Deep Learning for Computer Vision

Francesco Pugliese, PhD

neural1977@gmail.com

Computer Vision: Where does Traditional Statistics fail?

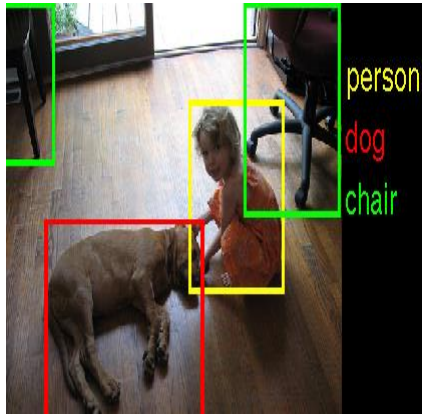
- **Computer Vision** is an interdisciplinary field that deals with the way algorithms can be made for gaining high-level understanding from digital images or videos.

- **Statistical methods** are not always welcome in computer vision.

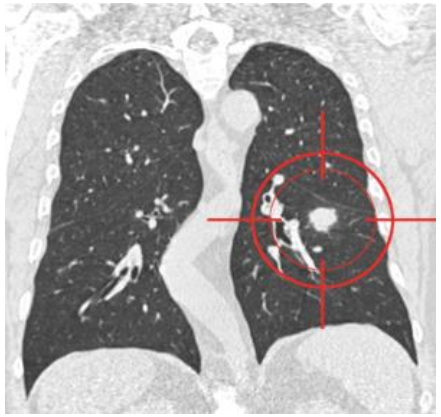
- Statistical methods seem **not scaling up** to the challenges of computer vision problems (Chellappa, R., 2012).

Why does Computer Vision matter so much?

EVERYDAY LIFE

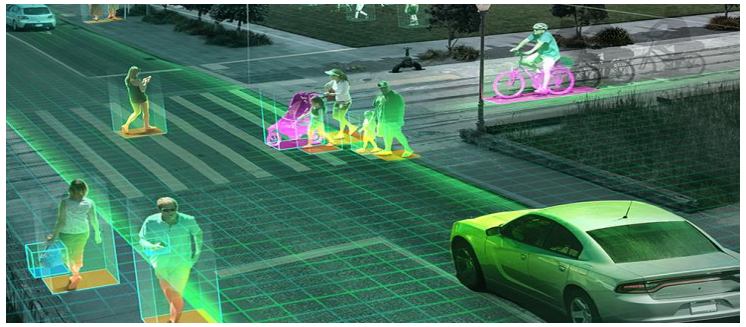


BIOMEDICAL IMAGES



- A new study proves the **relationship** between **Vision capabilities and Intelligence** (Tsukahara et al., 2016).
- Seemingly, **human beings** have one of **the most complex vision systems** within the **animal realm** and this fact would have fostered **the evolution of human intelligence.**
- In other words, **Computer Vision** needs **human-like intellectual capabilities** in order to achieve the vision performances required by humans for every-day applications.

Why does Computer Vision matter so much?



- **A new generation of machines** might accomplish typical human tasks such as recognizing and moving objects, driving cars, cultivating fields, cleaning streets, city garbage collecting, etc.

Computer Vision for Recycling

- **Smart Garbage Bins** could classify waste and differentiate it automatically. This might reduce the enormous impact of human errors.

- As far as food **sustainability** is concerned, a first version of a classifier might enable a **restaurant** or a shop to recognize whether there is still a return of residual food from the production process. Afterwards, by exploiting other sub-classifiers we could catch the opportunity of regenerating this residual food saving resources.

- **AI applications** which can optimize all the **flow of interactions** with a **restaurant**, decreasing the process costs and increasing sustainability *(«Deep Learning al servizio del riciclo», Intervista su Wired, Pugliese, F.)*. Thanks to **Artificial Intelligence**, in one work, some researchers have generated menus of some restaurants analyzing all **non-structured data** from customers online comments.

Image Classification Problem

What we see

What computers see

```
08 02 22 97 38 15 00 40 00 75 04 05 07 78 52 12 50 77 91 08
49 49 99 40 17 81 18 57 60 87 17 40 98 43 69 48 04 56 62 00
81 49 31 73 55 79 14 29 93 71 40 67 53 88 30 03 49 13 36 65
52 70 95 23 04 60 11 42 69 24 68 56 01 32 56 71 37 02 36 91
22 31 16 71 51 67 63 89 41 92 36 54 22 40 40 28 66 33 13 80
24 47 32 60 99 03 45 02 44 75 33 53 78 36 84 20 35 17 12 50
32 98 81 28 64 23 67 10 26 38 40 67 59 54 70 66 18 38 64 70
67 26 20 68 02 62 12 20 95 63 94 39 63 08 40 91 66 49 94 21
24 55 58 05 66 73 99 26 97 17 78 78 96 83 14 88 34 89 63 72
21 36 23 09 75 00 76 44 20 45 35 14 00 61 33 97 34 31 33 95
78 17 53 28 22 75 31 67 15 94 03 80 04 62 16 14 09 53 56 92
16 39 05 42 96 35 31 47 55 58 88 24 00 17 54 24 36 29 85 57
86 56 00 48 35 71 89 07 05 44 44 37 44 60 21 58 51 54 17 58
19 80 81 68 05 94 47 69 28 73 92 13 86 52 17 77 04 89 55 40
04 52 08 83 97 35 99 16 07 97 57 32 16 26 26 79 33 27 98 66
88 36 68 87 57 62 20 72 03 46 33 67 46 55 12 32 63 93 53 69
04 42 16 73 38 25 39 11 24 94 72 18 08 46 29 32 40 62 76 36
20 69 36 41 72 30 23 88 34 62 99 69 82 67 59 85 74 04 36 16
20 73 35 29 78 31 90 01 74 31 49 71 48 86 81 16 23 57 05 54
01 70 54 71 83 51 54 69 16 92 33 48 61 43 52 01 89 19 67 48
```

Bag of Words and Term Document Matrix

Document

In the beginning God created the heaven and the earth.
And the earth was without form, and void; and darkness was upon the face of the deep.
And the Spirit of God moved upon the face of the waters.
And God said, Let there be light: and there was light.

Representation

beginning	1
earth	2
God	3

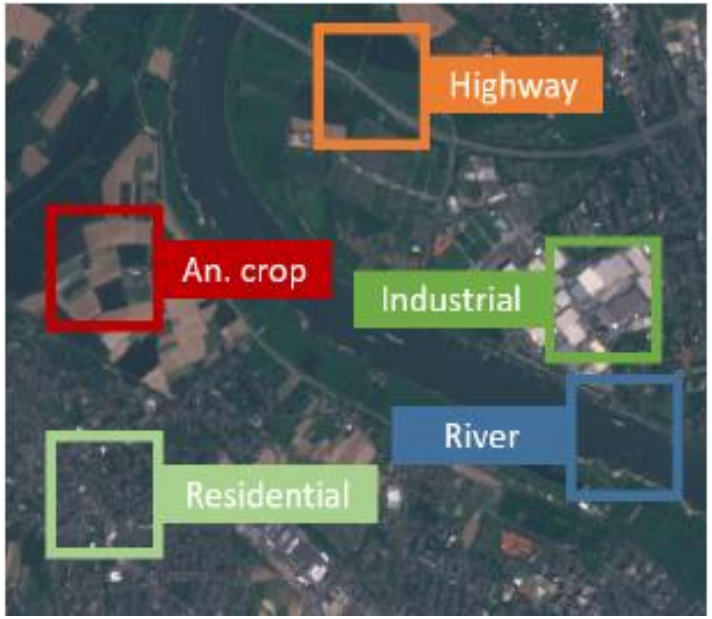
Type	relax	carrello	prenotare	shop	wishlist
0	0	0	0	0	0
0	0	0	0	0	0
1	0	0	0	0	0
1	0	0	1	0	0
0	0	0	0	0	0
0	0	1	0	1	1
1	1	1	1	0	0

From each web-site: Bag-Of-Words

From all web-sites' bag-of-words: TDM

From each TDM row: 32x32 squared image encoding

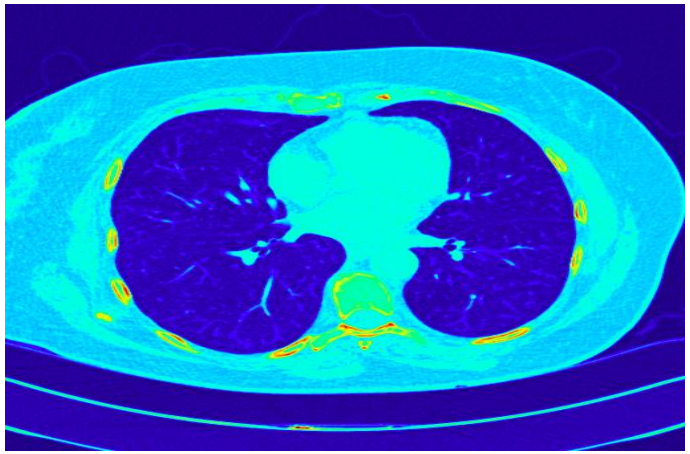
Automatic Extraction of Statistics from Satellite Imagery: Land Use and Land Cover Classification

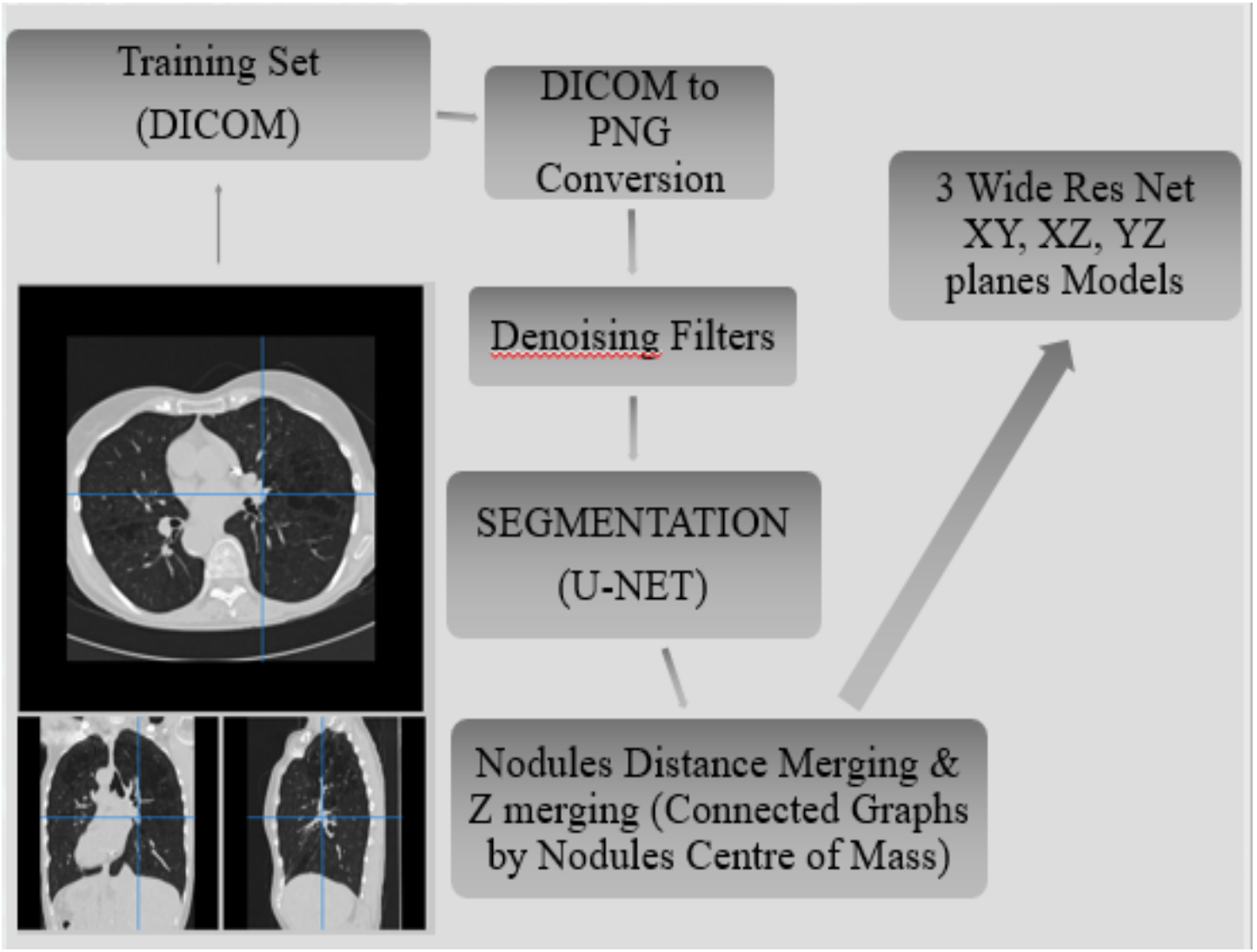


- Nowadays, more and more public and up-to-dated **satellite imagery** data for Earth obser-vation are **available**.

