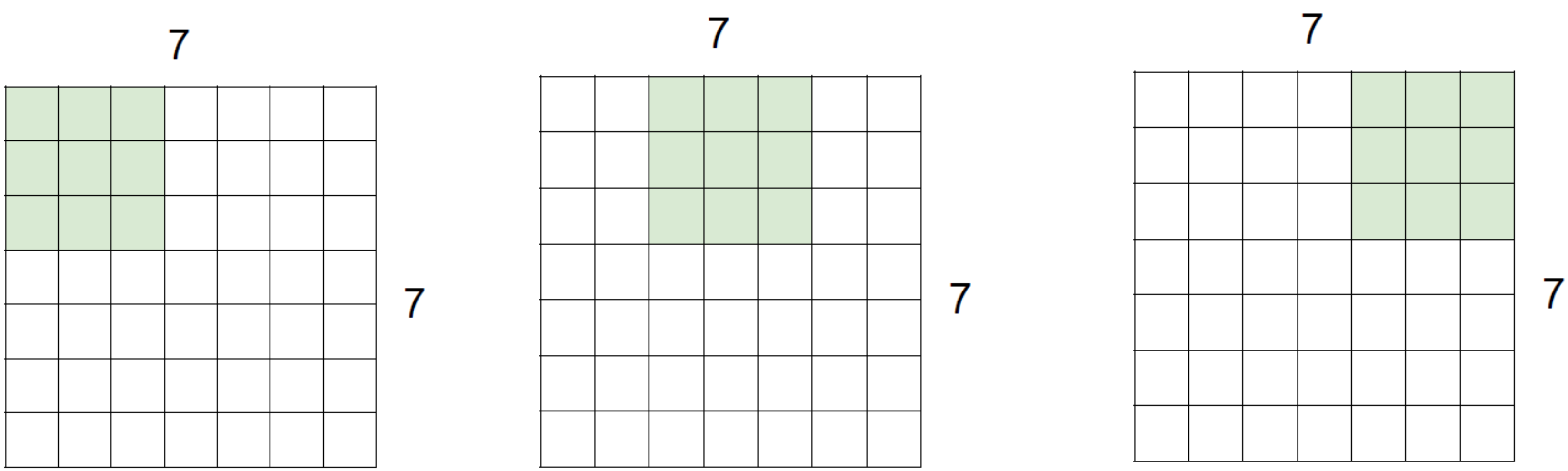
$$y = x_1 * w_1 + x_2 * w_2 + x_3 * w_3 + x_4 * w_4 + \ldots x_9 * w_9$$

http://neuralnetworksanddeeplearning.com/chap6.html

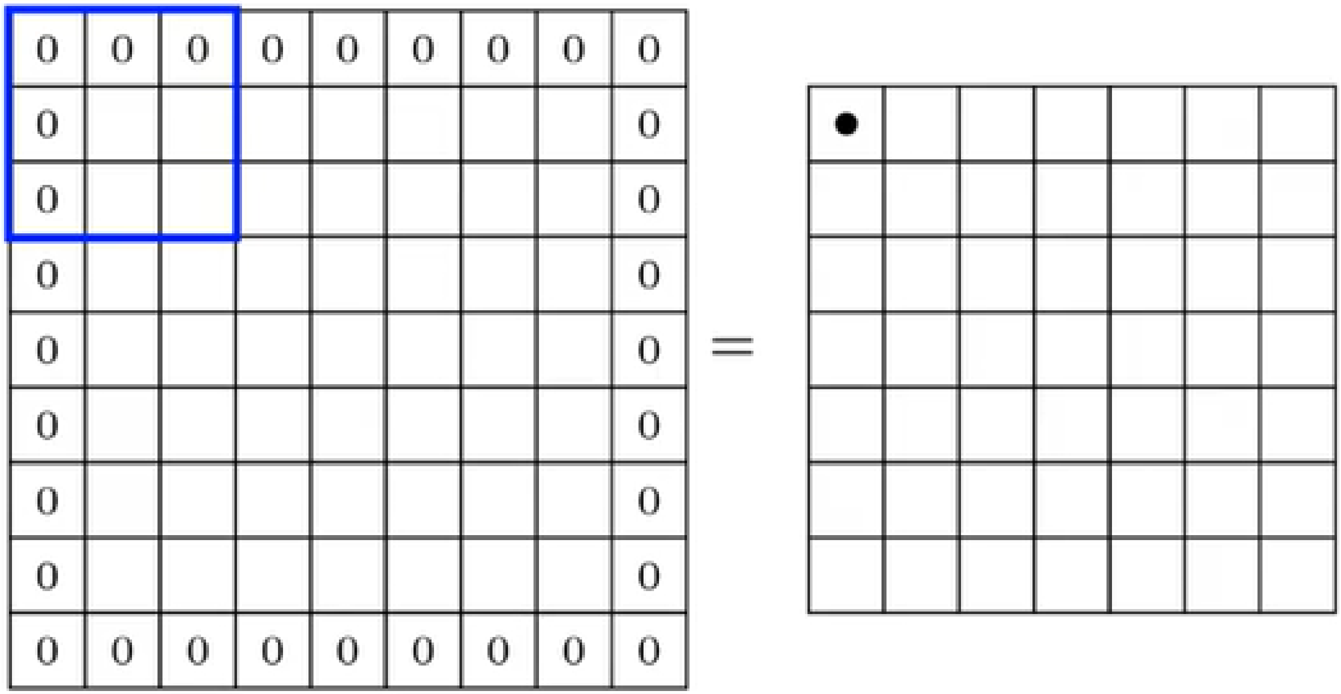
kernel size = 3, stride = 1



kernel size = 3, stride = 2



zero padding



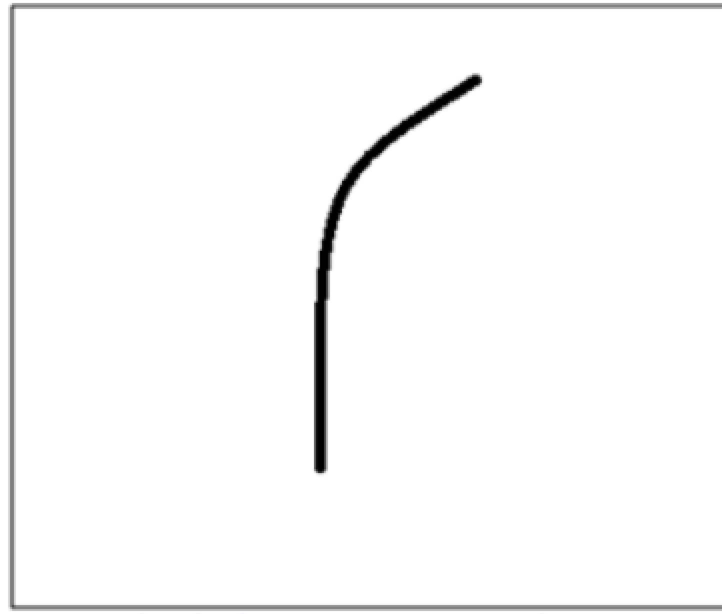
spatial size of the output volume

$$(W - F + 2P)/S + 1$$

- W – input volume size
- F – receptive field (filter) size
- S – stride
- P – zero padding

