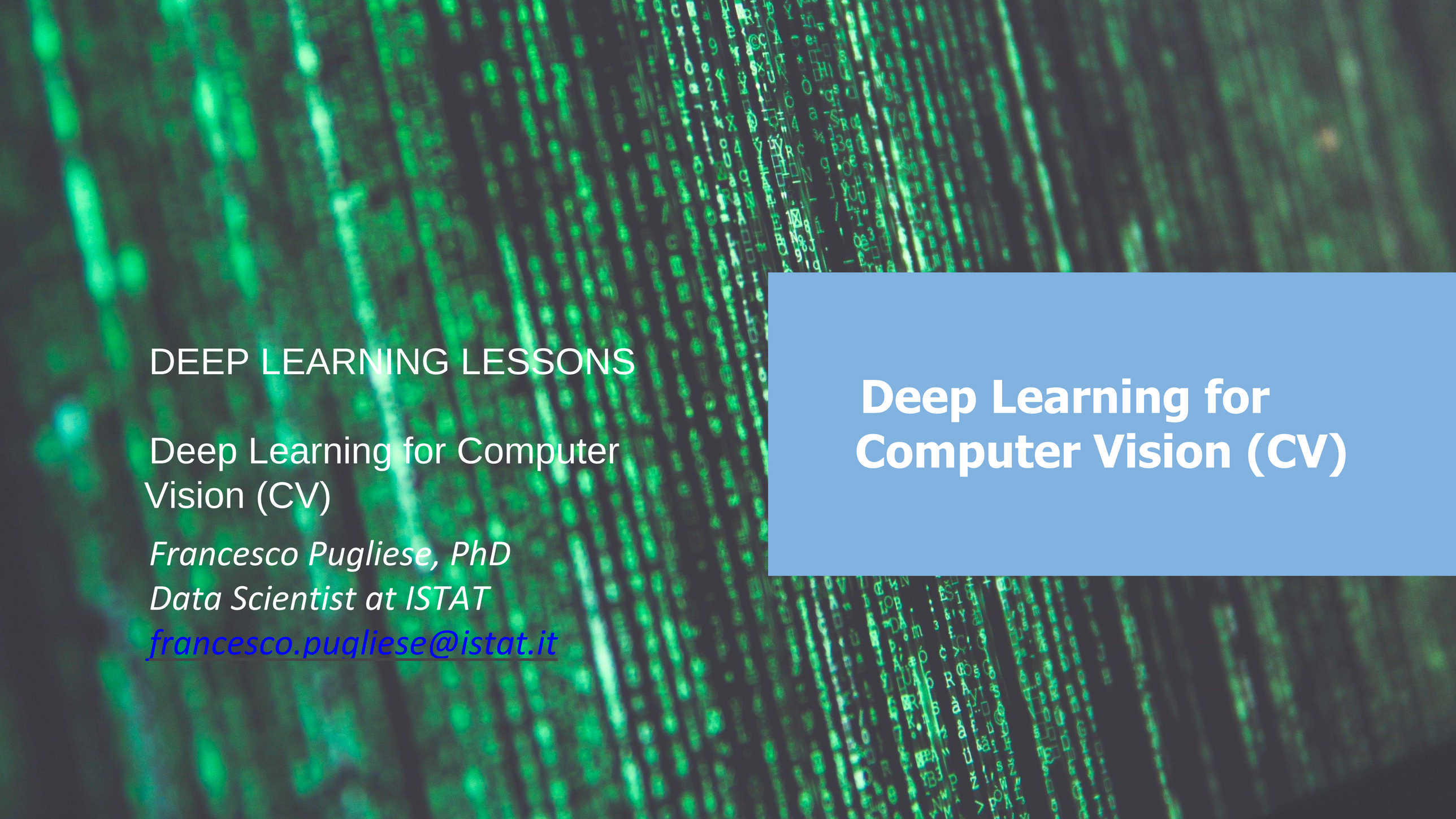


DEEP LEARNING LESSONS

Deep Learning for Computer Vision (CV)

Francesco Pugliese, PhD
Data Scientist at ISTAT
francesco.pugliese@istat.it

Deep Learning for Computer Vision (CV)