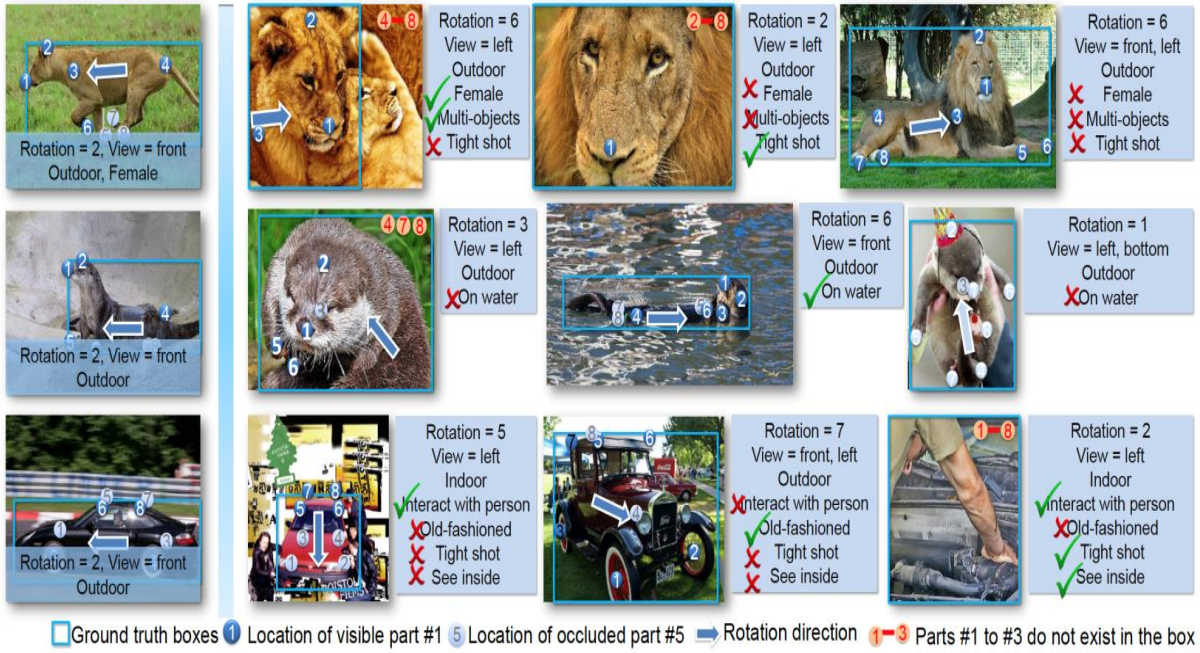
- However, to fully utilize this data, in order to automatically extract statistics, **satellite images** must be processed and transformed into **structured semantics**.

Lung Cancer Detection





Computer Vision Datasets and Competitions



- **ImageNet**: ImageNet is a dataset of over **15** million labeled high-resolution images belonging to roughly **22,000** categories.

- Since **2010** a competition called «ImageNet Large-Scale Visual Recognition Challenge (ILSVRC)» uses a subset of ImageNet with roughly **1000** images in each of **1000** categories.

Train Set: 1.2 million

Validation Set: 50,000

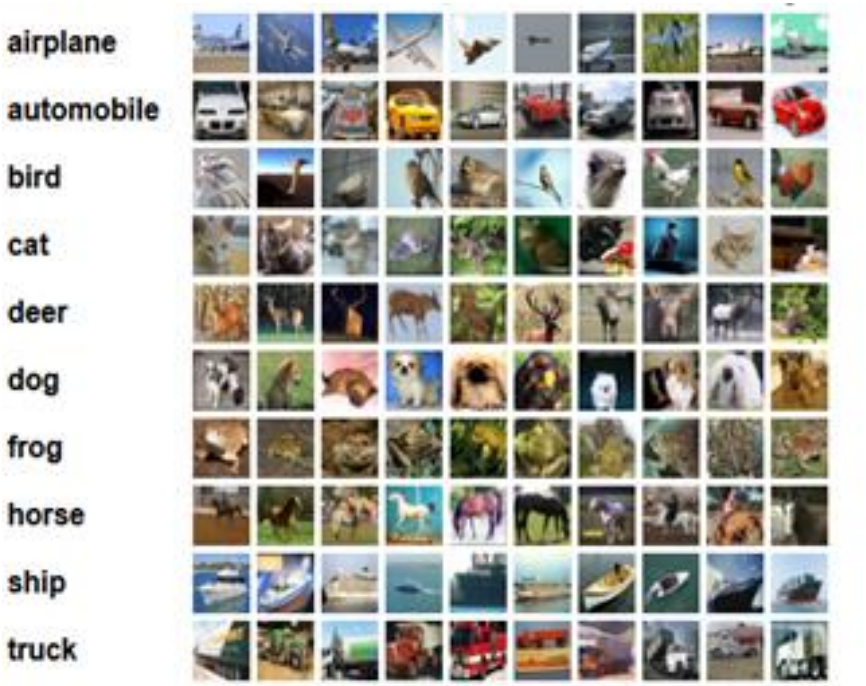
Test set: 150,000

Computer Vision Datasets and Competitions

- **Kaggle**: In 2010, Kaggle was founded as a platform for predictive **modeling** and **analytics competitions** on which companies and researchers post their data.
- Statisticians and data scientists from all over the world compete to produce the best models.
- **Data Science Bowl 2017** was the biggest competition focused on "Lung Cancer Detection". The competition was founded by **Arnold Foundation** and awarded **$1 million in prizes** (**1st** ranked **$500,000**).

- **Train Set:** around 150 CT labelled scans images per patient from 1200 patients encoded in **DICOM** format.
- **Stage 1 test set:** 190 patients CT scans.
- **Stage 2 test :** 500 patients CT scans.

Computer Vision Benchmark Datasets

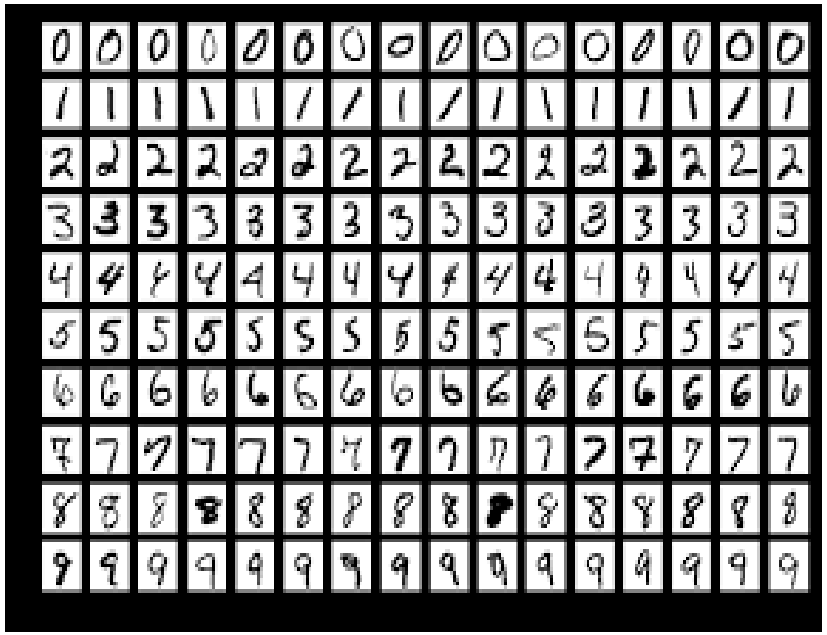


- The **CIFAR-10** dataset consists of **60,000 32x32** coloured images divided into **10 classes**, with **6000 images per class**. There are **50,000 training** images and **10,000 test** images.

- The dataset is divided into **five training batches** and one test batch, each with **10,000** images. The test batch contains exactly 1000 randomly-selected images from each class. The training batches contain the remaining images in random order, but some training batches may contain more images from one class than another. Between them, the training batches contain exactly 5000 images from each class.

- The classes are completely **mutually exclusive**. There is **no overlap** between **automobiles** and **trucks**. "Automobile" includes sedans, SUVs, things of that sort. "Truck" includes only big trucks. Neither includes pickup trucks.

Computer Vision Benchmark Datasets



- The **MNIST** database of handwritten digits has a training set of **60,000** examples, and a test set of **10,000** examples.
- It is a subset of a larger set available from **NIST**. The digits have been size-normalized and centered in a fixed-size image.
- It is a good database for people who want to try learning techniques and pattern recognition methods on real-world data while spending minimal efforts on **preprocessing** and **formatting**.

- The images contain **grey levels** as a result of the **anti-aliasing technique** used by the normalization algorithm. The images were centered in a **28x28** image by computing the center of mass of the pixels, and translating the image so as to position this point at the center of the **28x28** field.

Computer Vision Benchmark Datasets

- **Fashion MNIST** is a dataset of **Zalando's** article images consisting of a training set of **60,000** images and a test set of **10,000** examples.

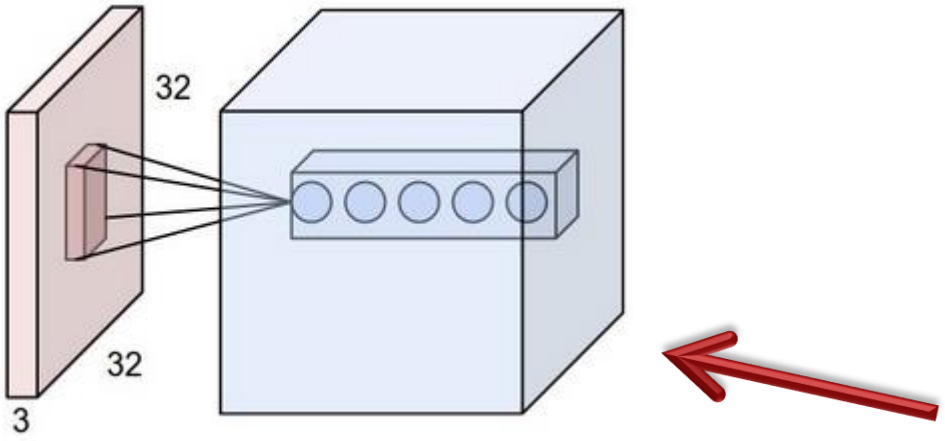
- Each example is a **28 x 28** grayscale image associated with a label from **10 classes**.

- Reasons for replacing MNIST according to **Zalando** opinions:

  - **MNIST is too easy.** Convolutional nets can achieve 99.7% on **MNIST**. Classic machine learning algorithms can also achieve 97% easily.

  - **MNIST is overused and cannot represent modern CV tasks**, as noted by deep learning **expert/Keras** author **François Chollet**.

Convolutional Neural Networks (ConvNets or CNNs)

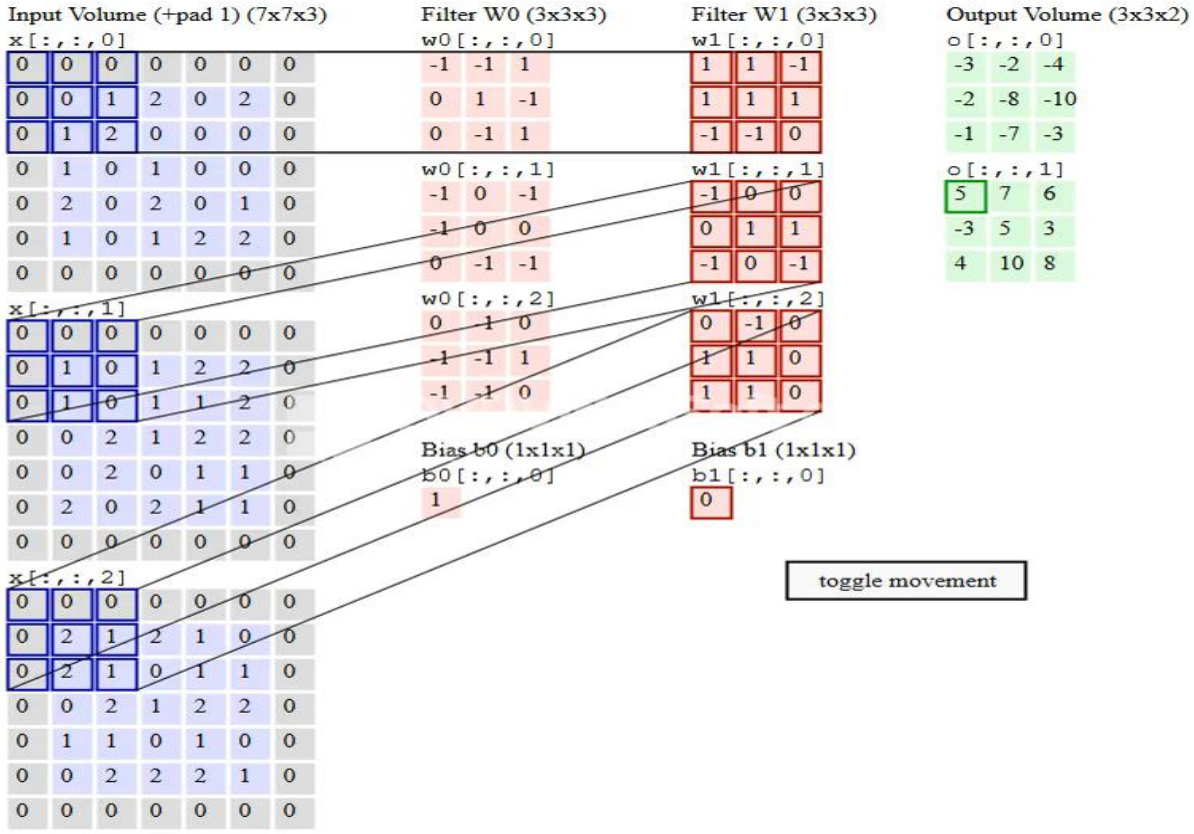


- **Convolutional Neural Networks (CNN)** are biologically-inspired variants of **MLPs**. We know the visual cortex contains a complex arrangement of cells **(*Hubel, D. and Wiesel, T., 1968*)**. These cells are sensitive to small sub-regions of the visual field, called a *receptive field*. Other layers are: **RELU layer**, **Pool Layer**. Typical CNNs settings are: a) **Number of Kernels (Filters)**, b) **Receptive Field size**, b) **Padding**, c) **Stride.** These parameters are tied by the following formula:

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

- **Each neuron in the convolutional layer is connected only to a local region in the input volume spatially. In this case there are 5 neurons along the depth all looking at the same region.**

Typical Settings of Convolutional Layers



a) **Number of Kernels (Filters)**

b) **Receptive Field size**

c) **0-Padding**

d) **Stride**

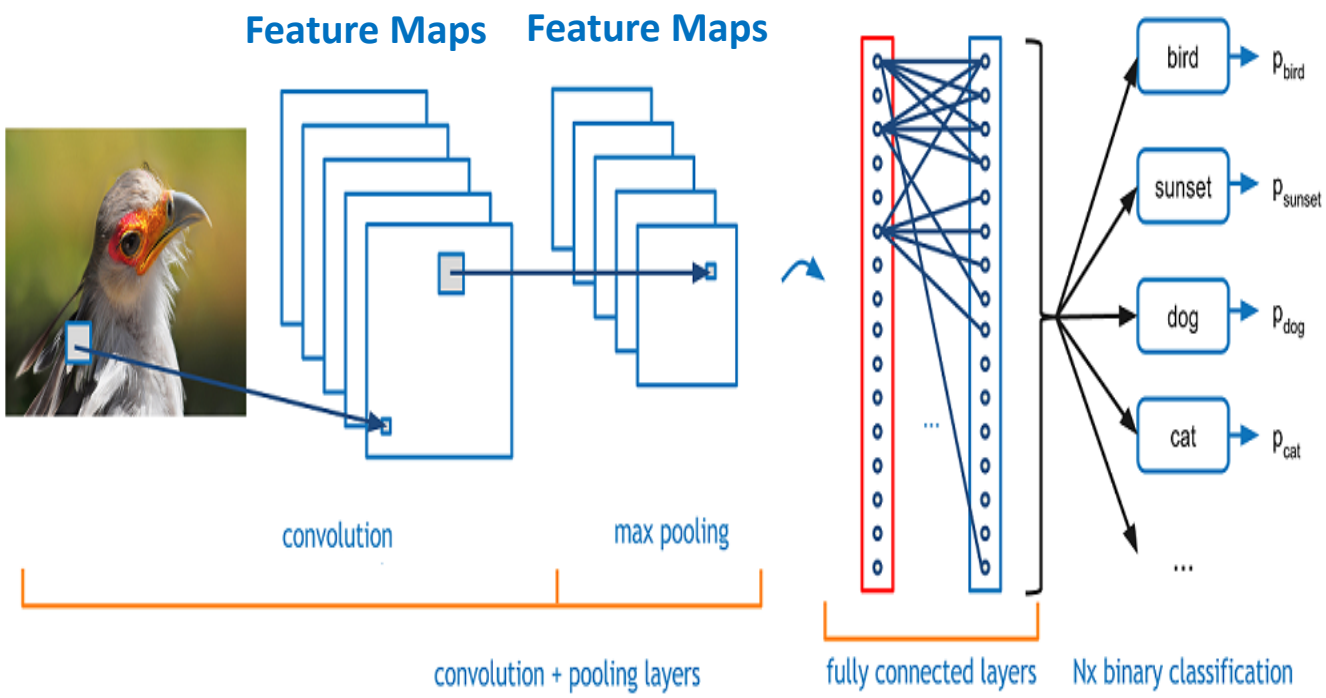
These parameters are tied by the following equation:

(W – F + 2P) / S + 1

Other layers:

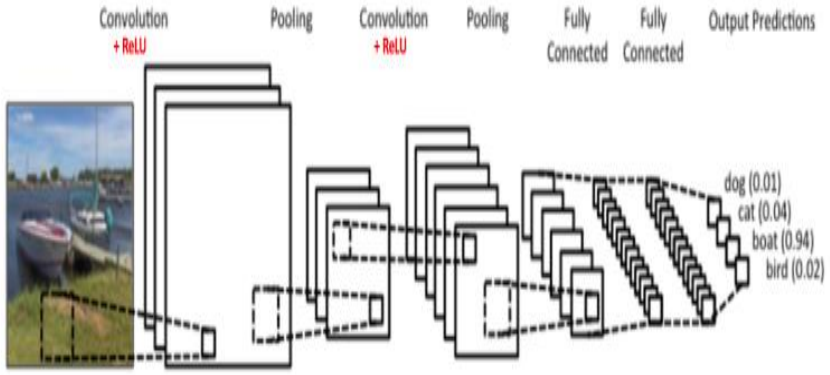
- **Pool Layer**
- **Activation Layer (RELU, TanH, Sigmoid)**
- **Fully Connected Layer**

Convolutional Neural Networks (ConvNets or CNNs)

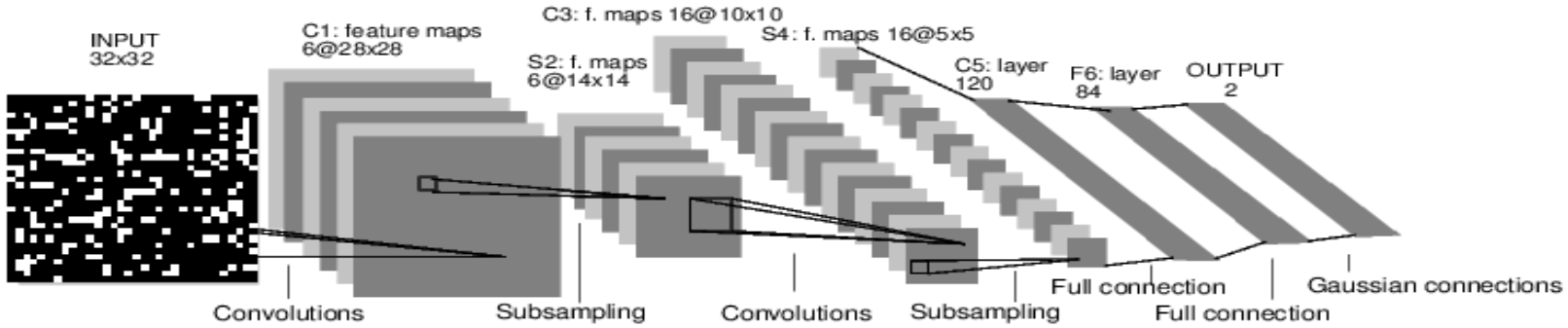


- **CNNs** were initially devised for **Image Recognition,** nowadays very often reach better-than-human accuracy
- **CNNs** need to be fed with **images,** but since for a machine images are just **numeric matrices…**
- **…**they are increasingly being used in **Natural Language Processing,** e.g. **text classification,** with excellent results

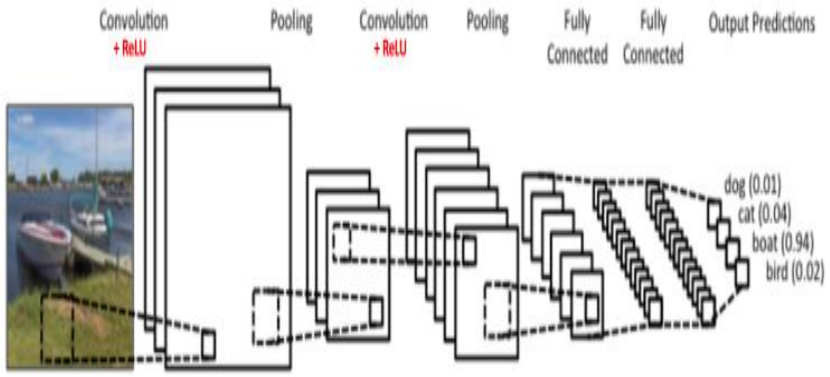
LeNet – Convolutional Neural Network



- **LeNet** was one of the very first convolutional neural networks which helped to propel the field of Deep Learning. This pioneering work by Yann LeCun was named LeNet5 after many previous successful iterations since the year 1989.

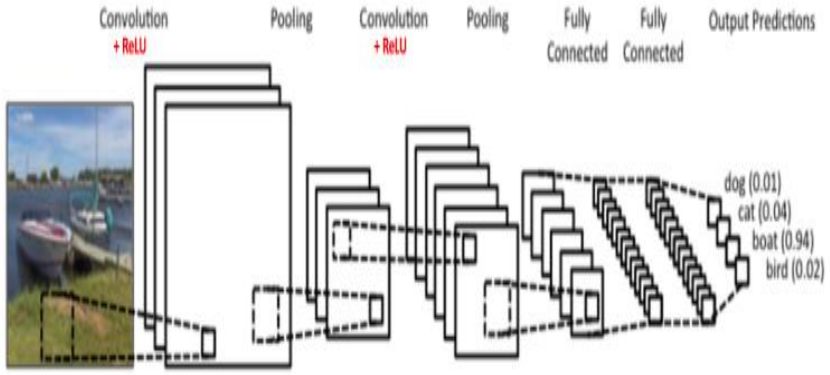


LeNet in Keras



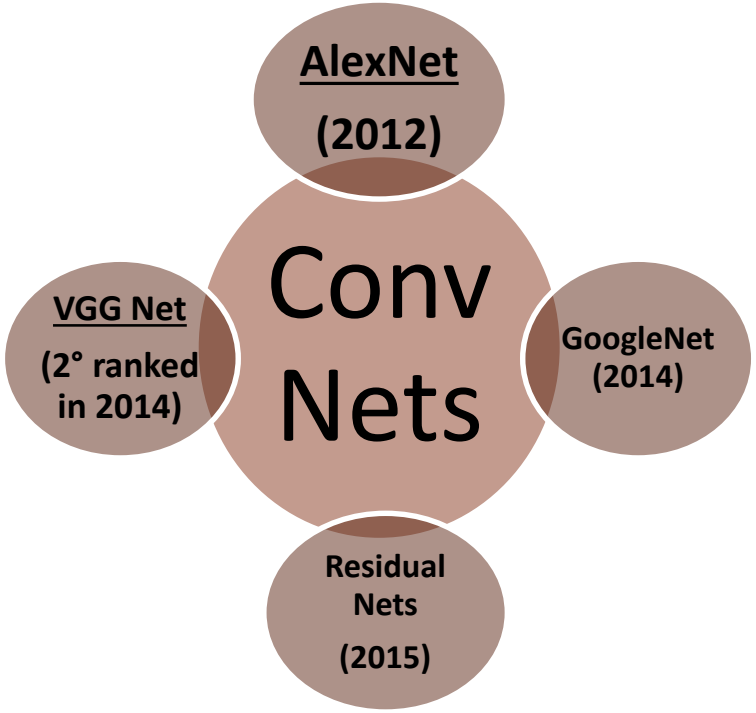

```
'''
Created on 22/03/2017

@author: Francesco Pugliese
'''

from keras.models import Sequential
from keras.layers.convolutional import Convolution2D
from keras.layers.convolutional import MaxPooling2D
from keras.layers.core import Activation
from keras.layers.core import Flatten
from keras.layers.core import Dense

class LeNet:
    @staticmethod
    def build(width, height, depth, classes, summary, weightsPath=None):
        # initialize the model
        model = Sequential()
        # first set of CONV => RELU => POOL
        model.add(Convolution2D(20, 5, 5, border_mode="same", input_shape=(depth, height, width)))
        model.add(Activation("relu"))
        model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))
```

LeNet in Keras




```
# second set of CONV => RELU => POOL
model.add(Convolution2D(50, 5, 5, border_mode="same"))
model.add(Activation("relu"))
model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))

#set of FC => RELU Layers
model.add(Flatten())
model.add(Dense(500))
model.add(Activation("relu"))


#softmax classifier
model.add(Dense(classes))
model.add(Activation("softmax"))

if summary==True:
    model.summary()

#if a weights path is supplied (indicating that the model was pre-trained), then load the weights
if weightsPath is not None:
    model.load_wights(weightsPath)

return model
```

Former Successful Deep Neural Networks



Traditional issues with Convolutional Layers:

•Wide convolutional layers determine overfitting and vanishing gradient problem with the Solver (SGD, Adam, etc).

•Low depth architectures produce raw features (need to push the depth forward).

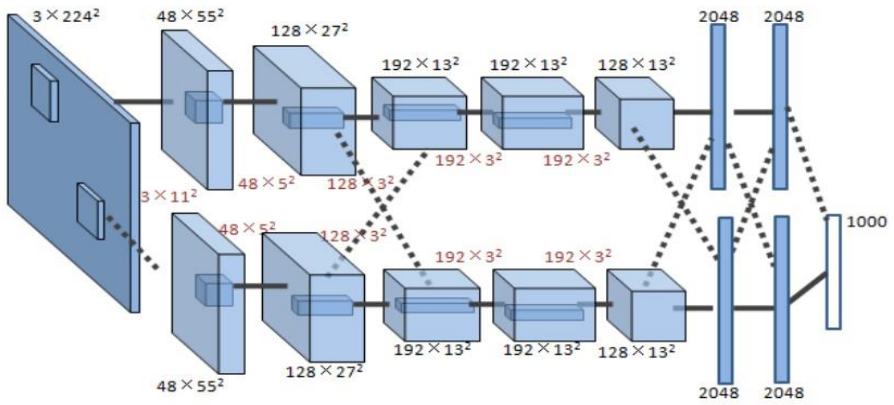
Model Regularization :

•Dropout - Co-adaptation and Models Ensemble (Srivastava, N. et al., 2014).

•Weight penalty L1/L2

•Data Augmentation (crop, flip, rotation, ZCA whitening, etc.)