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

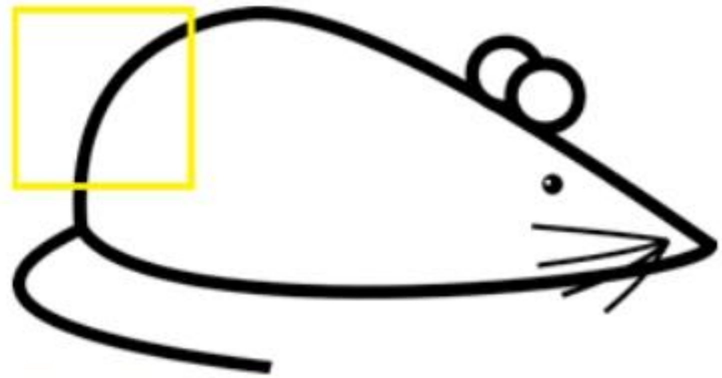
Pixel representation of filter



Visualization of a curve detector filter

https://adeshpande3.github.io/A-Beginner's-Guide-To-Understanding-Convolutional-Neural-Networks/

Original image



Visualization of the filter on the image

https://adeshpande3.github.io/A-Beginner's-Guide-To-Understanding-Convolutional-Neural-Networks/

Visualization of the receptive field

0	0	0	0	0	0	30
0	0	0	0	50	50	50
0	0	0	20	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0

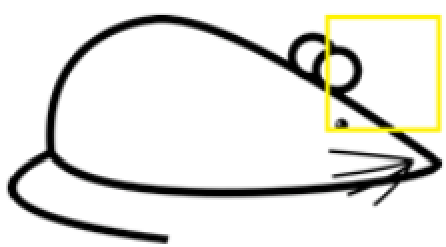
Pixel representation of the receptive field

*

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter

Multiplication and Summation = (50*30)+(50*30)+(50*30)+(20*30)+(50*30) = 6600 (A large number!)



Visualization of the filter on the image

0	0	0	0	0	0	0
0	40	0	0	0	0	0
40	0	40	0	0	0	0
40	20	0	0	0	0	0
0	50	0	0	0	0	0
0	0	50	0	0	0	0
25	25	0	50	0	0	0

Pixel representation of receptive field

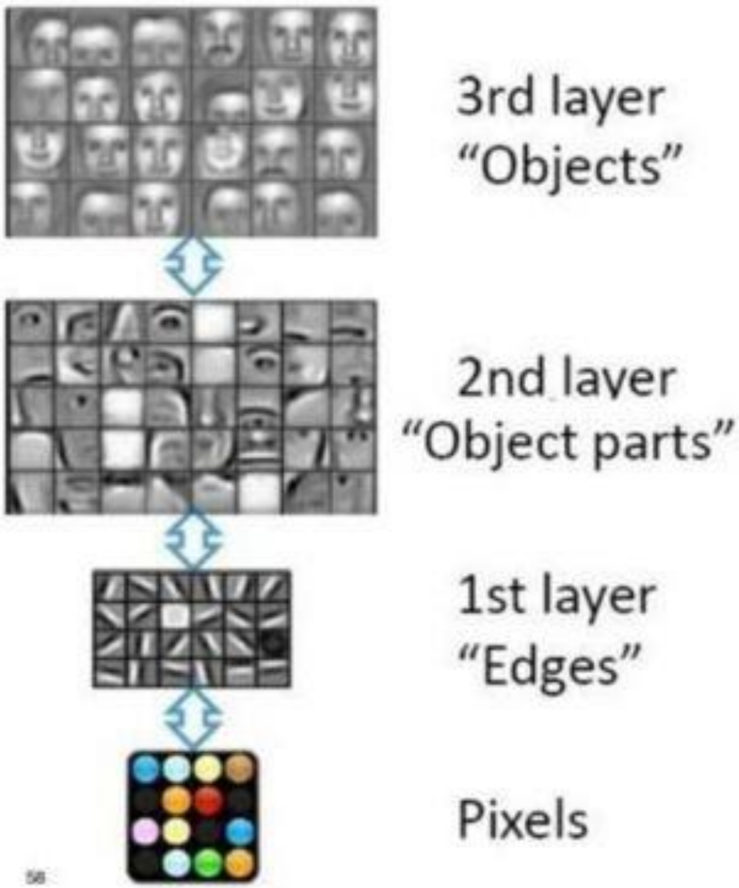
*

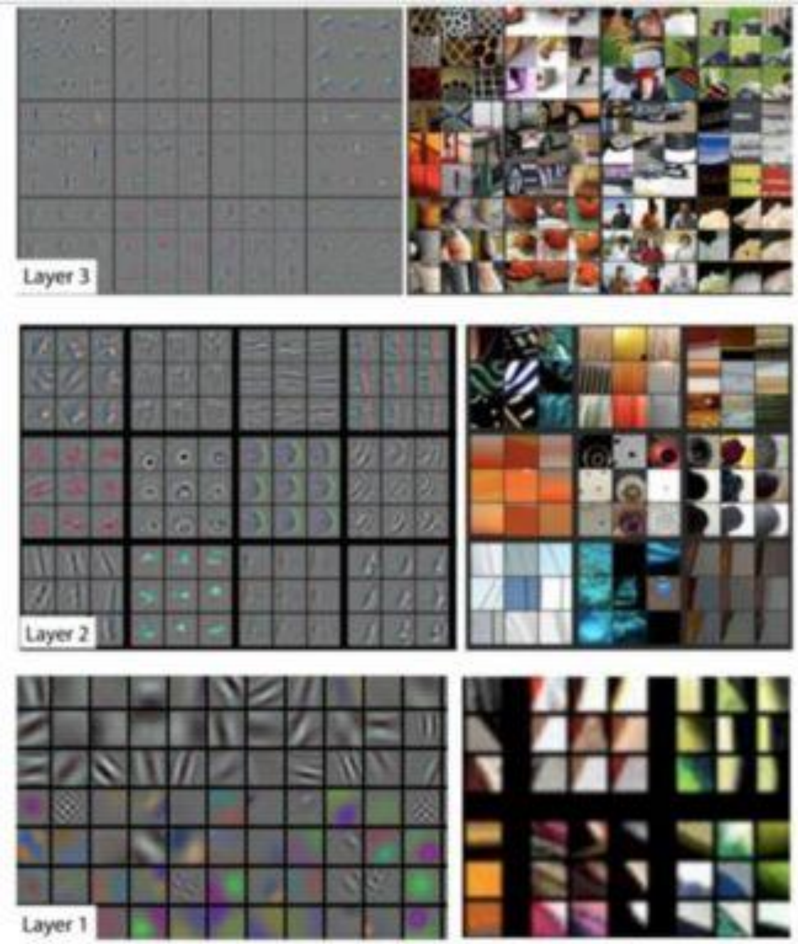
0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