DEEP LEARNING: Neural Networks become more effective

GOOGLE DATACENTER

1,000 CPU Servers
2,000 CPUs • 16,000 cores

600 kWatts
$5,000,000

STANFORD AI LAB

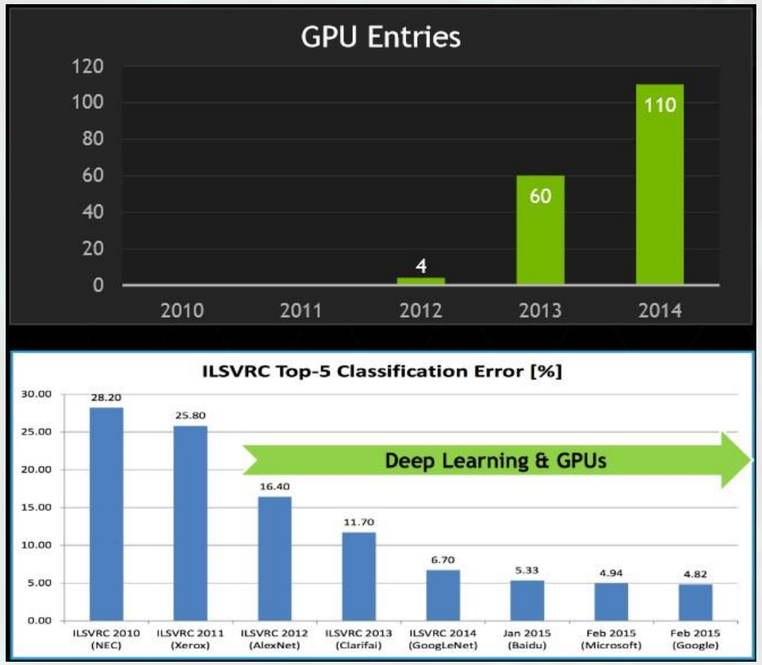
3 GPU-Accelerated Servers
12 GPUs • 18,432 cores

4 kWatts
$33,000

- In recent years **Deep Neural Networks** have achieved noticeably breakthroughs in research (Bengio, 2009). This new methodology dealing with deep neural networks and their training algorithms was called “Deep Learning” by Hinton in 2006, for the first time (). From that, Hinton became the godfather of Deep Learning and of the moder Artificial Intelligence, in general.

- So far, in all the experiments, the resulting performances of Deep Learning were many magnitudes better than other traditional machine learning techniques available.

DEEP LEARNING: a cutting-edge approach to Computer Vision and NLP



GPU

- The advent of **GPUs** makes possible the **training of very large neural networks** with even **more than 150 millions of parameters**.

BIG DATA

- A **new generation** of larger **training and test sets.**

DROPOUT

- Better **model regularization techniques** have been discovered such as ***"Dropout"*** , ***"Batch Normalization"***, or ***"Data Augmentation".***

Why Deep Learning over-performed traditional statistics models?

- “Deep Learning” approaches can be **end-to-end trained** without a task-specific feature engineering.
- **These model are scalable**: adding GPUs they can be trained faster.
- ***“Deep Learning is killing every problem in AI”*** (Elizabeth Gibney, 2016)

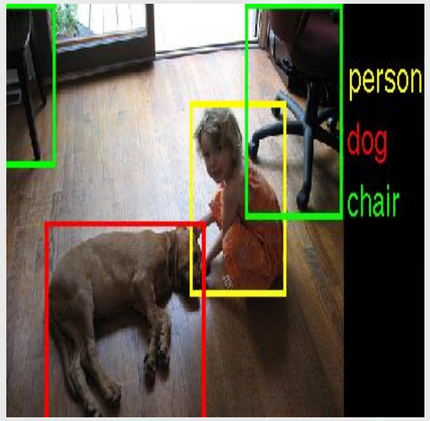
- Basically, **statistics is not able to deal with very high dimensionalities** of data as Deep Learning does.

Computer Vision: Where does Traditional Statistics fail?

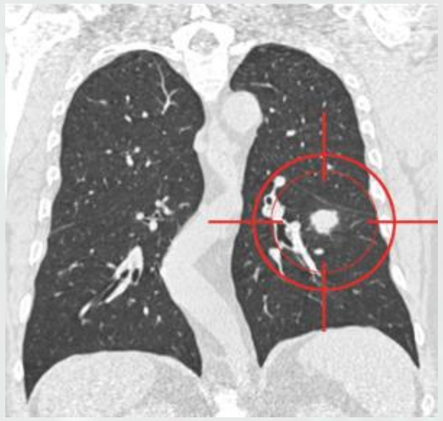
- **Computer Vision** is an interdisciplinary field that deals with the way algorithms can be made for gaining high-level understanding from digital images or videos.
- **Statistical methods** are not always welcome in computer vision.
- Statistical methods seem **not scaling up** to the challenges of computer vision problems (Chellappa, R., 2012).

Why does Computer Vision matter so much?

EVERYDAY LIFE

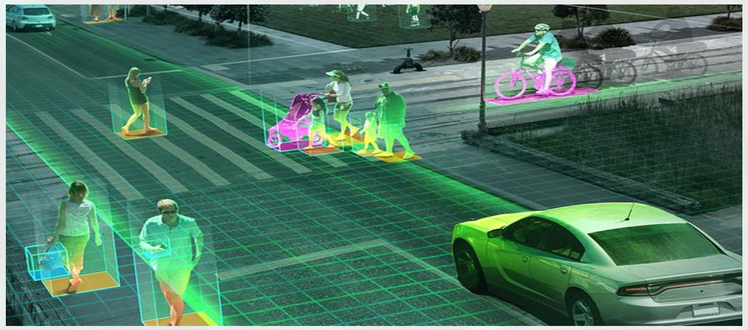


BIOMEDICAL IMAGES



- 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.