AlexNet - University of Toronto (Krizhevsky, A. et al, 2012)



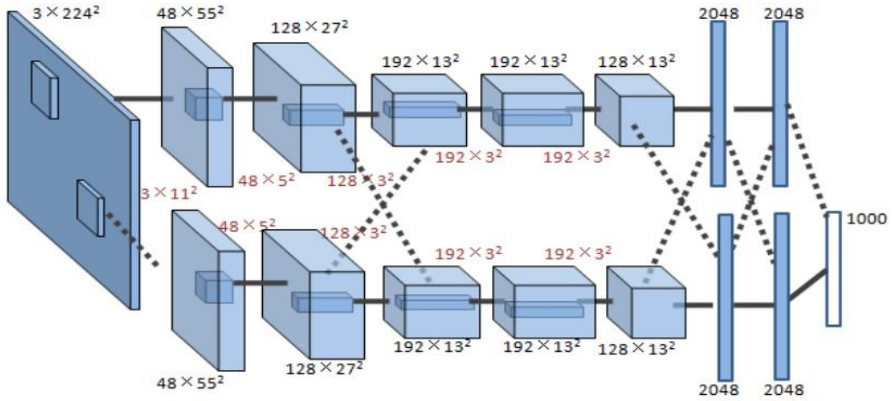
Critical Feautures:

- **8 trainable layers:** 5 convolutional layers and 3 fully connected layers.
- **Max pooling layers** after 1st, 2nd and 5th layer.
- **Rectified Linear Units (ReLUs) (**Nair, V., & Hinton, G. E. 2010).
- **Local Response Normalization.**
- **60 millions parameters, 650 thousands neurons**.
- **Regularizations**: Dropout (prob 0.5 in the first 2 fc layers, Data Augmentation (translactions, horizontal reflections, PCA on RGB).
- **Trained** on 2 **GTX 580** 3 GB GPUs.

Results:

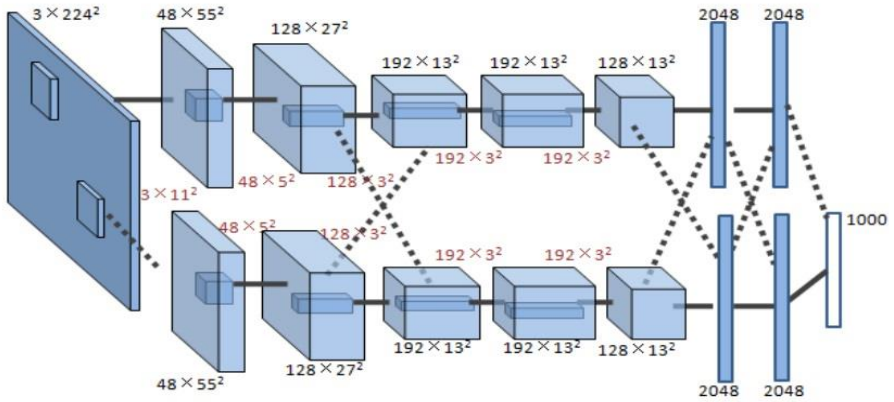
- **1 CNNs: 40.7%** Top-1 Error, **18.2%** Top-5 Error
- **5 CNNs: 38.1%** Top-1 Error,**16.4%** Top-5 Error
  - **SIFT+FVs: 26.2%** Top-5 Error (Sánchez, J., et al., 2013).

AlexNet - University of Toronto (Krizhevsky, A. et al, 2012)



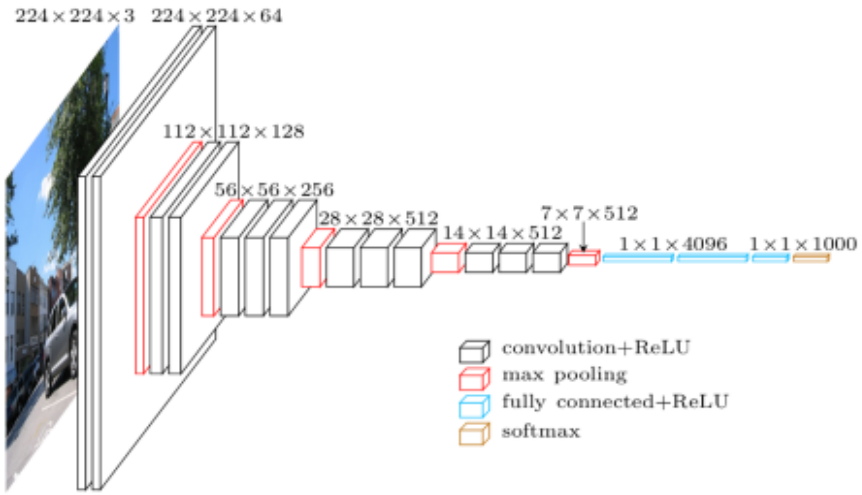

```
def get_alexnet(input_shape,nb_classes,mean_flag):
        # code adapted from https://github.com/heuritech/convnets-keras

        inputs = Input(shape=input_shape)

        if mean_flag:
                mean_subtraction = Lambda(mean_subtract, name='mean_subtraction')(inputs)
                conv_1 = Convolution2D(96, 11, 11,subsample=(4,4),activation='relu',
                                       name='conv_1', init='he_normal')(mean_subtraction)
        else:
                conv_1 = Convolution2D(96, 11, 11,subsample=(4,4),activation='relu',
                                       name='conv_1', init='he_normal')(inputs)

        conv_2 = MaxPooling2D((3, 3), strides=(2,2))(conv_1)
        conv_2 = crosschannelnormalization(name="convpool_1")(conv_2)
        conv_2 = ZeroPadding2D((2,2))(conv_2)
        conv_2 = merge([
            Convolution2D(128,5,5,activation="relu",init='he_normal', name='conv_2_'+str(i+1))(
                splittensor(ratio_split=2,id_split=i)(conv_2)
            ) for i in range(2)], mode='concat',concat_axis=1,name="conv_2")

        conv_3 = MaxPooling2D((3, 3), strides=(2, 2))(conv_2)
        conv_3 = crosschannelnormalization()(conv_3)
        conv_3 = ZeroPadding2D((1,1))(conv_3)
        conv_3 = Convolution2D(384,3,3,activation='relu',name='conv_3',init='he_normal')(conv_3)
```

AlexNet - University of Toronto (Krizhevsky, A. et al, 2012)




```
conv_4 = ZeroPadding2D((1,1))(conv_3)
conv_4 = merge([
    Convolution2D(192,3,3,activation="relu", init='he_normal', name='conv_4_'+str(i+1))(
        splittensor(ratio_split=2,id_split=i)(conv_4)
    ) for i in range(2)], mode='concat',concat_axis=1,name="conv_4")

conv_5 = ZeroPadding2D((1,1))(conv_4)
conv_5 = merge([
    Convolution2D(128,3,3,activation="relu",init='he_normal', name='conv_5_'+str(i+1))(
        splittensor(ratio_split=2,id_split=i)(conv_5)
    ) for i in range(2)], mode='concat',concat_axis=1,name="conv_5")

dense_1 = MaxPooling2D((3, 3), strides=(2,2),name="convpool_5")(conv_5)

dense_1 = Flatten(name="flatten")(dense_1)
dense_1 = Dense(4096, activation='relu',name='dense_1',init='he_normal')(dense_1)
dense_2 = Dropout(0.5)(dense_1)
dense_2 = Dense(4096, activation='relu',name='dense_2',init='he_normal')(dense_2)
dense_3 = Dropout(0.5)(dense_2)
dense_3 = Dense(nb_classes,name='dense_3_new',init='he_normal')(dense_3)

prediction = Activation("softmax",name="softmax")(dense_3)

alexnet = Model(input=inputs, output=prediction)

return alexnet
```

VGG-Net – University of Oxford (Simonyan, K., & Zisserman, A., 2014)



Critical Feautures (Simonyan, K., & Zisserman, A., 2014):

Kernels with small receptive fields: 3x3 which is the smallest size to capture the notion of left/right up/down, center. It is easy to see that a stack of two 3×3 conv. layers (without spatial pooling in between) has an effective receptive field of 5×5, and so on.

Small size Receptive Field is a way to increase the nonlinearity of the decision function fields of the conv. layers.

Increasing depth architectures: VGG-16 (2xConv3-64, 2xConv3-128, 3xConv3-256, 6xConv3-512, 3xFC), VGG-19 (same as VGG-16 but with 8xConv3-512).

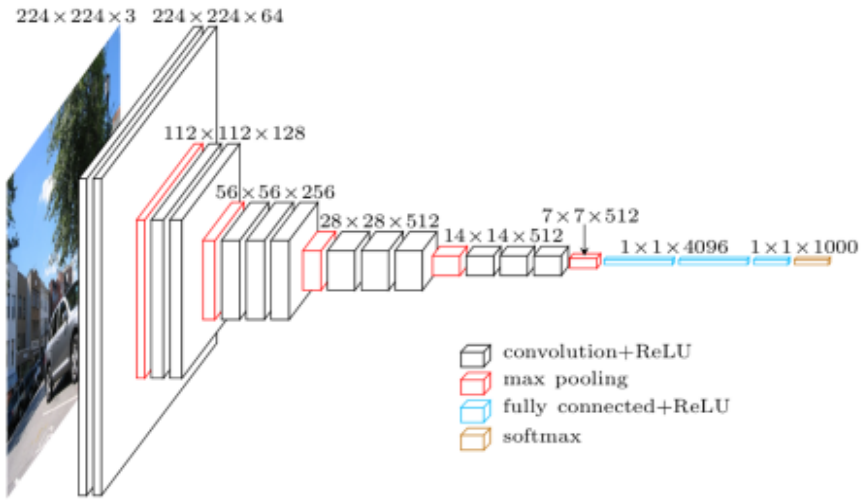
Upside: less complex topology, outperforms GoogleNet on single-network classification accuracy

Downside: 138 millions parameters for VGG-16 !

Results:

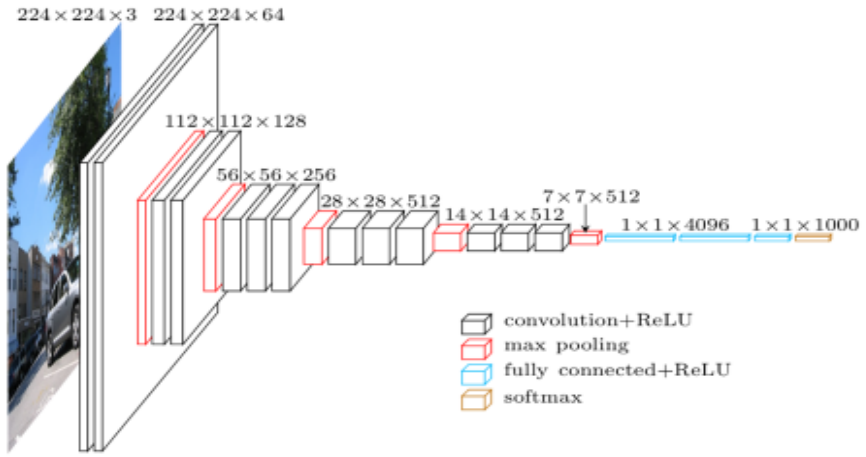
Multi ConvNet model : (D/[256;512]/256,384,512), (E/[256;512]/256,384,512), multi-crop & dense eval: **23.7% Top-1 Error, 6.8% Top-5 Error.**

VGG-Net – University of Oxford (Simonyan, K., & Zisserman, A., 2014)



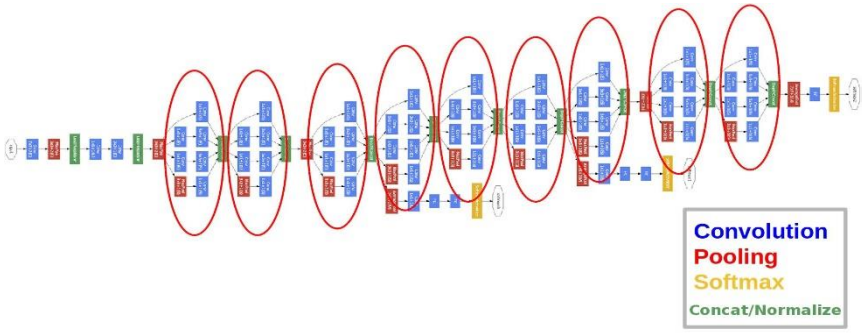

```
class VGG_16:

    @staticmethod
    def build(width, height, depth, classes, mul_factor, summary, weightsPath=None):

        model = Sequential()
        model.add(ZeroPadding2D((1,1),input_shape=(depth, height, width)))
        model.add(Convolution2D(64, 3, 3, activation='relu'))
        model.add(ZeroPadding2D((1,1)))
        model.add(Convolution2D(64, 3, 3, activation='relu'))
        model.add(MaxPooling2D((2,2), strides=(2,2)))

        model.add(ZeroPadding2D((1,1)))
        model.add(Convolution2D(128, 3, 3, activation='relu'))
        model.add(ZeroPadding2D((1,1)))
        model.add(Convolution2D(128, 3, 3, activation='relu'))
        model.add(MaxPooling2D((2,2), strides=(2,2)))

        model.add(ZeroPadding2D((1,1)))
        model.add(Convolution2D(256, 3, 3, activation='relu'))
        model.add(ZeroPadding2D((1,1)))
        model.add(Convolution2D(256, 3, 3, activation='relu'))
        model.add(ZeroPadding2D((1,1)))
        model.add(Convolution2D(256, 3, 3, activation='relu'))
        model.add(MaxPooling2D((2,2), strides=(2,2)))
```

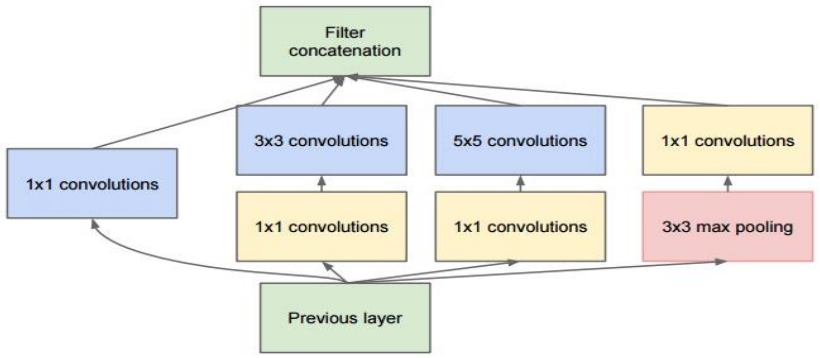
VGG-Net – University of Oxford (Simonyan, K., & Zisserman, A., 2014)




```
model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(512, 3, 3, activation='relu'))
model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(512, 3, 3, activation='relu'))
model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(512, 3, 3, activation='relu'))
model.add(MaxPooling2D((2,2), strides=(2,2)))

model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(512, 3, 3, activation='relu'))
model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(512, 3, 3, activation='relu'))
model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(512, 3, 3, activation='relu'))
model.add(MaxPooling2D((2,2), strides=(2,2)))

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

if summary==True:
    model.summary()

if weightsPath:
    model.load_weights(weightsPath)

return model
```

GoogleNet – Google (Szegedy, C., et al., 2015)





Critical Feautures (Szegedy, C., et al., 2015):

Computationally Effective Deep architecture: 22 layers

Why the name inception, you ask? Because the module represents a network within a network. If you don't get the reference, go watch Christopher Nolan's ***"INCEPTION"***, computer scientists are hilarious.