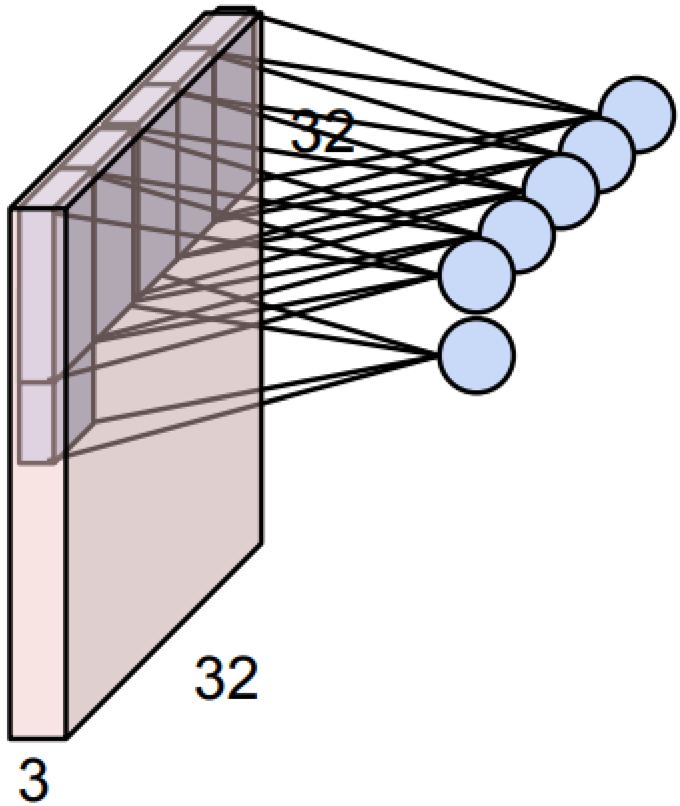
Pixel representation of filter

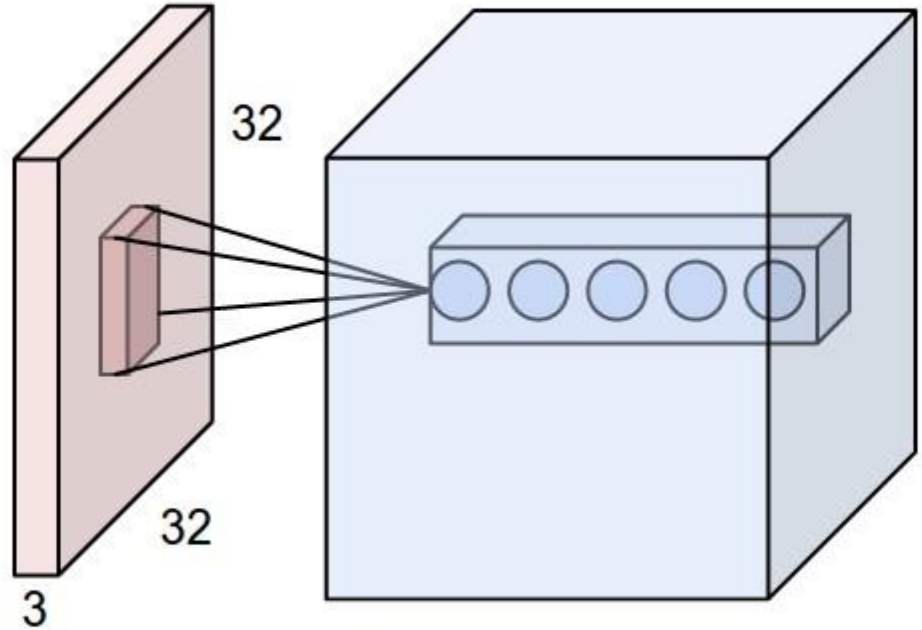
Multiplication and Summation = 0

https://adeshpande3.github.io/A-Beginner's-Guide-To-Understanding-Convolutional-Neural-Networks/