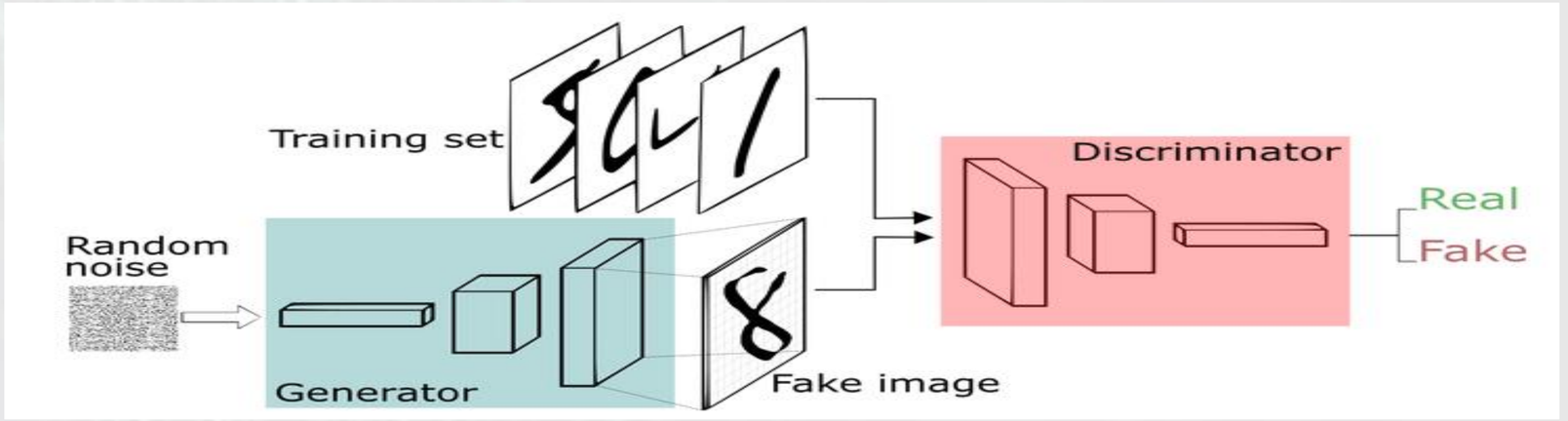
Computer Vision for Recycling

- **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.

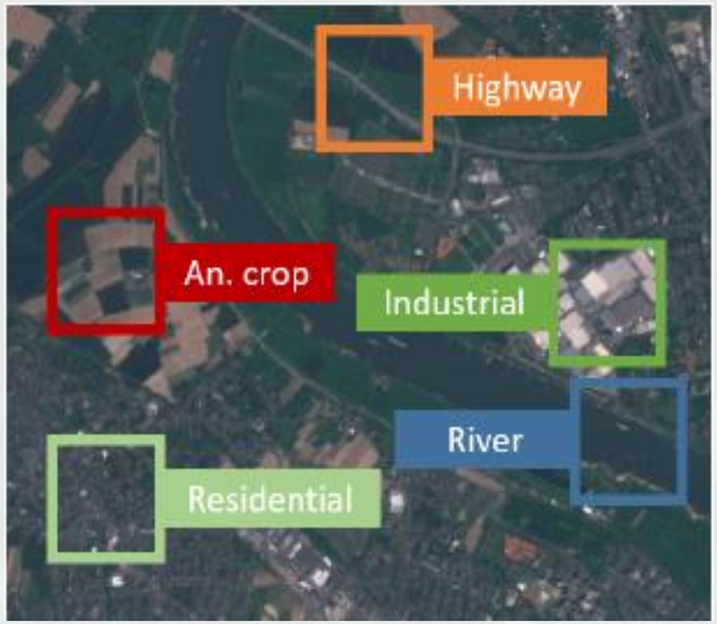
- **AI applications** which can optimize all the **flow of interactions** with a **restaurant**, decreasing the process costs and increasing sustainability *(«Deep Learning al servizio del riciclo», Intervista su Wired, Pugliese, F.)*. Thanks to **Artificial Intelligence**, in one work, some researchers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

Generative Adversarial Networks (GAN) (Goodfellow, et al.,2014)



$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))].$$

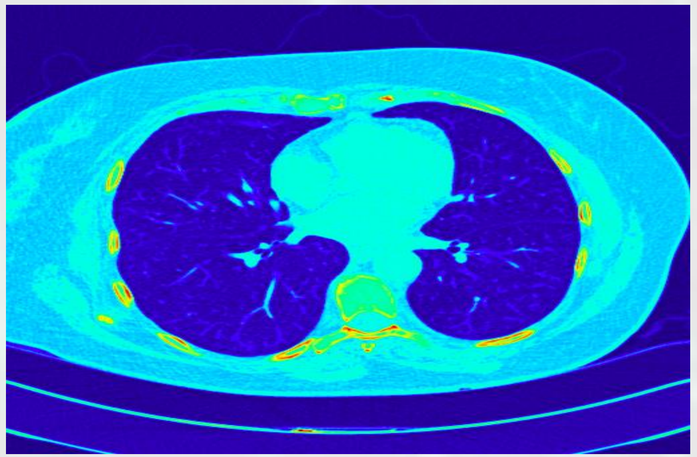
Automatic Extraction of Statistics from Satellite Imagery: Land Use and Land Cover Classification