Inception: it isbasically the parallel combination of 1×1, 3×3, and 5×5 convolutional filters.

Bottleneck layer: The great insight of the inception module is the use of 1×1 convolutional blocks (NiN) to reduce the number of features before the expensive parallel blocks.

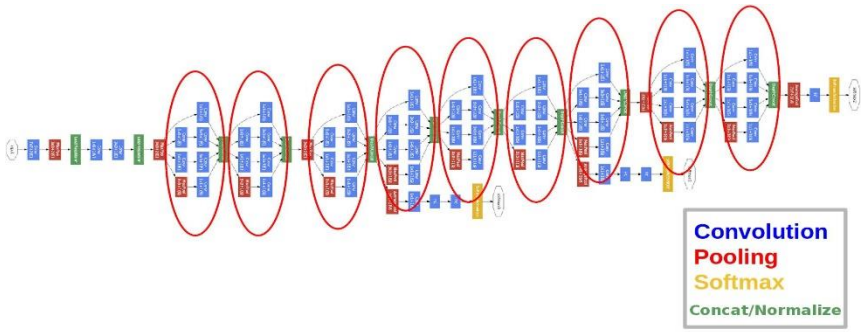
Upside: 4 millions parameters!

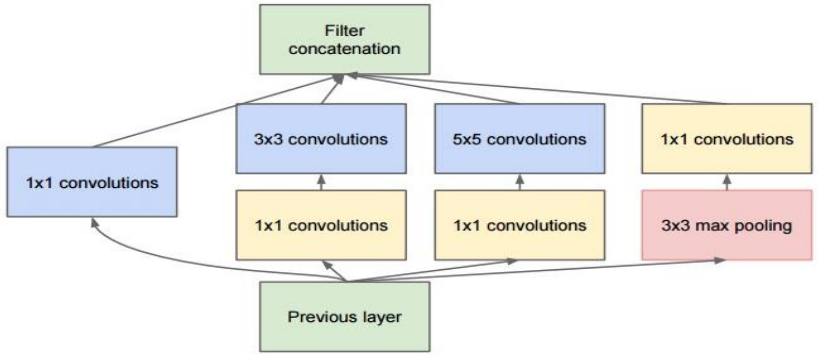
Downside: Not scalable!

Results:

7 Models Ensemble : 6.67% Top-5 Error.

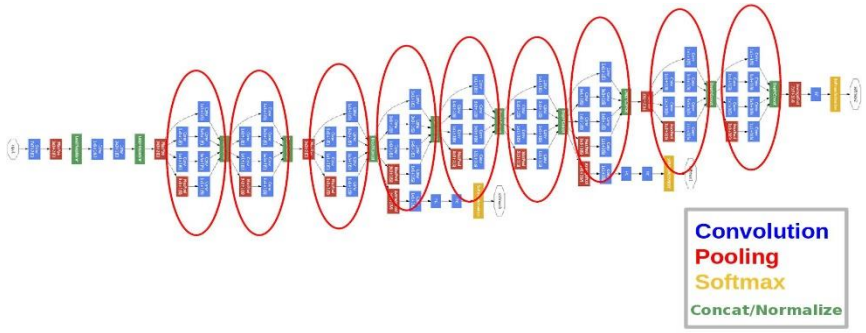
GoogleNet – Google (Szegedy, C., et al., 2015)

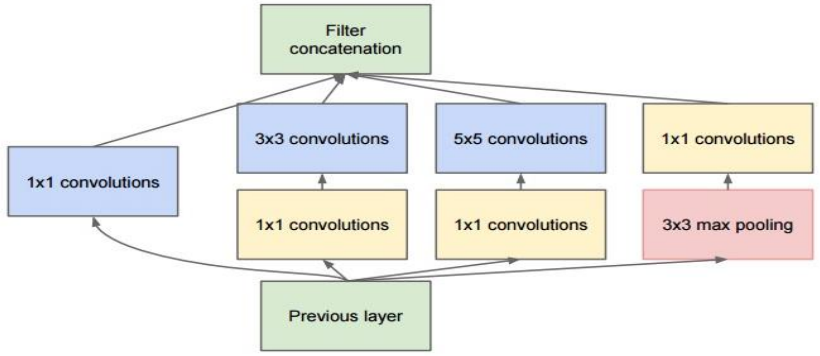





```
class GoogleNet:

    @staticmethod
    def build(width, height, depth, classes, mul_factor, weightsPath=None):
        input = Input(shape=(depth, height, width))                                    # Set Input Shape
        conv1_7x7_s2 = Convolution2D(64,7,7,subsample=(2,2),border_mode='same',activation='relu',name='conv1/7x7_s2',W_regularizer=l2(0.0002))(input)
        conv1_zero_pad = ZeroPadding2D(padding=(1, 1))(conv1_7x7_s2)
        pool1_helper = PoolHelper()(conv1_zero_pad)
        pool1_3x3_s2 = MaxPooling2D(pool_size=(3,3),strides=(2,2),border_mode='valid',name='pool1/3x3_s2')(pool1_helper)
        pool1_norm1 = LRN(name='pool1/norm1')(pool1_3x3_s2)
        conv2_3x3_reduce = Convolution2D(64,1,1,border_mode='same',activation='relu',name='conv2/3x3_reduce',W_regularizer=l2(0.0002))(pool1_norm1)
        conv2_3x3 = Convolution2D(192,3,3,border_mode='same',activation='relu',name='conv2/3x3',W_regularizer=l2(0.0002))(conv2_3x3_reduce)
        conv2_norm2 = LRN(name='conv2/norm2')(conv2_3x3)
        conv2_zero_pad = ZeroPadding2D(padding=(1, 1))(conv2_norm2)
        pool2_helper = PoolHelper()(conv2_zero_pad)
        pool2_3x3_s2 = MaxPooling2D(pool_size=(3,3),strides=(2,2),border_mode='valid',name='pool2/3x3_s2')(pool2_helper)

        # First Inception Module
        inception_3a_1x1 = Convolution2D(64,1,1,border_mode='same',activation='relu',name='inception_3a/1x1',W_regularizer=l2(0.0002))(pool2_3x3_s2)
        inception_3a_3x3_reduce = Convolution2D(96,1,1,border_mode='same',activation='relu',name='inception_3a/3x3_reduce',W_regularizer=l2(0.0002))(pool2_3x3_s2)
        inception_3a_3x3 = Convolution2D(128,3,3,border_mode='same',activation='relu',name='inception_3a/3x3',W_regularizer=l2(0.0002))(inception_3a_3x3_reduce)
        inception_3a_5x5_reduce = Convolution2D(16,1,1,border_mode='same',activation='relu',name='inception_3a/5x5_reduce',W_regularizer=l2(0.0002))(pool2_3x3_s2)
        inception_3a_5x5 = Convolution2D(32,5,5,border_mode='same',activation='relu',name='inception_3a/5x5',W_regularizer=l2(0.0002))(inception_3a_5x5_reduce)
        inception_3a_pool = MaxPooling2D(pool_size=(3,3),strides=(1,1),border_mode='same',name='inception_3a/pool')(pool2_3x3_s2)
        inception_3a_pool_proj = Convolution2D(32,1,1,border_mode='same',activation='relu',name='inception_3a/pool_proj',W_regularizer=l2(0.0002))(inception_3a_pool)
        inception_3a_output = merge([inception_3a_1x1,inception_3a_3x3,inception_3a_5x5,inception_3a_pool_proj],mode='concat',concat_axis=1,name='inception_3a/output')
```

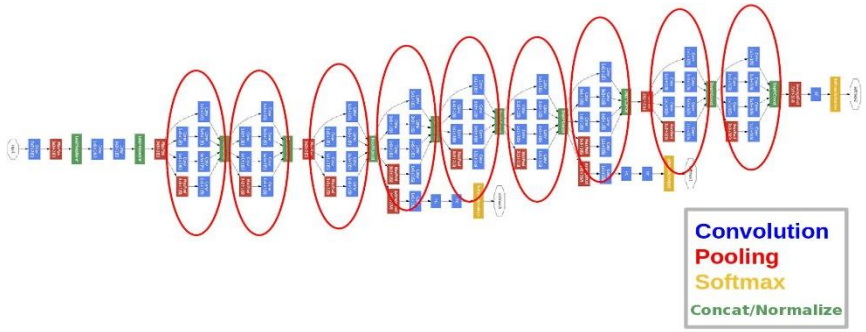
GoogleNet – Google (Szegedy, C., et al., 2015)

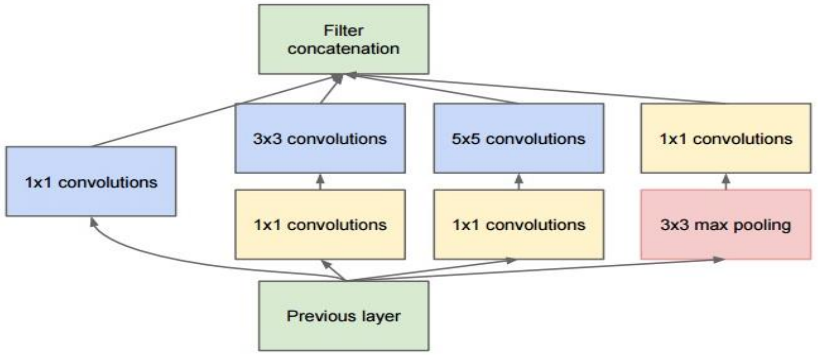





```
# Second Inception Module
inception_3b_1x1 = Convolution2D(128,1,1,border_mode='same',activation='relu',name='inception_3b/1x1',W_regularizer=l2(0.0002))(inception_3a_output)
inception_3b_3x3_reduce = Convolution2D(128,1,1,border_mode='same',activation='relu',name='inception_3b/3x3_reduce',W_regularizer=l2(0.0002))(inception_3a_output)
inception_3b_3x3 = Convolution2D(192,3,3,border_mode='same',activation='relu',name='inception_3b/3x3',W_regularizer=l2(0.0002))(inception_3b_3x3_reduce)
inception_3b_5x5_reduce = Convolution2D(32,1,1,border_mode='same',activation='relu',name='inception_3b/5x5_reduce',W_regularizer=l2(0.0002))(inception_3a_output)
inception_3b_5x5 = Convolution2D(96,5,5,border_mode='same',activation='relu',name='inception_3b/5x5',W_regularizer=l2(0.0002))(inception_3b_5x5_reduce)
inception_3b_pool = MaxPooling2D(pool_size=(3,3),strides=(1,1),border_mode='same',name='inception_3b/pool')(inception_3a_output)
inception_3b_pool_proj = Convolution2D(64,1,1,border_mode='same',activation='relu',name='inception_3b/pool_proj',W_regularizer=l2(0.0002))(inception_3b_pool)
inception_3b_output = merge([inception_3b_1x1,inception_3b_3x3,inception_3b_5x5,inception_3b_pool_proj],mode='concat',concat_axis=1,name='inception_3b/output')
inception_3b_output_zero_pad = ZeroPadding2D(padding=(1, 1))(inception_3b_output)
pool3_helper = PoolHelper()(inception_3b_output_zero_pad)
pool3_3x3_s2 = MaxPooling2D(pool_size=(3,3),strides=(2,2),border_mode='valid',name='pool3/3x3_s2')(pool3_helper)

# Third Inception Module
inception_4a_1x1 = Convolution2D(192,1,1,border_mode='same',activation='relu',name='inception_4a/1x1',W_regularizer=l2(0.0002))(pool3_3x3_s2)
inception_4a_3x3_reduce = Convolution2D(96,1,1,border_mode='same',activation='relu',name='inception_4a/3x3_reduce',W_regularizer=l2(0.0002))(pool3_3x3_s2)
inception_4a_3x3 = Convolution2D(208,3,3,border_mode='same',activation='relu',name='inception_4a/3x3',W_regularizer=l2(0.0002))(inception_4a_3x3_reduce)
inception_4a_5x5_reduce = Convolution2D(16,1,1,border_mode='same',activation='relu',name='inception_4a/5x5_reduce',W_regularizer=l2(0.0002))(pool3_3x3_s2)
inception_4a_5x5 = Convolution2D(48,5,5,border_mode='same',activation='relu',name='inception_4a/5x5',W_regularizer=l2(0.0002))(inception_4a_5x5_reduce)
inception_4a_pool = MaxPooling2D(pool_size=(3,3),strides=(1,1),border_mode='same',name='inception_4a/pool')(pool3_3x3_s2)
inception_4a_pool_proj = Convolution2D(64,1,1,border_mode='same',activation='relu',name='inception_4a/pool_proj',W_regularizer=l2(0.0002))(inception_4a_pool)
inception_4a_output = merge([inception_4a_1x1,inception_4a_3x3,inception_4a_5x5,inception_4a_pool_proj],mode='concat',concat_axis=1,name='inception_4a/output')
loss1_ave_pool = AveragePooling2D(pool_size=(5,5),strides=(3,3),name='loss1/ave_pool')(inception_4a_output)
loss1_conv = Convolution2D(128,1,1,border_mode='same',activation='relu',name='loss1/conv',W_regularizer=l2(0.0002))(loss1_ave_pool)
loss1_flat = Flatten()(loss1_conv)
loss1_fc = Dense(1024,activation='relu',name='loss1/fc',W_regularizer=l2(0.0002))(loss1_flat)
loss1_drop_fc = Dropout(0.7)(loss1_fc)
loss1_classifier = Dense(1000,name='loss1/classifier',W_regularizer=l2(0.0002))(loss1_drop_fc)
loss1_classifier_act = Activation('softmax')(loss1_classifier)
```