3rd layer
"Objects"
2nd layer
"Object parts"
1st layer
"Edges"
Pixels
58

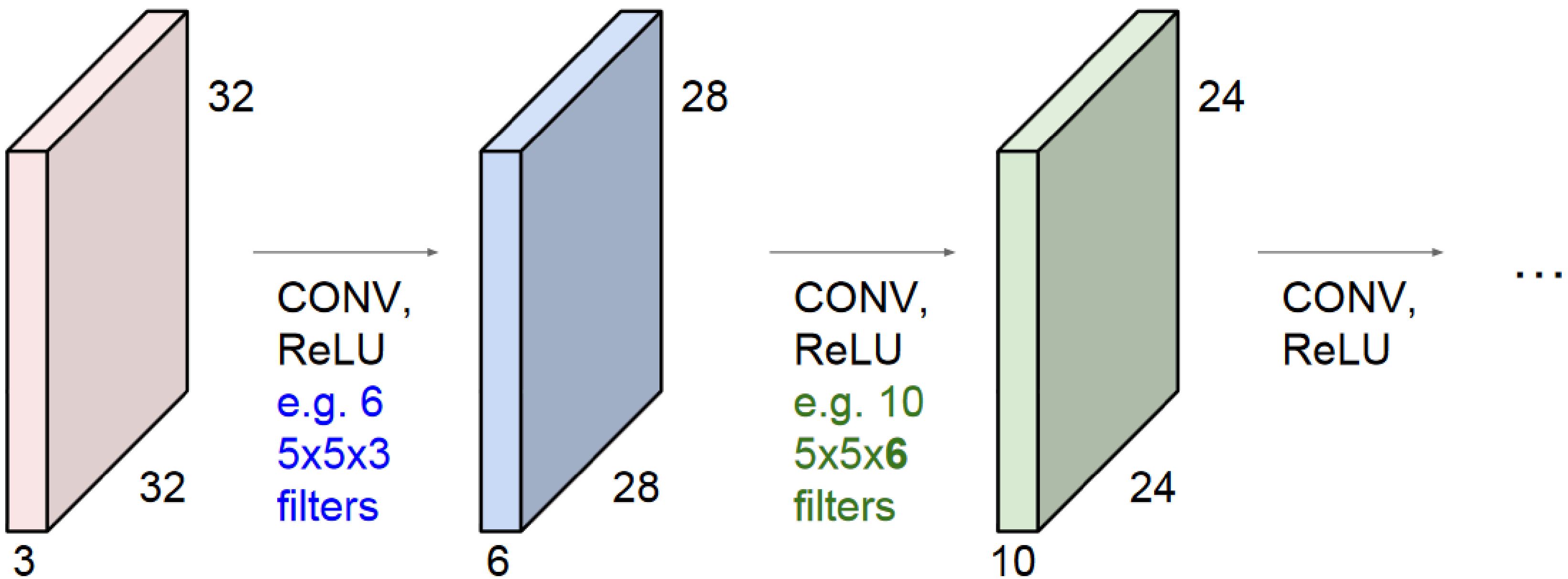

Layer 3
Layer 2
Layer 1



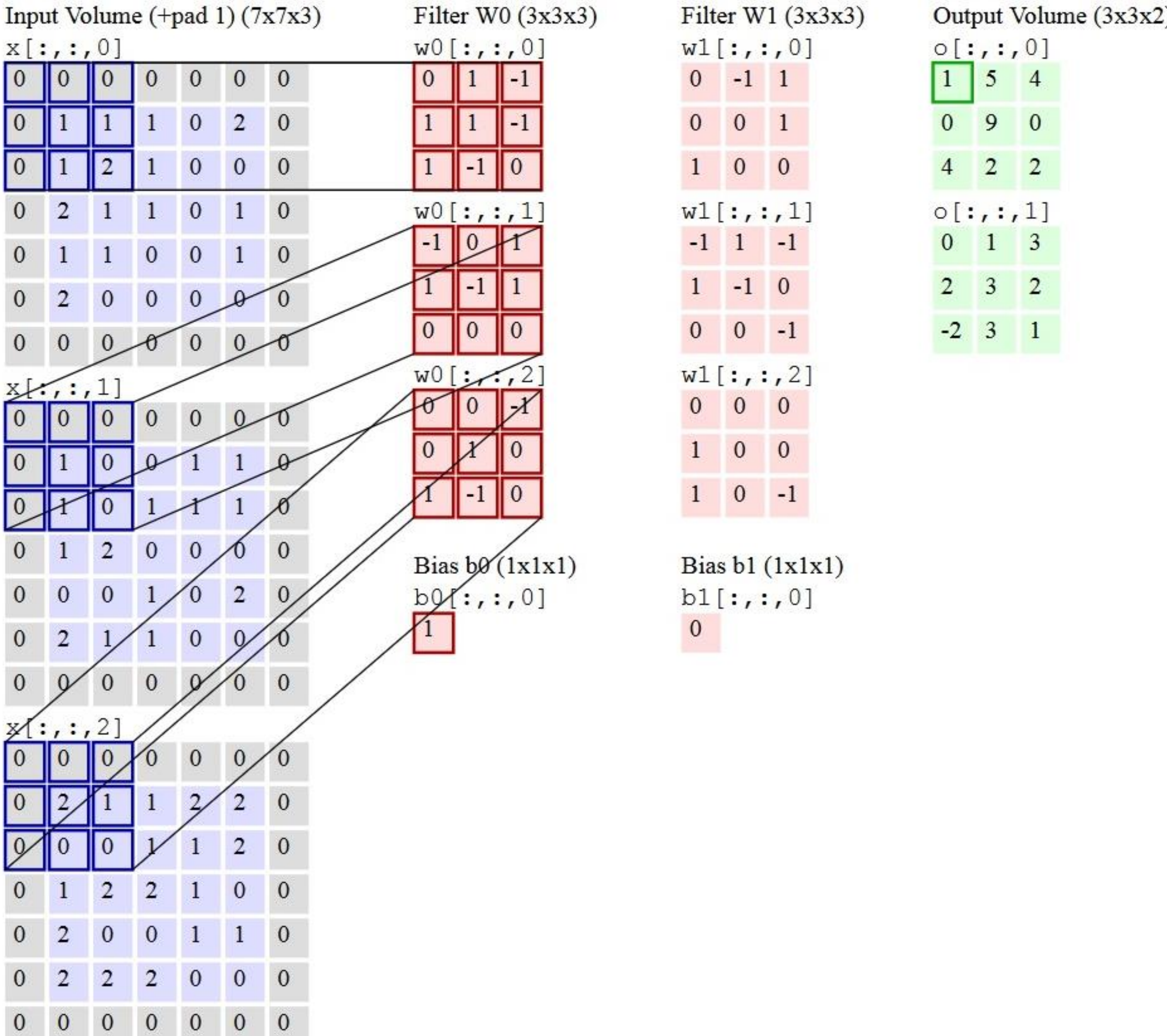


http://cs231n.stanford.edu/slides/2022/lecture_5_ruohan.pdf

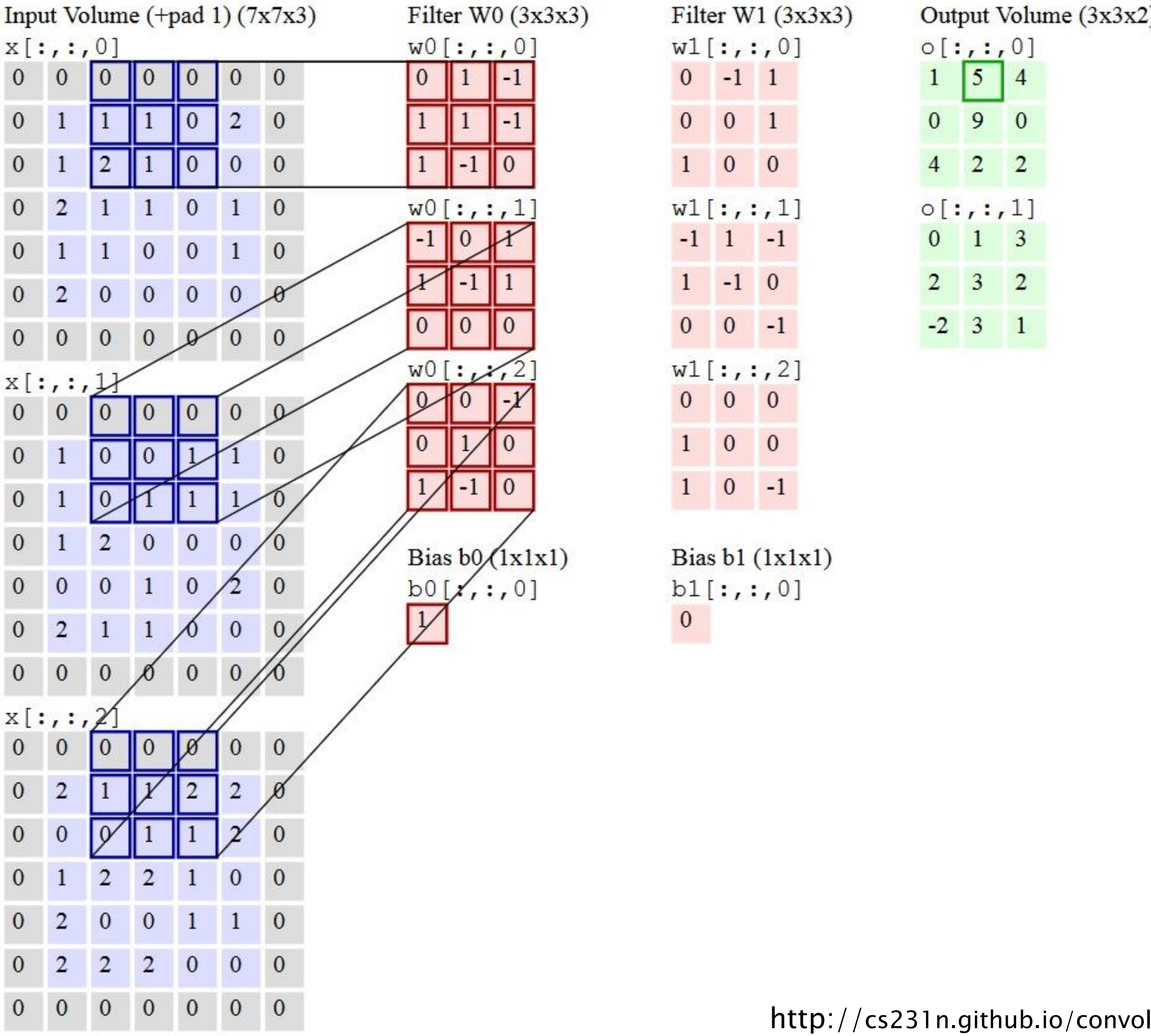
Preview: ConvNet is a sequence of Convolutional Layers, interspersed with activation functions



http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture5.pdf



http://cs231n.github.io/convolutional-networks/



http://cs231n.github.io/convolutional-networks/

Input Volume (+pad 1) (7x7x3)

x[:,:,0]

0	0	0	0	0	0	0
0	1	1	1	0	2	0
0	1	2	1	0	0	0
0	2	1	1	0	1	0
0	1	1	0	0	1	0
0	2	0	0	0	0	0
0	0	0	0	0	0	0

x[:,:,1]

0	0	0	0	0	0	0
0	1	0	0	1	1	0
0	1	0	1	1	1	0
0	1	2	0	0	0	0
0	0	0	1	0	2	0
0	2	1	1	0	0	0
0	0	0	0	0	0	0

x[:,:,2]

0	0	0	0	0	0	0
0	2	1	1	2	2	0
0	0	0	1	1	2	0
0	1	2	2	1	0	0
0	2	0	0	1	1	0
0	2	2	2	0	0	0
0	0	0	0	0	0	0

Filter W0 (3x3x3)

w0[:,:,0]

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

w0[:,:,1]

-1	0	1
1	-1	1
0	0	0

w0[:,:,2]

0	0	-1
0	1	0
1	-1	0

Bias b0 (1x1x1)

b0[:,:,0]

1

Filter W1 (3x3x3)

w1[:,:,0]

0	-1	1
0	0	1
1	0	0

w1[:,:,1]

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

w1[:,:,2]

0	0	0
1	0	0
1	0	-1

Bias b1 (1x1x1)

b1[:,:,0]

0

Output Volume (3x3x2)

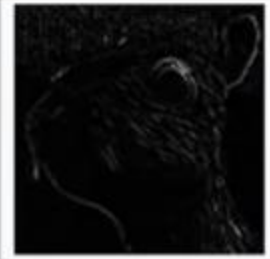
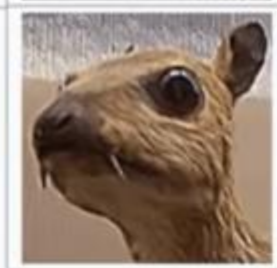
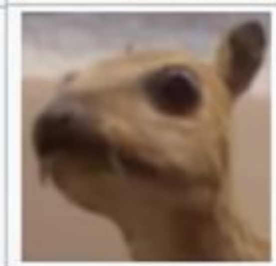
o[:,:,0]