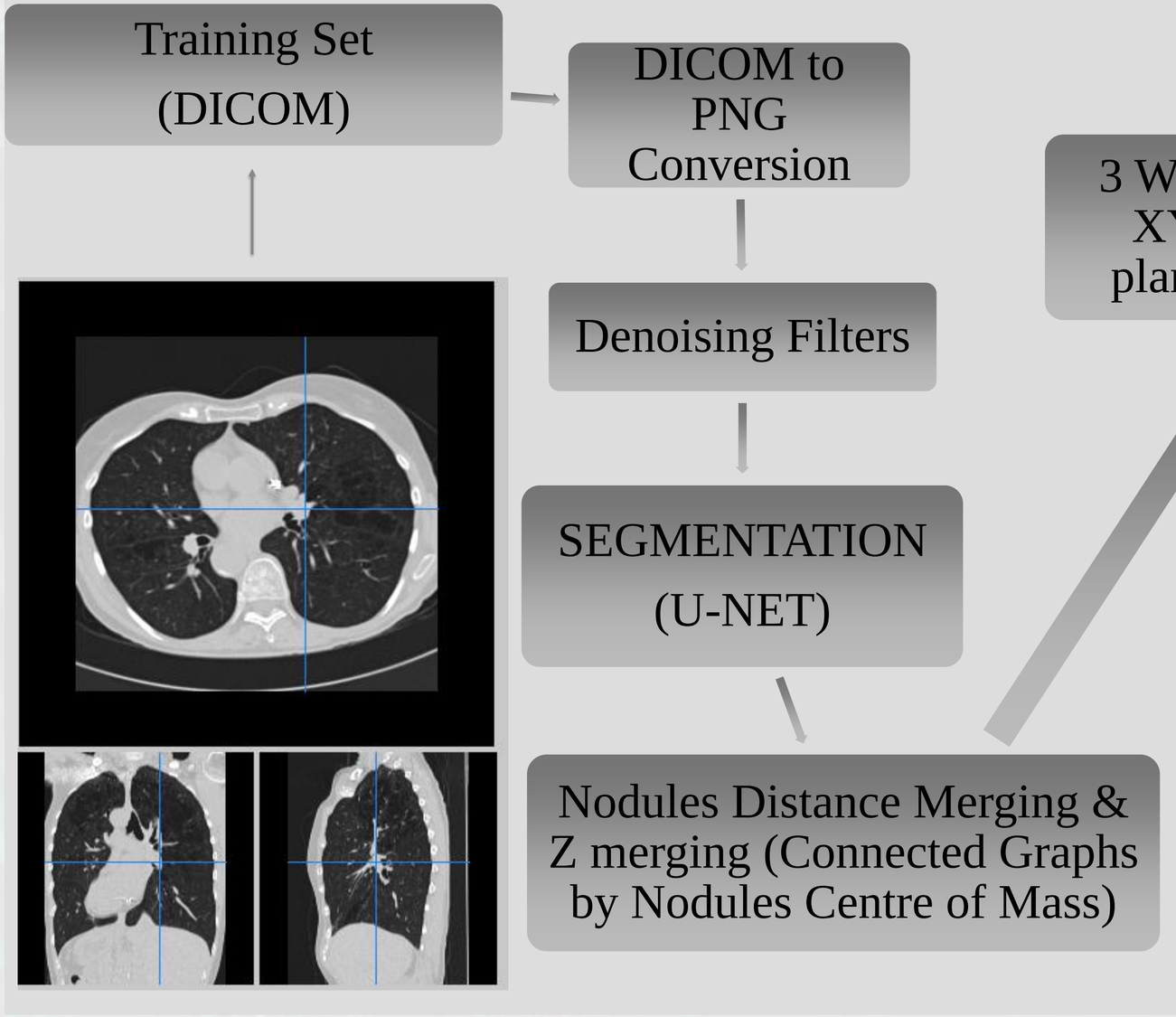


- Nowadays, more and more public and up-to-dated **satellite imagery** data for Earth observation are **available**.

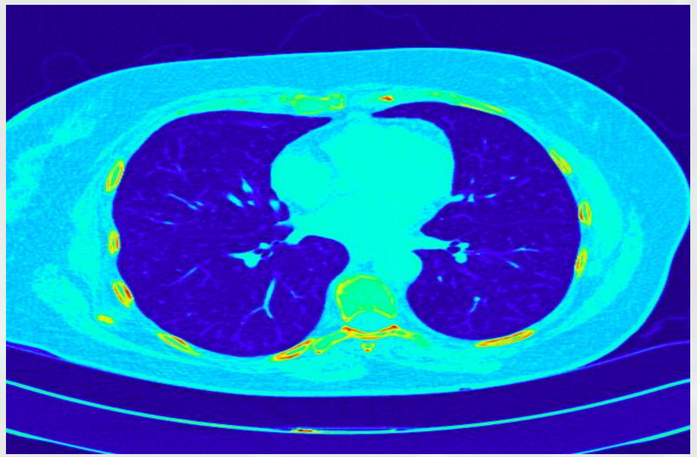
- However, to fully utilize this data, in order to automatically extract statistics, **satellite images** must be processed and transformed into **structured semantics**.