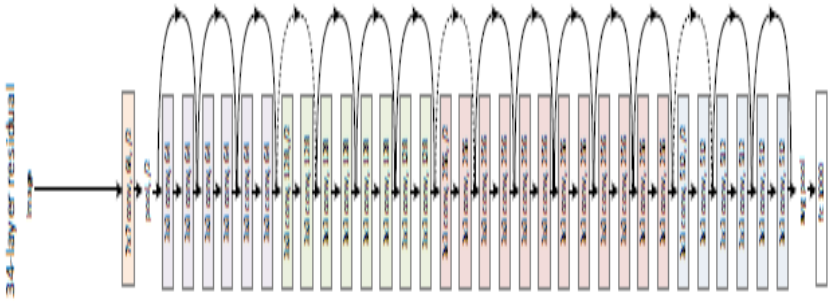
GoogleNet – Google (Szegedy, C., et al., 2015)

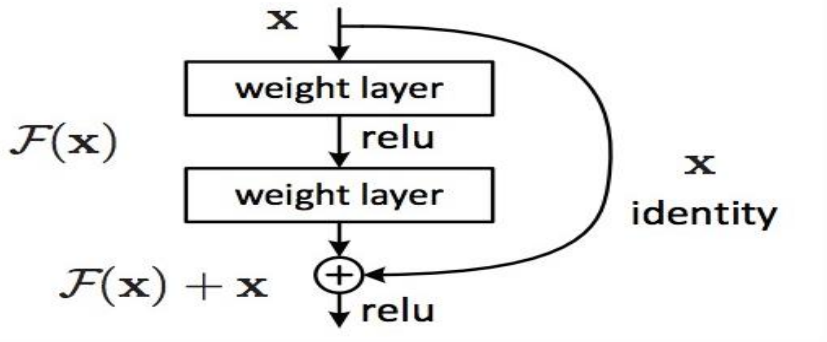





```
# Fourth Inception Module
inception_4b_1x1 = Convolution2D(160,1,1,border_mode='same',activation='relu',name='inception_4b/1x1',W_regularizer=l2(0.0002))(inception_4a_output)
inception_4b_3x3_reduce = Convolution2D(112,1,1,border_mode='same',activation='relu',name='inception_4b/3x3_reduce',W_regularizer=l2(0.0002))(inception_4a_output)
inception_4b_3x3 = Convolution2D(224,3,3,border_mode='same',activation='relu',name='inception_4b/3x3',W_regularizer=l2(0.0002))(inception_4b_3x3_reduce)
inception_4b_5x5_reduce = Convolution2D(24,1,1,border_mode='same',activation='relu',name='inception_4b/5x5_reduce',W_regularizer=l2(0.0002))(inception_4a_output)
inception_4b_5x5 = Convolution2D(64,5,5,border_mode='same',activation='relu',name='inception_4b/5x5',W_regularizer=l2(0.0002))(inception_4b_5x5_reduce)
inception_4b_pool = MaxPooling2D(pool_size=(3,3),strides=(1,1),border_mode='same',name='inception_4b/pool')(inception_4a_output)
inception_4b_pool_proj = Convolution2D(64,1,1,border_mode='same',activation='relu',name='inception_4b/pool_proj',W_regularizer=l2(0.0002))(inception_4b_pool)
inception_4b_output = merge([inception_4b_1x1,inception_4b_3x3,inception_4b_5x5,inception_4b_pool_proj],mode='concat',concat_axis=1,name='inception_4b_output')


model = Model(input=input, output=[loss1_classifier])


model.summary()


#if a weights path is supplied (indicating that the model was pre-trained), then load the weights
if weightsPath is not None:
    model.load_wights(weightsPath)


return model
```

Residual Networks – Microsoft (He, K., et al., 2016)





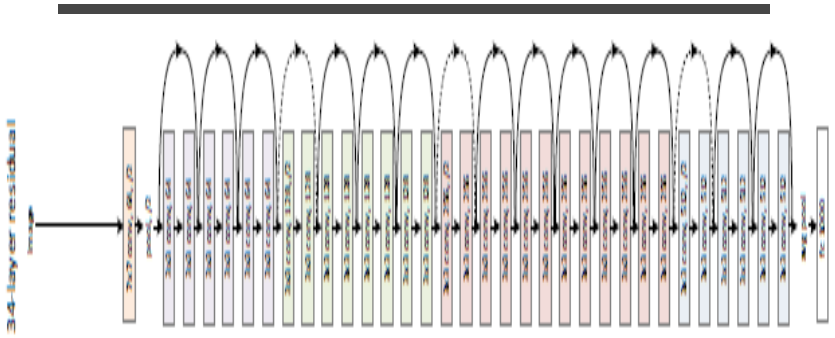
Critical Feautures

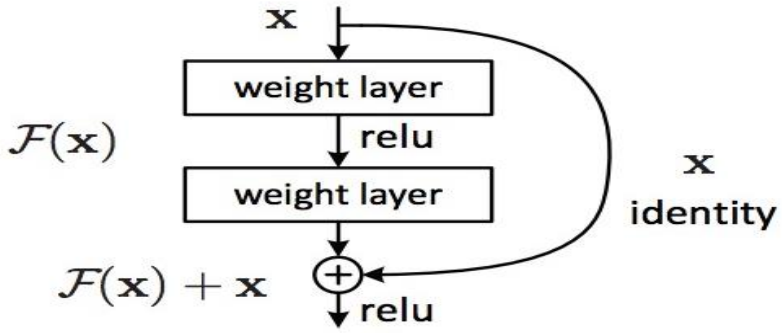
- **Degradation Problem:** Stacking more and more layers **IS NOT** better. With the network depth increasing, accuracy gets saturated and then degrades rapidly! It's an issue of "solvers".
- **Solves the "Degradation problem":** by fitting a residual mapping which is easier to optimize.
- **Shortcut connections:**
- **Very deep architecture:** up to 1202 layers with WideResnet with only **19.4 millon parameters**!
- **Upside: Increasing accuracy with more depth**
- **Downside: They don't consider other architectures breakthroughs.**

Results:

- **ResNet : 3.57%** Top-5 Error.
- **CNNs** show superhuman abilities at Image Recognition! **5%** Human estimated Top-5 error (Johnson, R. C., 2015).

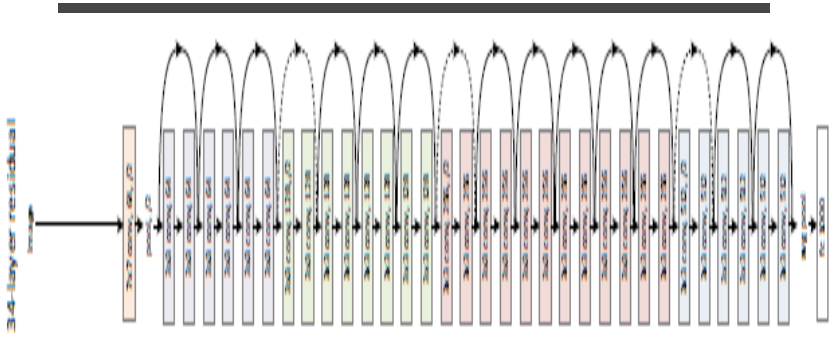
Residual Networks – Microsoft (He, K., et al., 2016)

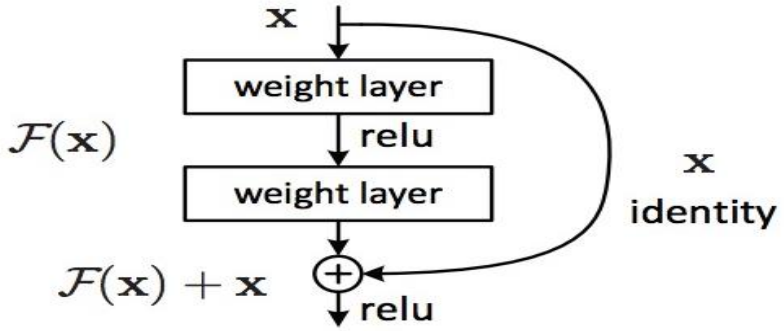





```python
class WideResNet:

    @staticmethod
    def build(width, height, depth, classes, summary, weightsPath=None):
        n = 8  # depth = 6*n + 4
        k = 4  # widen factor

        img_input = Input(shape=(depth, height, width))

        # one conv at the beginning (spatial size: 32x32)
        x = ZeroPadding2D((1, 1))(img_input)
        x = Conv2D(16, (3, 3))(x)

        # Stage 1 (spatial size: 32x32)
        x = bottleneck(x, n, 16, 16 * k, dropout=0.3, subsample=(1, 1))
        # Stage 2 (spatial size: 16x16)
        x = bottleneck(x, n, 16 * k, 32 * k, dropout=0.3, subsample=(2, 2))
        # Stage 3 (spatial size: 8x8)
        x = bottleneck(x, n, 32 * k, 64 * k, dropout=0.3, subsample=(2, 2))

        x = BatchNormalization(axis=1)(x)
        x = Activation('relu')(x)
        x = AveragePooling2D((8, 8), strides=(1, 1))(x)
        x = Flatten()(x)
        preds = Dense(classes, activation='softmax')(x)

        model = Model(inputs=img_input, outputs=preds)

        if summary==True:
            model.summary()

        #model = to_multi_gpu(model, 2)

        #if a weights path is supplied (indicating that the model was pre-trained), then load the weights
        if weightsPath is not None:
            model.load_wights(weightsPath)

        return model
```

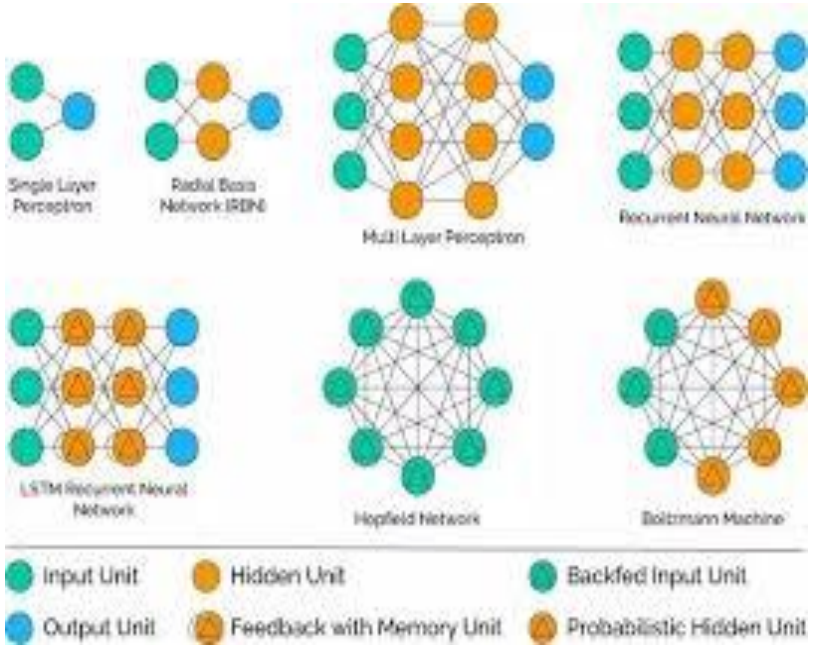
Residual Networks – Microsoft (He, K., et al., 2016)