1	5	4
0	9	0
4	2	2

o[:,:,1]

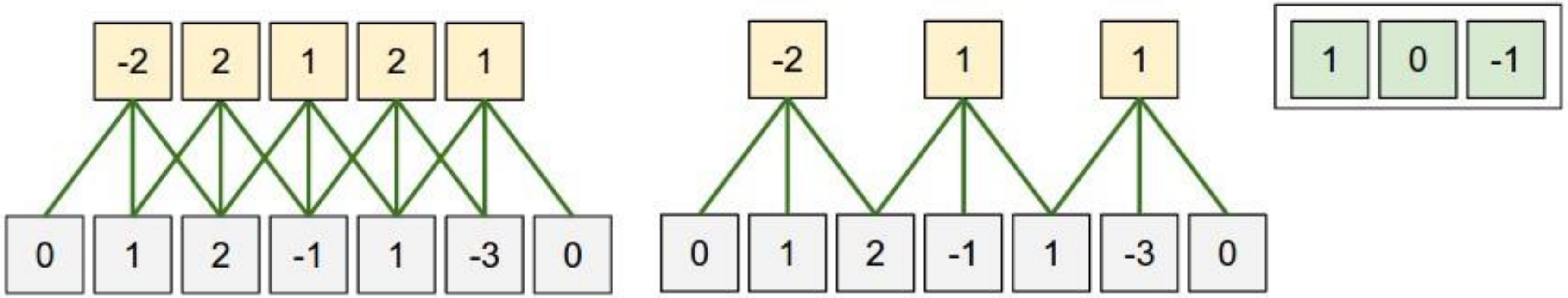
0	1	3
2	3	2
-2	3	1

http://cs231n.github.io/convolutional-networks/

Edge detection 	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	

https://en.wikipedia.org/wiki/Kernel_(image_processing)

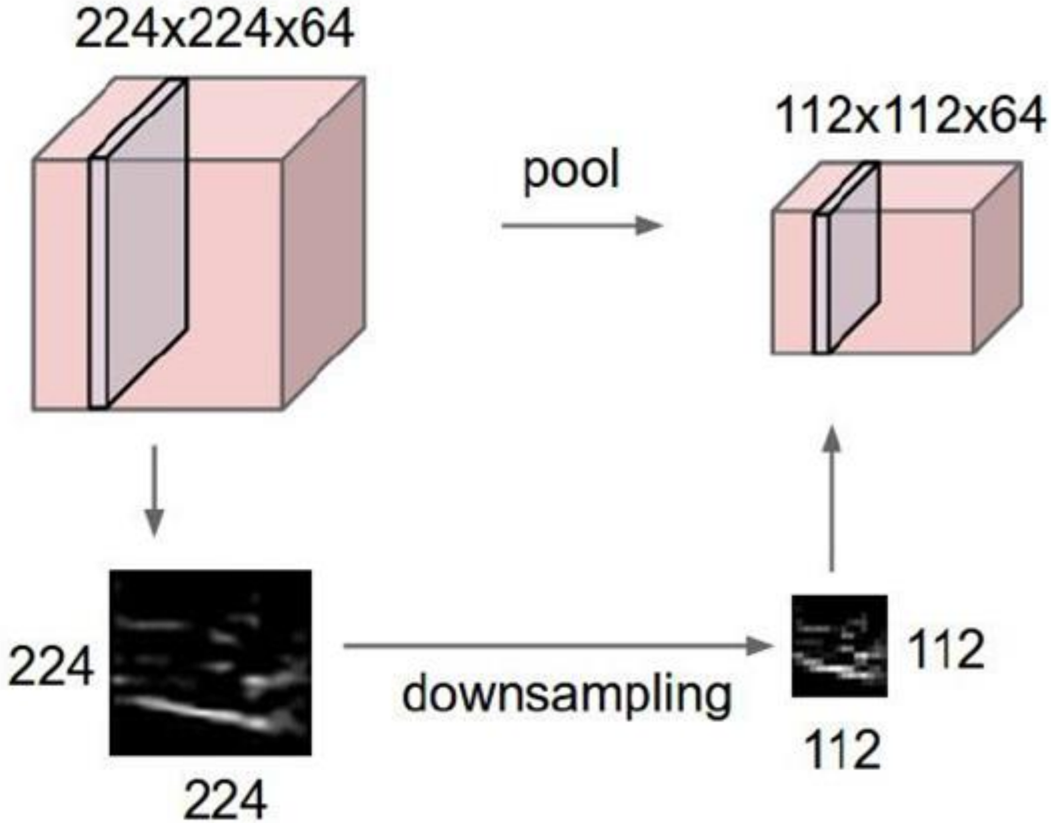
Conv1D



https://cs231n.github.io/convolutional-networks/

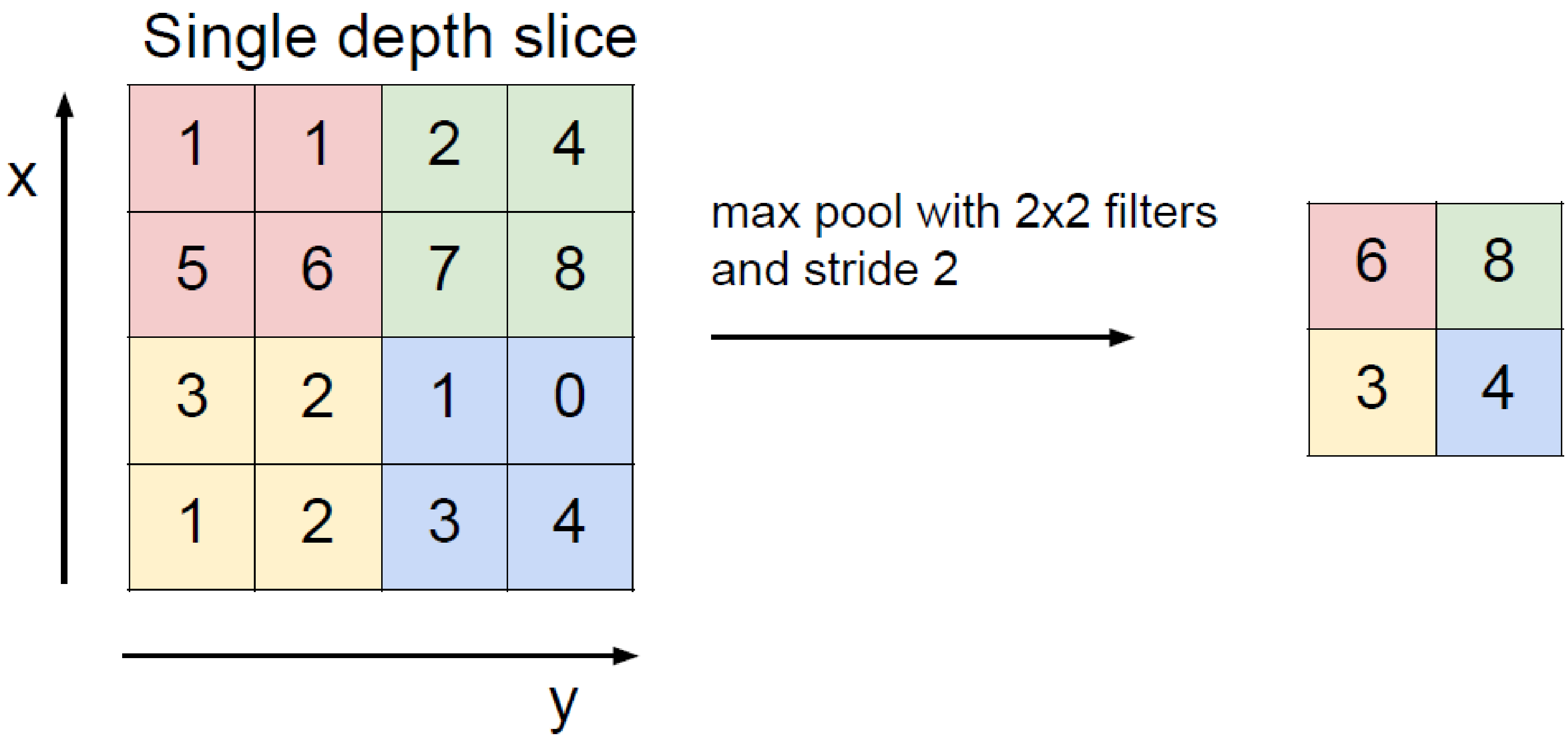
Pooling layer

- makes the representations smaller and more manageable
- operates over each activation map independently:



https://cs231n.github.io/convolutional-networks/

MAX POOLING



https://cs231n.github.io/convolutional-networks/

Convolutional Neural Network with TensorFlow

tf.keras.layers.Conv2D

```
tf.keras.layers.Conv2D(
    filters,
    kernel_size,
    strides=(1, 1),
    padding='valid',
    data_format=None,
    dilation_rate=(1, 1),
    groups=1,
    activation=None,
    use_bias=True,
    kernel_initializer='glorot_uniform',
    bias_initializer='zeros',
    kernel_regularizer=None,
    bias_regularizer=None,
    activity_regularizer=None,
    kernel_constraint=None,
    bias_constraint=None,
    **kwargs
)
```

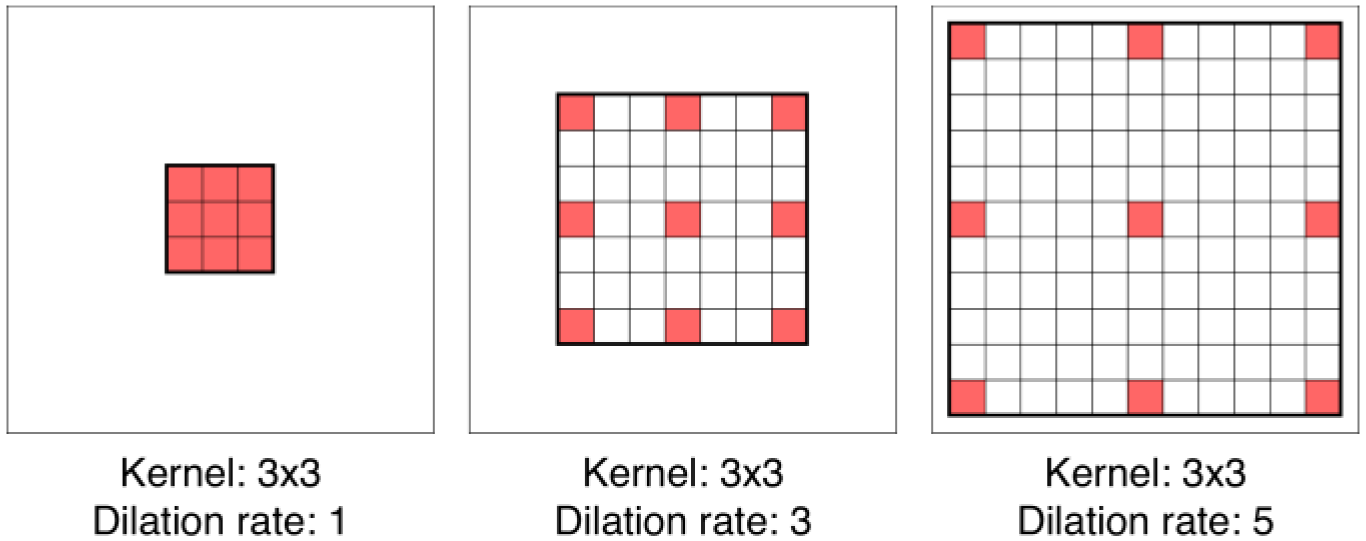
https://www.tensorflow.org/api_docs/python/tf/keras/layers/Conv2D

`filters`	Integer, the dimensionality of the output space (i.e. the number of output filters in the convolution).
`kernel_size`	An integer or tuple/list of 2 integers, specifying the height and width of the 2D convolution window. Can be a single integer to specify the same value for all spatial dimensions.
`strides`	An integer or tuple/list of 2 integers, specifying the strides of the convolution along the height and width. Can be a single integer to specify the same value for all spatial dimensions. Specifying any stride value != 1 is incompatible with specifying any `dilation_rate` value != 1.
`padding`	one of `"valid"` or `"same"` (case-insensitive). `"valid"` means no padding. `"same"` results in padding with zeros evenly to the left/right or up/down of the input. When `padding="same"` and `strides=1`, the output has the same size as the input.

`dilation_rate`	an integer or tuple/list of 2 integers, specifying the dilation rate to use for dilated convolution. Can be a single integer to specify the same value for all spatial dimensions. Currently, specifying any `dilation_rate` value != 1 is incompatible with specifying any stride value != 1.

https://www.tensorflow.org/api_docs/python/tf/keras/layers/Conv2D

dilated convolution



https://arxiv.org/pdf/1903.07992.pdf

tf.keras.layers.MaxPool2D

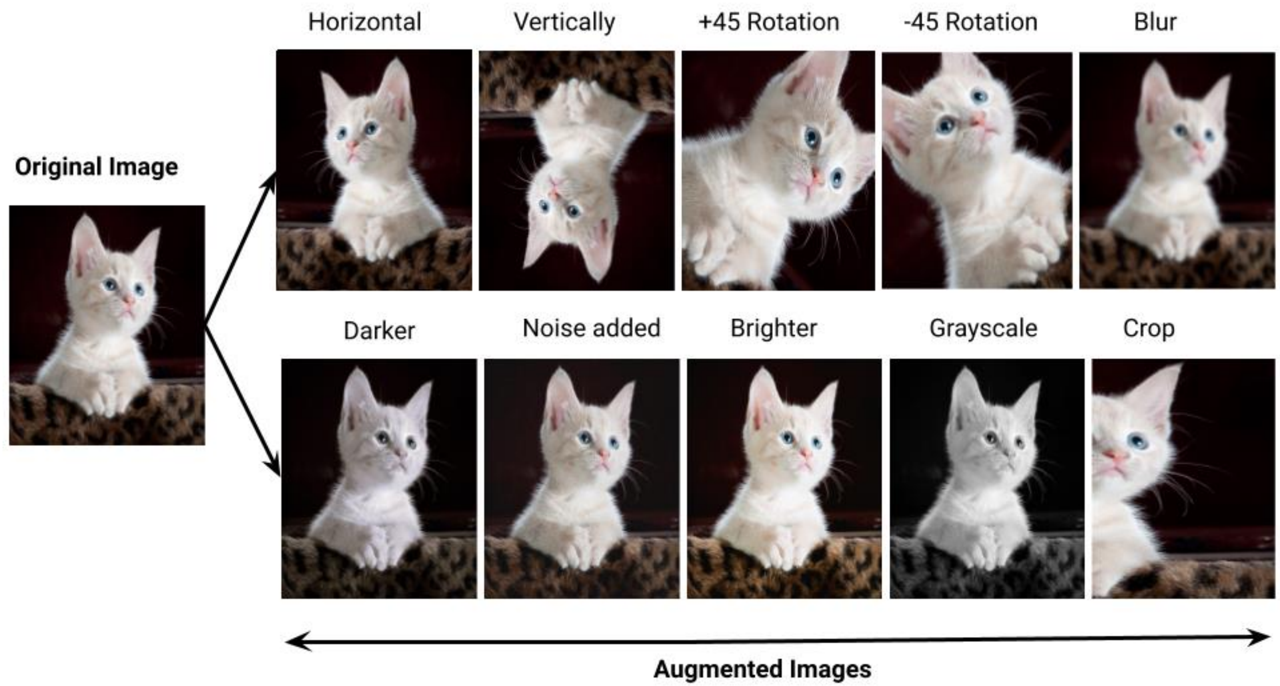
```
tf.keras.layers.MaxPool2D(
    pool_size=(2, 2), strides=None, padding='valid', data_format=None,
    **kwargs
)
```