Lung Cancer Detection

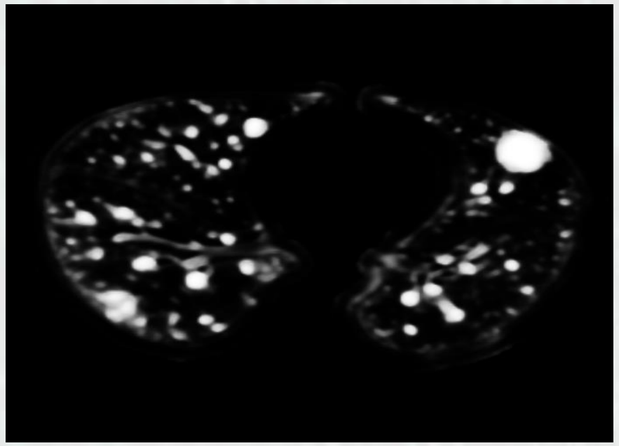




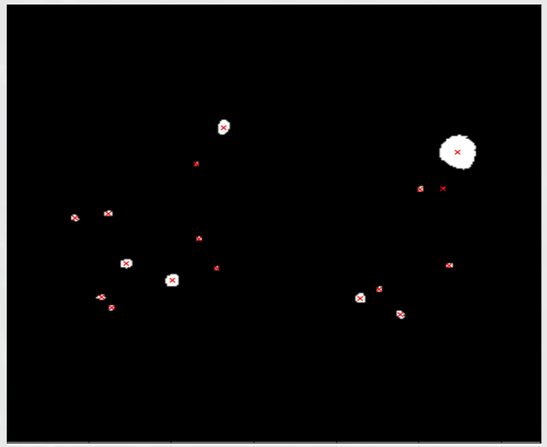
Lung Cancer Detection



Candidate Nodule Selection via UNET



Dilation, Erosion, Nodules Distance Merging



False Positive Reduction via WideResNet

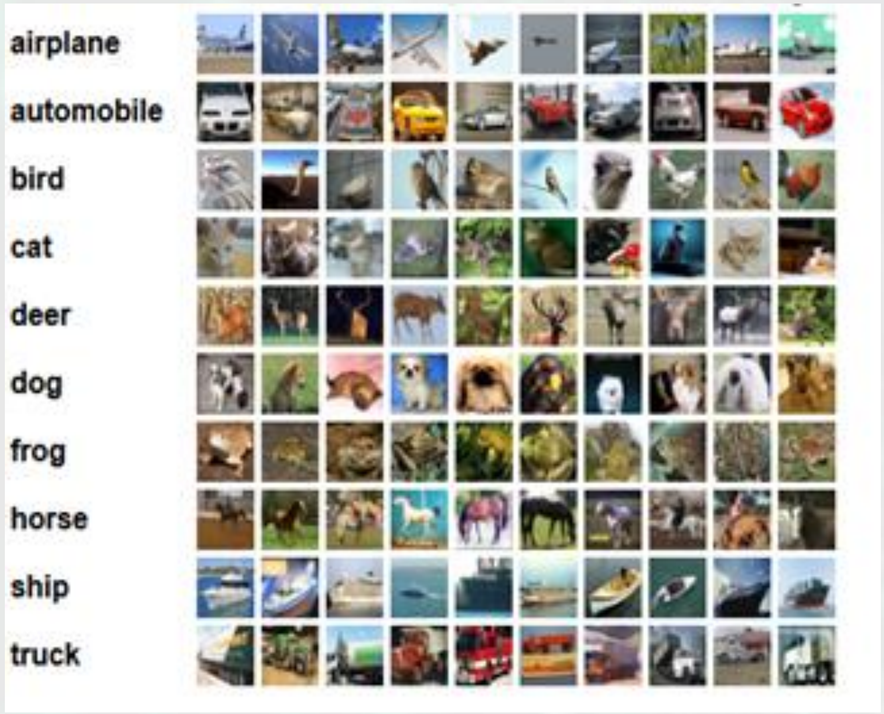
Cancer / Non cancer

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 Datasets and Competitions

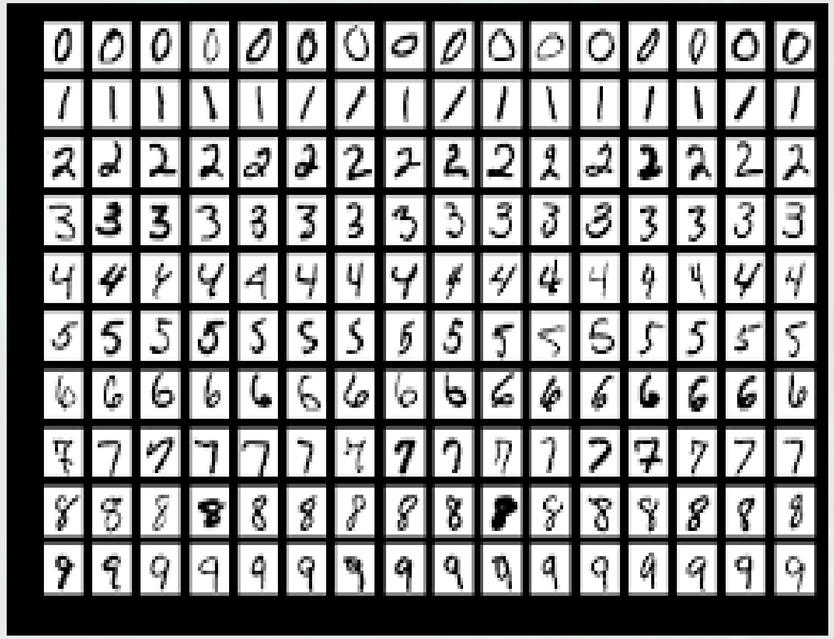


- The **CIFAR-10** dataset consists of **60,000 32x32** coloured images divided into **10 classes**, with **6000 images per class**. There are **50,000 training** images and **10,000 test** images.

- 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.