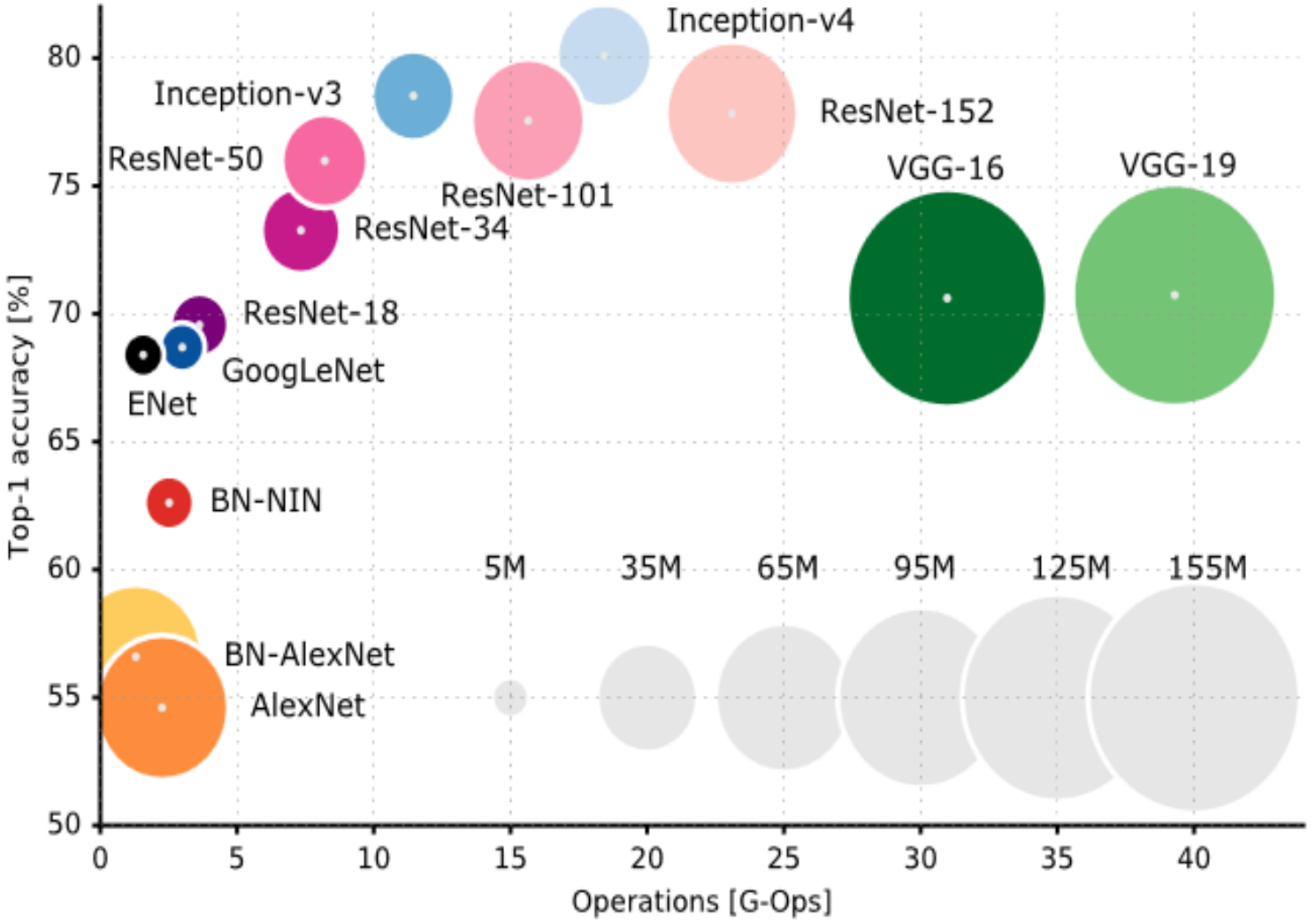




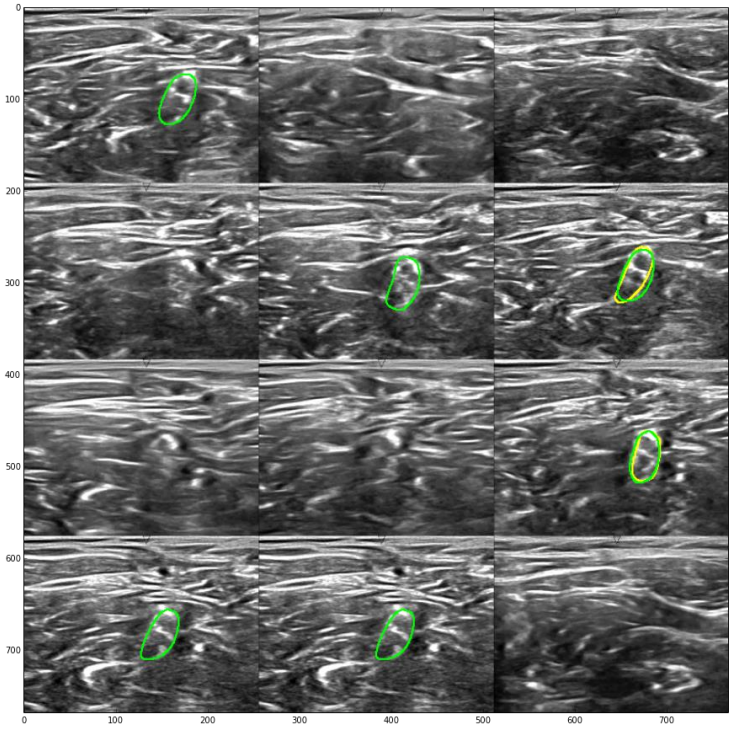

```python
def bottleneck(incoming, count, nb_in_filters, nb_out_filters, dropout=None, subsample=(2, 2)):
    outgoing = wide_basic(incoming, nb_in_filters, nb_out_filters, dropout, subsample)
    for i in range(1, count):
        outgoing = wide_basic(outgoing, nb_out_filters, nb_out_filters, dropout, subsample=(1, 1))

    return outgoing


def wide_basic(incoming, nb_in_filters, nb_out_filters, dropout=None, subsample=(2, 2)):
    nb_bottleneck_filter = nb_out_filters

    if nb_in_filters == nb_out_filters:
        # conv3x3
        y = BatchNormalization(axis=1)(incoming)
        y = Activation('relu')(y)
        y = ZeroPadding2D((1, 1))(y)
        y = Conv2D(nb_bottleneck_filter, (3, 3), strides=subsample, kernel_initializer='he_normal', padding='valid')(y)

        # conv3x3
        y = BatchNormalization(axis=1)(y)
        y = Activation('relu')(y)
        if dropout is not None:
            y = Dropout(dropout)(y)
        y = ZeroPadding2D((1, 1))(y)
        y = Conv2D(nb_bottleneck_filter, (3, 3), strides=(1, 1), kernel_initializer='he_normal', padding='valid')(y)

        return add([incoming, y])

    else:  # Residual Units for increasing dimensions
        # common BN, ReLU
        shortcut = BatchNormalization(axis=1)(incoming)
        shortcut = Activation('relu')(shortcut)

        # conv3x3
        y = ZeroPadding2D((1, 1))(shortcut)
        y = Conv2D(nb_bottleneck_filter, (3, 3), strides=subsample, kernel_initializer='he_normal', padding='valid')(y)

        # conv3x3
        y = BatchNormalization(axis=1)(y)
        y = Activation('relu')(y)
        if dropout is not None:
            y = Dropout(dropout)(y)
        y = ZeroPadding2D((1, 1))(y)
        y = Conv2D(nb_out_filters, (3, 3), strides=(1, 1), kernel_initializer='he_normal', padding='valid')(y)

        # shortcut
        shortcut = Conv2D(nb_out_filters, (1, 1), strides=subsample, kernel_initializer='he_normal', padding='same')(shortcut)

    return add([shortcut, y])
```

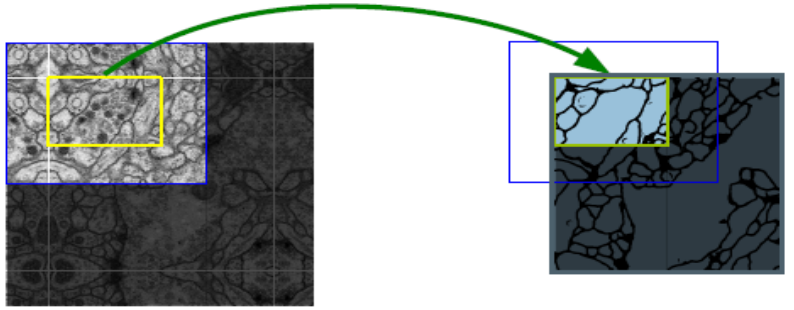
Neural Network Architectures





Segmentation problem: alias with biomedical images CNNs fail



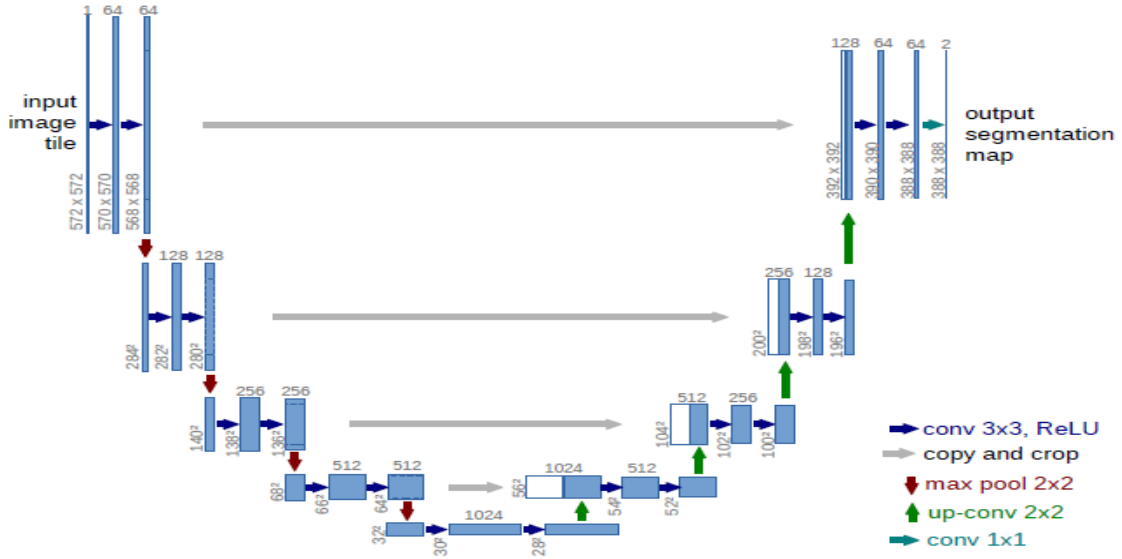


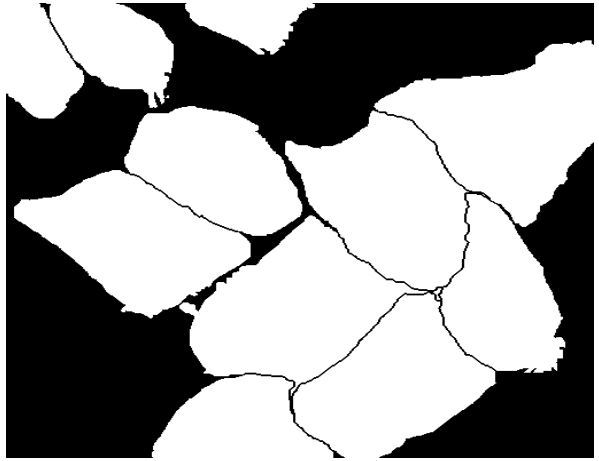
Problems:

Feature extraction: In biomedicine feature extraction is not as easy as in an Imagenet competition with general images. A previous Image Preprocessing is needed. This is called Segmentation.

On Kaggle website there are whole competitions just regarding Segmentation. One of these was called «Ultrasound Nerve Segmentation».

U-NET (Fully connected CNN)





Critical Feautures (Ronneberger, O., et al., 2015):

U-NET can be trained end-to-end from very few images and outperforms the prior best methods.

It consists of a contracting path (left side) to capture context and an symmetric expansive path (right side) enabling precise localization.

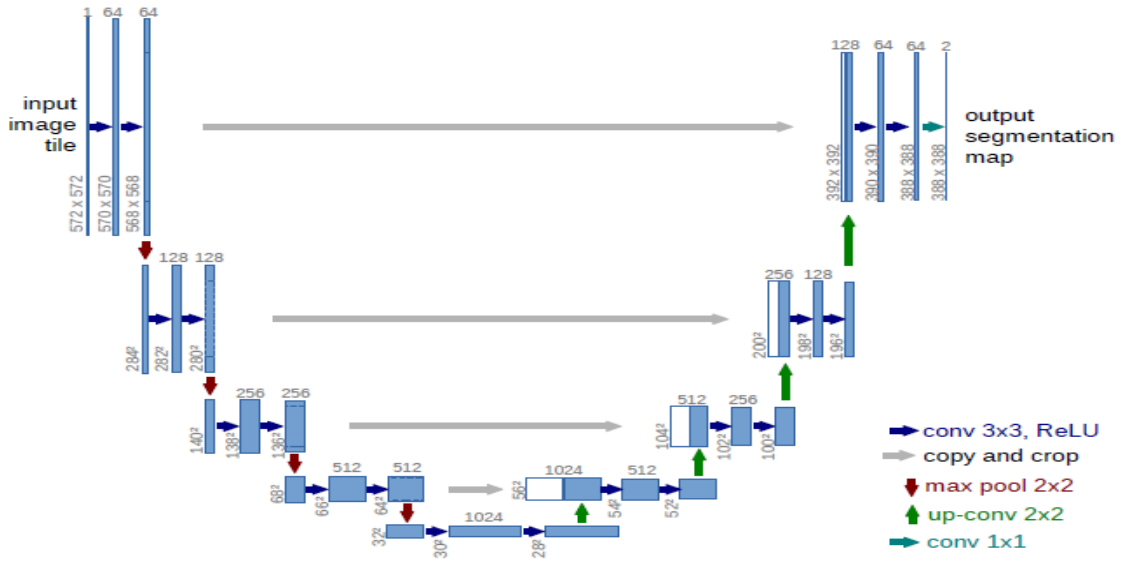
Upsampling part (repeating rows and cols) has a large number of feature channels which allow the network to propagate context information to higher resolution layers.