tf.keras.layers.AveragePooling2D

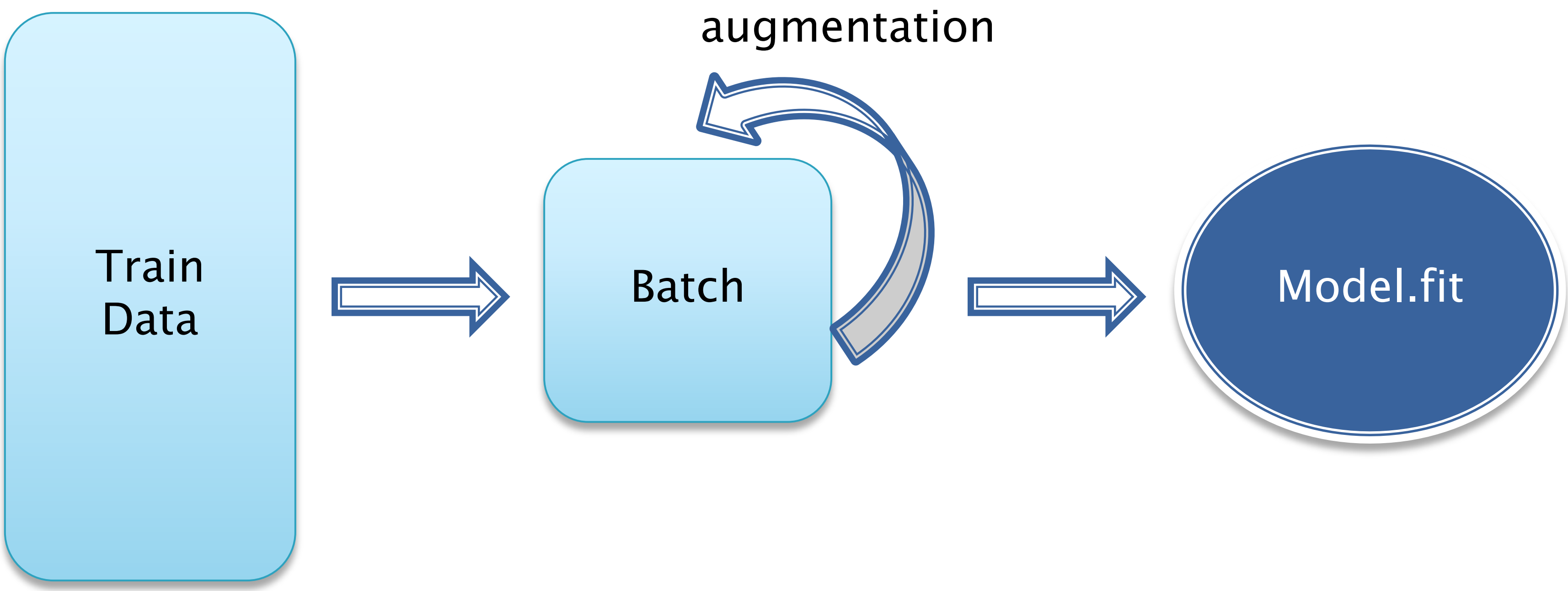
```
tf.keras.layers.AveragePooling2D(
    pool_size=(2, 2), strides=None, padding='valid', data_format=None,
    **kwargs
)
```

```
model = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(32, (3,3), activation='relu', input_shape=(150, 150, 3)),
    tf.keras.layers.MaxPooling2D(2, 2),
    tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
    tf.keras.layers.Conv2D(128, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Conv2D(128, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(512, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])
```

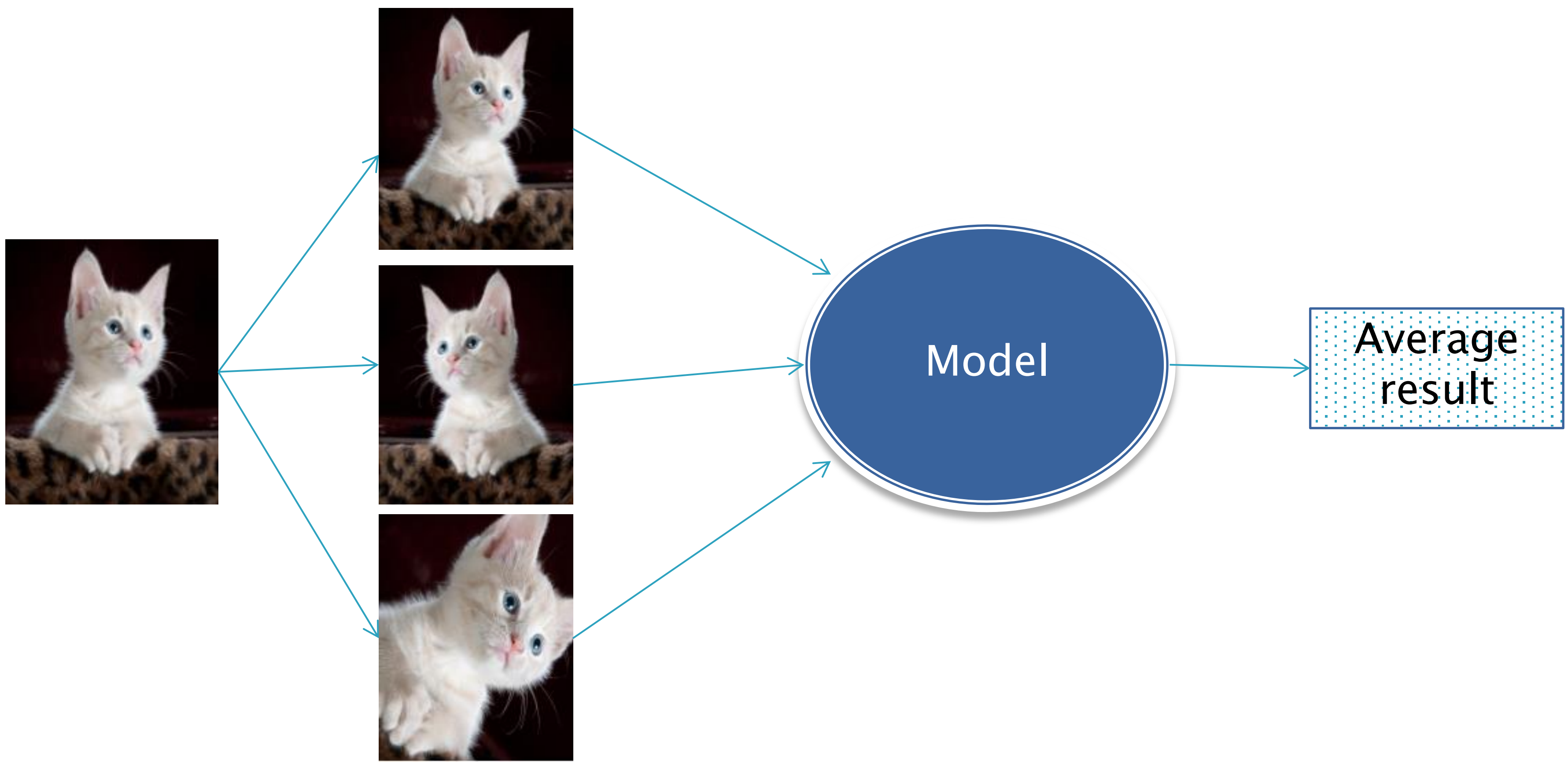
Data augmentation



Online augmentation



Test time augmentation



Data augmentation

Run in Google Colab | View source on GitHub | Download notebook

Overview

This tutorial demonstrates data augmentation: a technique to increase the diversity of your training set by applying random (but realistic) transformations, such as image rotation.