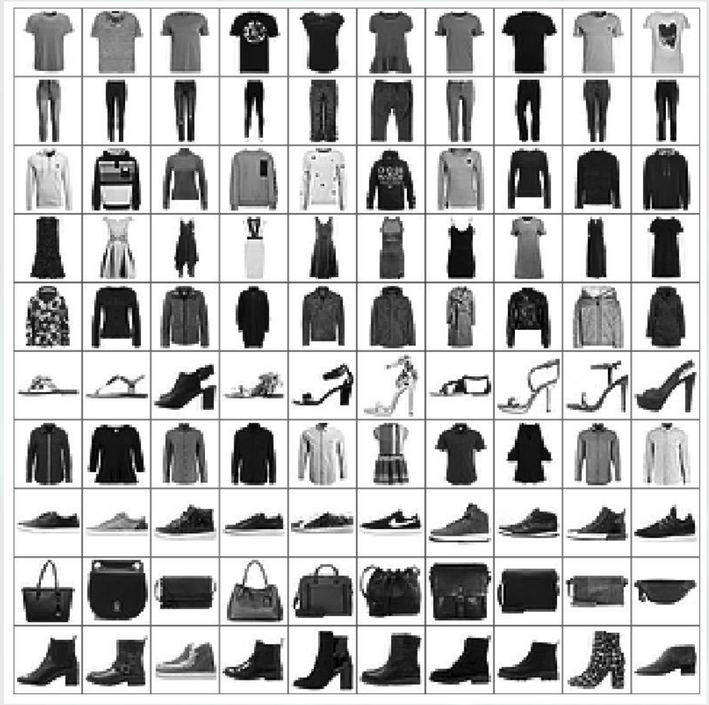
- The classes are completely **mutually exclusive**. There is **no overlap** between **automobiles** and **trucks**. "Automobile" includes sedans, SUVs, things of that sort. "Truck" includes only big trucks. Neither includes pickup trucks.

Computer Vision Datasets and Competitions



- The **MNIST** database of handwritten digits, available from this page, 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 Datasets and Competitions



- **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.

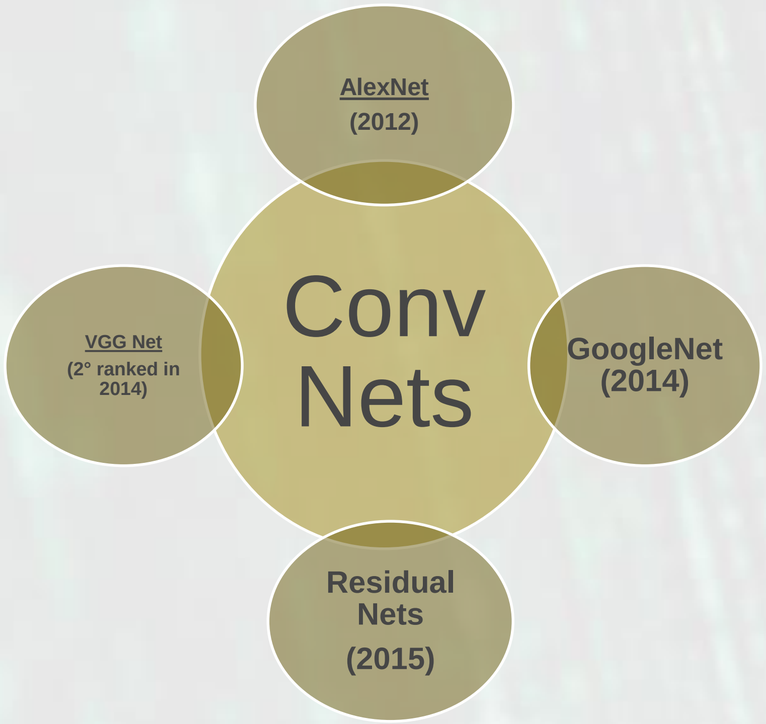
- Each example is a **28 x 28** grayscale image associated with a label from **10 classes**.