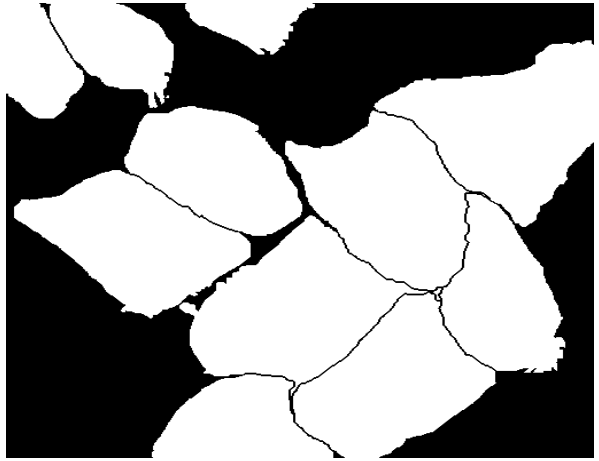
Spatial Dropout: feature maps dropout.

Upside: Small training set.

Downside: Risk of overfitting.

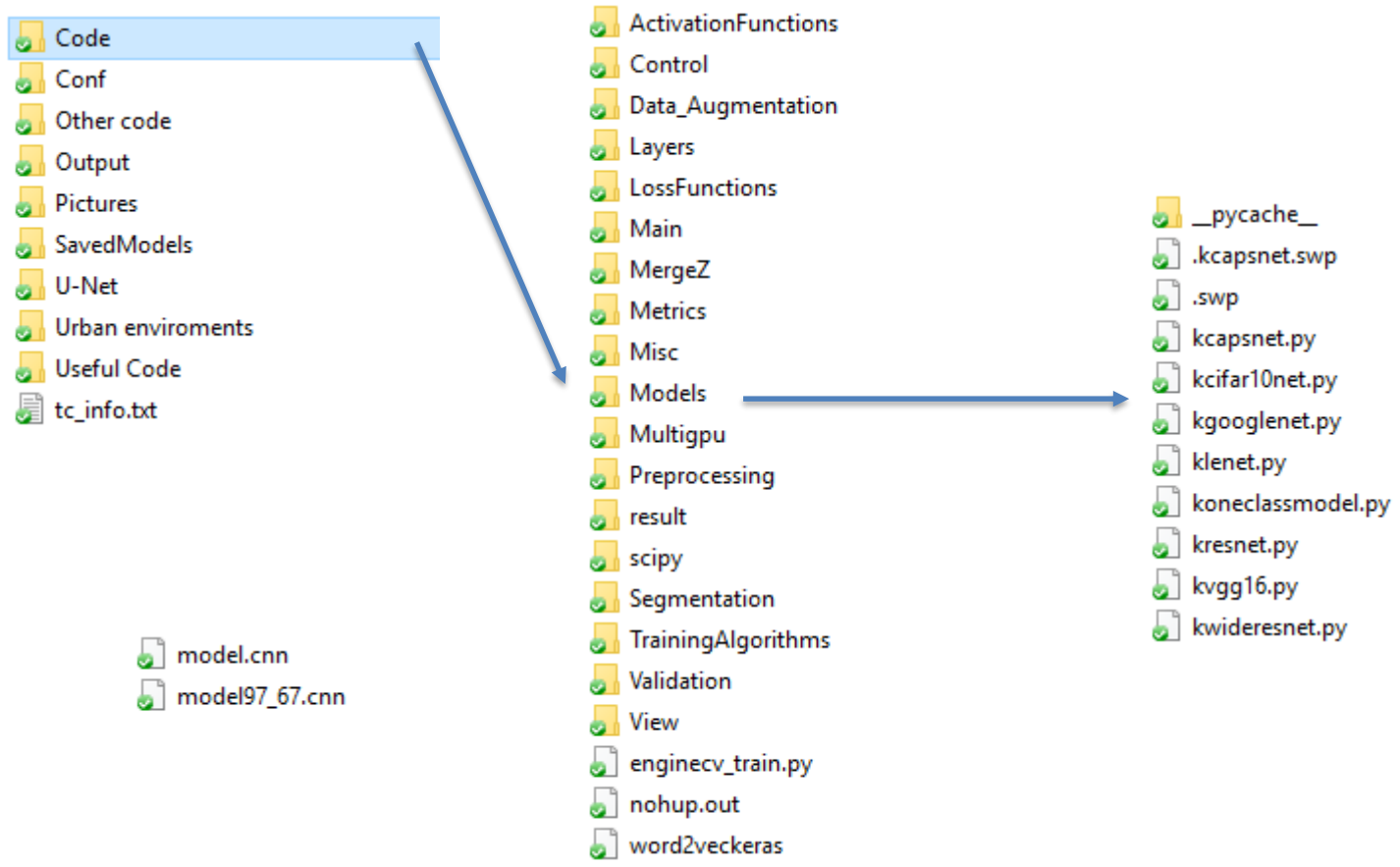
U-NET in Keras





```
def get_unet():
    inputs = Input((1,img_rows, img_cols))
    conv1 = Convolution2D(32, 3, 3, activation='relu', border_mode='same')(inputs)
    conv1 = Convolution2D(32, 3, 3, activation='relu', border_mode='same')(conv1)
    pool1 = MaxPooling2D(pool_size=(2, 2))(conv1)

    conv2 = Convolution2D(64, 3, 3, activation='relu', border_mode='same')(pool1)
    conv2 = Convolution2D(64, 3, 3, activation='relu', border_mode='same')(conv2)
    pool2 = MaxPooling2D(pool_size=(2, 2))(conv2)

    conv3 = Convolution2D(128, 3, 3, activation='relu', border_mode='same')(pool2)
    conv3 = Convolution2D(128, 3, 3, activation='relu', border_mode='same')(conv3)
    pool3 = MaxPooling2D(pool_size=(2, 2))(conv3)

    conv4 = Convolution2D(256, 3, 3, activation='relu', border_mode='same')(pool3)
    conv4 = Convolution2D(256, 3, 3, activation='relu', border_mode='same')(conv4)
    pool4 = MaxPooling2D(pool_size=(2, 2))(conv4)

    conv5 = Convolution2D(512, 3, 3, activation='relu', border_mode='same')(pool4)
    conv5 = Convolution2D(512, 3, 3, activation='relu', border_mode='same')(conv5)

    up6 = merge([UpSampling2D(size=(2, 2))(conv5), conv4], mode='concat', concat_axis=1)
    conv6 = Convolution2D(256, 3, 3, activation='relu', border_mode='same')(up6)
    conv6 = Convolution2D(256, 3, 3, activation='relu', border_mode='same')(conv6)

    up7 = merge([UpSampling2D(size=(2, 2))(conv6), conv3], mode='concat', concat_axis=1)
    conv7 = Convolution2D(128, 3, 3, activation='relu', border_mode='same')(up7)
    conv7 = Convolution2D(128, 3, 3, activation='relu', border_mode='same')(conv7)

    up8 = merge([UpSampling2D(size=(2, 2))(conv7), conv2], mode='concat', concat_axis=1)
    conv8 = Convolution2D(64, 3, 3, activation='relu', border_mode='same')(up8)
    conv8 = Convolution2D(64, 3, 3, activation='relu', border_mode='same')(conv8)

    up9 = merge([UpSampling2D(size=(2, 2))(conv8), conv1], mode='concat', concat_axis=1)
    conv9 = Convolution2D(32, 3, 3, activation='relu', border_mode='same')(up9)
    conv9 = Convolution2D(32, 3, 3, activation='relu', border_mode='same')(conv9)

    conv10 = Convolution2D(1, 1, 1, activation='sigmoid')(conv9)

    model = Model(input=inputs, output=conv10)
```

Computer Vision Back-End Structure



Exercise: Classification of Cifar 10 with LeNet in Keras

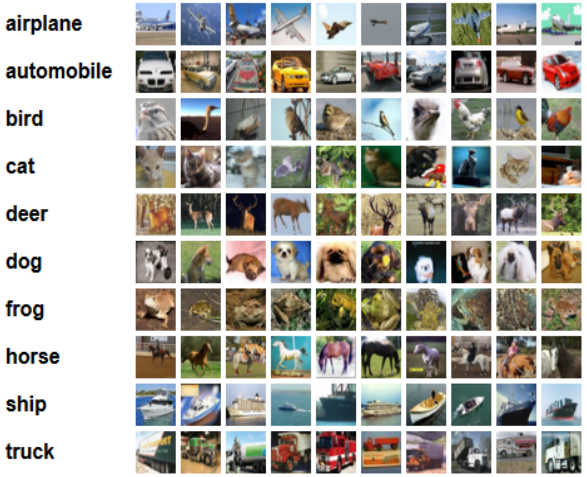
Cifar10 Dataset

The CIFAR-10 dataset

The CIFAR-10 dataset consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images.

The dataset is divided into five training batches and one test batch, each with 10000 images. The test batch contains exactly 1000 randomly-selected images from each class. The training batches contain the remaining images in random order, but some training batches may contain more images from one class than another. Between them, the training batches contain exactly 5000 images from each class.

Here are the classes in the dataset, as well as 10 random images from each:



The classes are completely mutually exclusive. There is no overlap between automobiles and trucks. "Automobile" includes sedans, SUVs, things of that sort. "Truck" includes only big trucks. Neither includes pickup trucks.

Exercise: Classification of Cifar 10 with LeNet in Keras

Pre-processing

```
train_set_x = numpy.zeros((50000, 3, 32, 32), dtype='uint8')
train_set_y = numpy.zeros((50000,), dtype='uint8')

# Merges all the data batches
# Loads Cifar10 train set
for i in range(1, 6):
    train_path = os.path.join(os.path.split(__file__)[0], cifar10_dataset_path, 'data_batch_' + str(i))
    data, labels = cifar.load_batch(train_path)
    train_set_x[(i - 1) * 10000: i * 10000, :, :, :] = data
    train_set_y[(i - 1) * 10000: i * 10000] = labels

# Loads Cifar10 test set
test_path = os.path.join(os.path.split(__file__)[0], cifar10_dataset_path, 'test_batch')
test_set_x, test_set_y = cifar.load_batch(test_path)

train_set_y = numpy.reshape(train_set_y, (len(train_set_y), 1))                # convert train row vector
test_set_y = numpy.reshape(test_set_y, (len(test_set_y), 1))                   # convert test row vector i
```

Exercise: Classification of Cifar 10 with LeNet in Keras

Pre-processing

```
# Convert class vectors to binary class matrices.
train_set_y = np_utils.to_categorical(train_set_y, output_size)
test_set_y = np_utils.to_categorical(test_set_y, output_size)

if training == True:

    # Normalization
    train_set_x = train_set_x.astype('float32')
    test_set_x = test_set_x.astype('float32')
    train_set_x /= 255
    test_set_x /= 255
```

Exercise: Classification of Cifar 10 with LeNet in Keras

Model Definition

```
if benchmark_neural_model.lower() == "lenet":
    topology = LeNet
elif benchmark_neural_model.lower() == "cifar10net":
    topology = Cifar10Net
elif benchmark_neural_model.lower() == "wideresnet":
    topology = WideResNet
elif benchmark_neural_model.lower() == "capsnet":
    topology = CapsuleNet
else:
    print("Model does not exist.")
    sys.exit()

if training == True:
    if benchmark_neural_model.lower() == "capsnet":
        model=CapsuleNet.build_caps(cifar10_input_size, cifar10_input_size, cifar10_input_channels, cifar10_output_size, True, weightsPath=None)
        CapsuleNet.train(model, data, args)
    else:
        deepnetwork = topology.build(width=input_size[0], height=input_size[1], depth=input_channels, classes=output_size, summary=True)
        deepnetwork.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
    checkPoint=ModelCheckpoint(os.path.join("./",models_path)+"/best_cifar10_model.cnn", monitor='val_acc', verbose=1, save_best_only=True, mode='max')
```

Exercise: Classification of Cifar 10 with LeNet in Keras

Model Training

```
deepnetwork.fit(train_set_x, train_set_y, batch_size=batch_size, epochs=epochs_number, validation_data = (test_set_x, test_set_y), callbacks = default_callbacks, shuffle = True, verbose=2)
```

Model Evaluation

```
print ('\n\nTesting the Best Trained Network...\n')
bestDeepNetwork = topology.build(width=input_size[0], height=input_size[1], depth=input_channels, classes=output_size, summary=False)
bestDeepNetwork.load_weights(os.path.join("./",models_path)+"/best_cifar10_model.cnn")
bestDeepNetwork.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
[loss, accuracy] = bestDeepNetwork.evaluate(test_set_x, test_set_y, verbose = 1)
print ("\nAccuracy  : %.2f%%" % (accuracy * 100))
print ("Loss      : %f" % loss)
```

References

Bengio, Y. (2009). Learning deep architectures for AI. Foundations and trends® in Machine Learning, 2(1), 1-127.

Hinton, G. E., & Salakhutdinov, R. R. (2006). Reducing the dimensionality of data with neural networks. *science*, *313*(5786), 504-507.

Chellappa, R. (2012). Mathematical statistics and computer vision. Image and Vision Computing, 30(8), 467-468.

Tsukahara, J. S., Harrison, T. L., & Engle, R. W. (2016). The relationship between baseline pupil size and intelligence. Cognitive Psychology, 91, 109-123.

Francesco Pugliese - Deep Learning al servizio del Riciclo https://www.wired.it/attualita/tech/2018/09/20/party-cloud-ibm-deep-learning/

Srivastava, N., Hinton, G. E., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. Journal of Machine Learning Research, 15(1), 1929-1958.

Francesco **Pugliese**