You will learn how to apply data augmentation in two ways:

- Use the Keras preprocessing layers, such as `tf.keras.layers.Resizing`, `tf.keras.layers.Rescaling`, `tf.keras.layers.RandomFlip`, and `tf.keras.layers.RandomRotation`.
- Use the `tf.image` methods, such as `tf.image.flip_left_right`, `tf.image.rgb_to_grayscale`, `tf.image.adjust_brightness`, `tf.image.central_crop`, and `tf.image.stateless_random*`.

https://www.tensorflow.org/tutorials/images/data_augmentation

tf.keras.preprocessing.image.ImageDataGenerator

View source on GitHub

Generate batches of tensor image data with real-time data augmentation.

View aliases

```
tf.keras.preprocessing.image.ImageDataGenerator(
    featurewise_center=False,
    samplewise_center=False,
    featurewise_std_normalization=False,
    samplewise_std_normalization=False,
    zca_whitening=False,
    zca_epsilon=1e-06,
    rotation_range=0,
    width_shift_range=0.0,
    height_shift_range=0.0,
    brightness_range=None,
    shear_range=0.0,
    zoom_range=0.0,
    channel_shift_range=0.0,
    fill_mode='nearest',
    cval=0.0,
    horizontal_flip=False,
    vertical_flip=False,
    rescale=None,
    preprocessing_function=None,
    data_format=None,
    validation_split=0.0,
    interpolation_order=1,
    dtype=None
)
```

https://www.tens[illegible]org/api_docs/python/tf/keras/preprocessing/image/ImageDataGenerator

Method flow

Takes data & label arrays, generates batches of augmented data.

Args	
`x`	Input data. Numpy array of rank 4 or a tuple. If tuple, the first element should contain the images and the second element another numpy array or a list of numpy arrays that gets passed to the output without any modifications. Can be used to feed the model miscellaneous data along with the images. In case of grayscale data, the channels axis of the image array should have value 1, in case of RGB data, it should have value 3, and in case of RGBA data, it should have value 4.
`y`	Labels.
`batch_size`	Int (default: 32).
`shuffle`	Boolean (default: True).
`sample_weight`	Sample weights.
`seed`	Int (default: None).
`save_to_dir`	None or str (default: None). This allows you to optionally specify a directory to which to save the augmented pictures being generated (useful for visualizing what you are doing).
`save_prefix`	Str (default: `' '`). Prefix to use for filenames of saved pictures (only relevant if `save_to_dir` is set).
`save_format`	one of "png", "jpeg", "bmp", "pdf", "ppm", "gif", "tif", "jpg" (only relevant if `save_to_dir` is set). Default: "png".
`ignore_class_split`	Boolean (default: False), ignore difference in number of classes in labels across train and validation split (useful for non-classification tasks)
`subset`	Subset of data (`"training"` or `"validation"`) if `validation_split` is set in `ImageDataGenerator`.

https://www.tens[illegible]org/api_docs/python/tf/keras/preprocessing/image/ImageDataGenerator

Example of using .flow(x, y):

```
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
y_train = utils.to_categorical(y_train, num_classes)
y_test = utils.to_categorical(y_test, num_classes)
datagen = ImageDataGenerator(
    featurewise_center=True,
    featurewise_std_normalization=True,
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    horizontal_flip=True,
    validation_split=0.2)
# compute quantities required for featurewise normalization
# (std, mean, and principal components if ZCA whitening is applied)
datagen.fit(x_train)
# fits the model on batches with real-time data augmentation:
model.fit(datagen.flow(x_train, y_train, batch_size=32,
         subset='training'),
         validation_data=datagen.flow(x_train, y_train,
         batch_size=8, subset='validation'),
         steps_per_epoch=len(x_train) / 32, epochs=epochs)
```

Method flow_from_directory

Takes the path to a directory & generates batches of augmented data.

```
flow_from_directory(
    directory,
    target_size=(256, 256),
    color_mode='rgb',
    classes=None,
    class_mode='categorical',
    batch_size=32,
    shuffle=True,
    seed=None,
    save_to_dir=None,
    save_prefix='',
    save_format='png',
    follow_links=False,
    subset=None,
    interpolation='nearest',
    keep_aspect_ratio=False
)
```

https://www.tens[illegible]org/api_docs/python/tf/keras/preprocessing/image/ImageDataGenerator

Args	
`directory`	string, path to the target directory. It should contain one subdirectory per class. Any PNG, JPG, BMP, PPM or TIF images inside each of the subdirectories directory tree will be included in the generator. See this script for more details.
`target_size`	Tuple of integers `(height, width)`, defaults to `(256,256)`. The dimensions to which all images found will be resized.
`color_mode`	One of "grayscale", "rgb", "rgba". Default: "rgb". Whether the images will be converted to have 1, 3, or 4 channels.
`classes`	Optional list of class subdirectories (e.g. `['dogs', 'cats']`). Default: None. If not provided, the list of classes will be automatically inferred from the subdirectory names/structure under `directory`, where each subdirectory will be treated as a different class (and the order of the classes, which will map to the label indices, will be alphanumeric). The dictionary containing the mapping from class names to class indices can be obtained via the attribute `class_indices`.
`class_mode`	One of "categorical", "binary", "sparse", "input", or None. Default: "categorical". Determines the type of label arrays that are returned: • "categorical" will be 2D one-hot encoded labels, • "binary" will be 1D binary labels, "sparse" will be 1D integer labels, • "input" will be images identical to input images (mainly used to work with autoencoders). • If None, no labels are returned (the generator will only yield batches of image data, which is useful to use with `model.predict_generator()`). Please note that in case of class_mode None, the data still needs to reside in a subdirectory of `directory` for it to work correctly.
`batch_size`	Size of the batches of data (default: 32).

https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/image/ImageDataGenerator

Example of using .flow_from_directory(directory):

```
train_datagen = ImageDataGenerator(
        rescale=1./255,
        shear_range=0.2,
        zoom_range=0.2,
        horizontal_flip=True)
test_datagen = ImageDataGenerator(rescale=1./255)
train_generator = train_datagen.flow_from_directory(
        'data/train',
        target_size=(150, 150),
        batch_size=32,
        class_mode='binary')
validation_generator = test_datagen.flow_from_directory(
        'data/validation',
        target_size=(150, 150),
        batch_size=32,
        class_mode='binary')
model.fit(
        train_generator,
        steps_per_epoch=2000,
        epochs=50,
        validation_data=validation_generator,
        validation_steps=800)
```

https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/image/ImageDataGenerator

`flow_from_dataframe`

View source

```
flow_from_dataframe(
    dataframe,
    directory=None,
    x_col='filename',
    y_col='class',
    weight_col=None,
    target_size=(256, 256),
    color_mode='rgb',
    classes=None,
    class_mode='categorical',
    batch_size=32,
    shuffle=True,
    seed=None,
    save_to_dir=None,
    save_prefix='',
    save_format='png',
    subset=None,
    interpolation='nearest',
    validate_filenames=True,
    **kwargs
)
```

Takes the dataframe and the path to a directory + generates batches.

The generated batches contain augmented/normalized data.

**A simple tutorial can be found **here.

https://www.tens[illegible]org/api_docs/python/tf/keras/preprocessing/image/ImageDataGenerator