- 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**.

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).

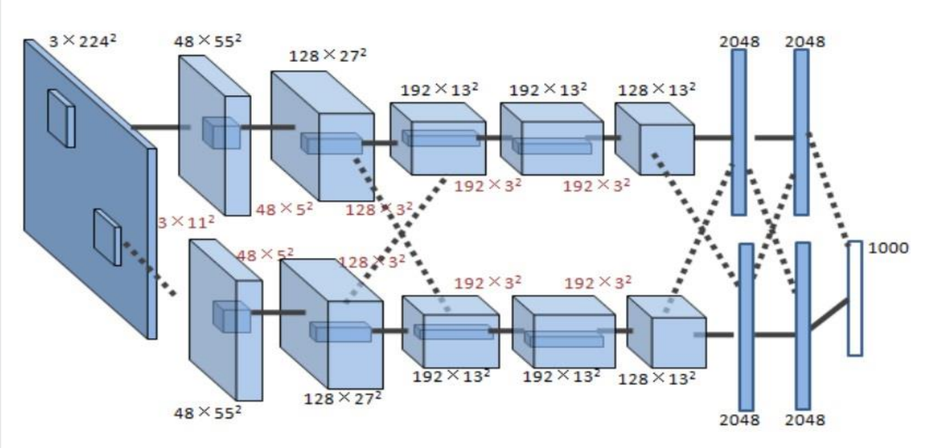
Model Regularization :

•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)



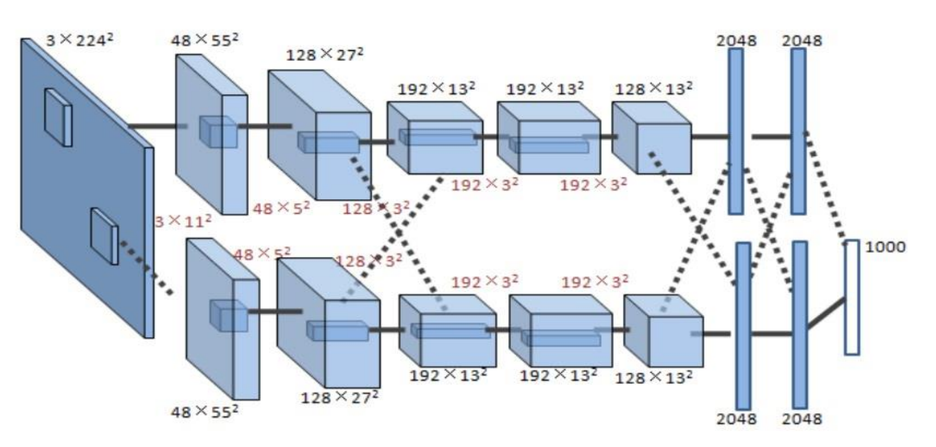
Critical Feautures:

- **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.[18]

Results:

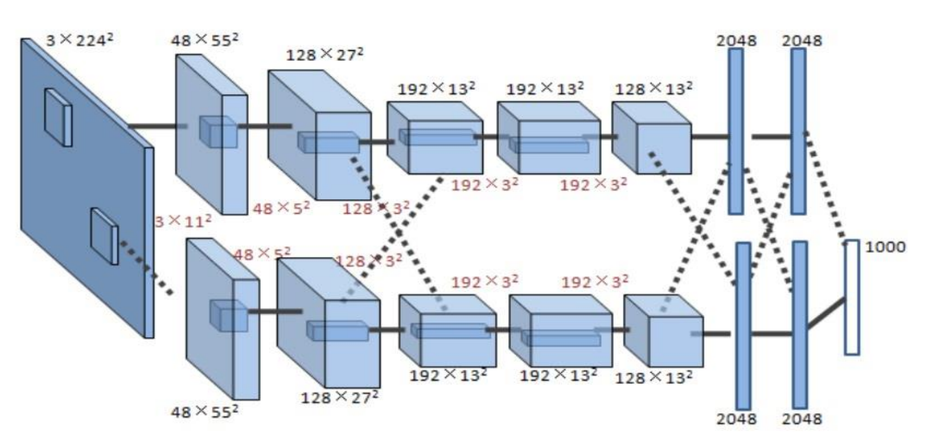
- **1 CNNs: 40.7%** Top-1 Error, **18.2%** Top-5 Error
- **5 CNNs: 38.1%** Top-1 Error,**16.4%** Top-5 Error
  - **SIFT+FVs: 26.2%** Top-5 Error (Sánchez, J., et al., 2013).

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



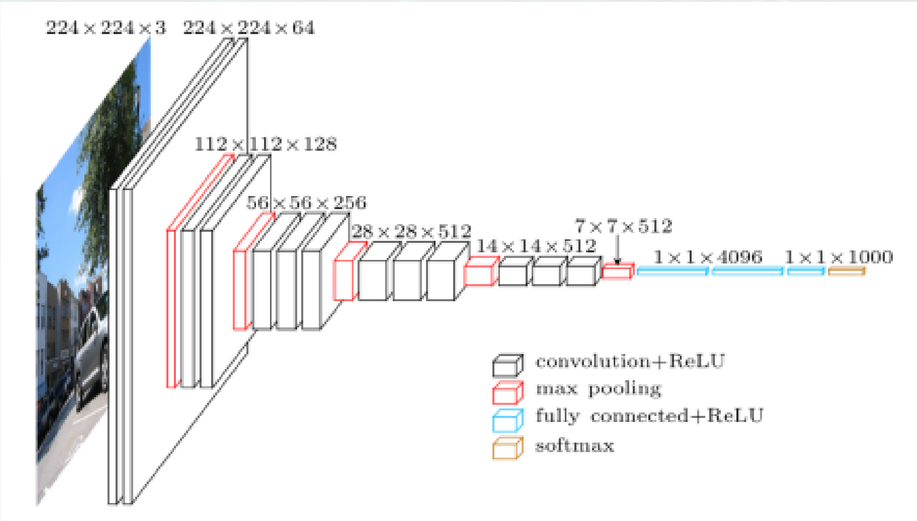

```
def get_alexnet(input_shape,nb_classes,mean_flag):
        # code adapted from https://github.com/heuritech/convnets-keras

        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:

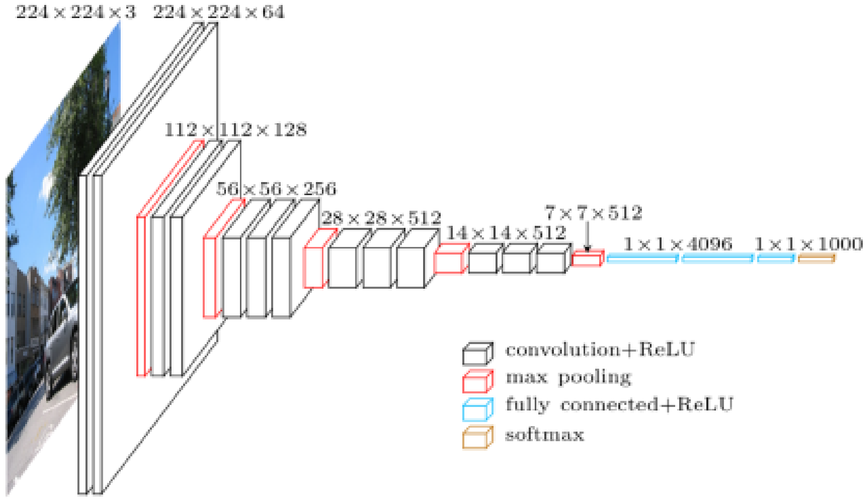
- **8 trainable layers:** 5 convolutional layers and 3 fully connected layers.
- **Max pooling layers** after 1st, 2nd and 5th layer.
- **Rectified Linear Units (ReLUs) (**Nair, V., & Hinton, G. E. 2010).
- **Local Response Normalization.**
- **60 millions parameters, 650 thousands neurons**.
- **Regularizations**: 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.

21

Results:

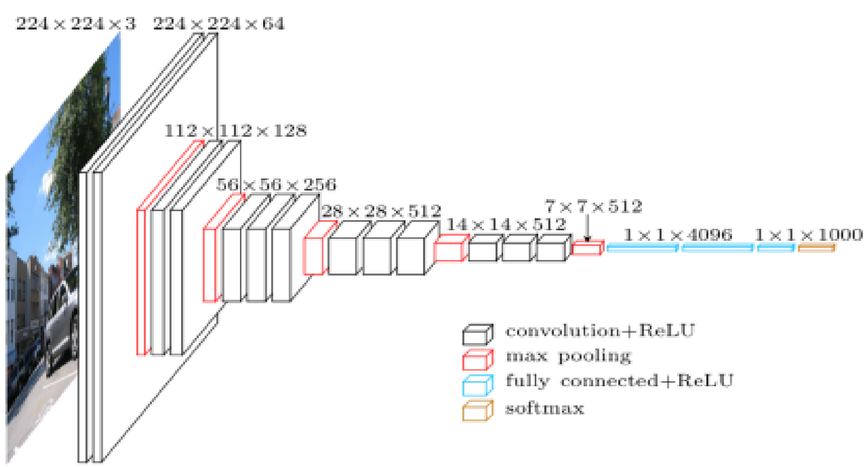
- **Multi ConvNet model :** (D/[256;512]/256,384,512), (E/[256;512]/256,384,512), multi-crop & dense eval: **23.7%** Top-1 Error, **6.8%** Top-5 Error.

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



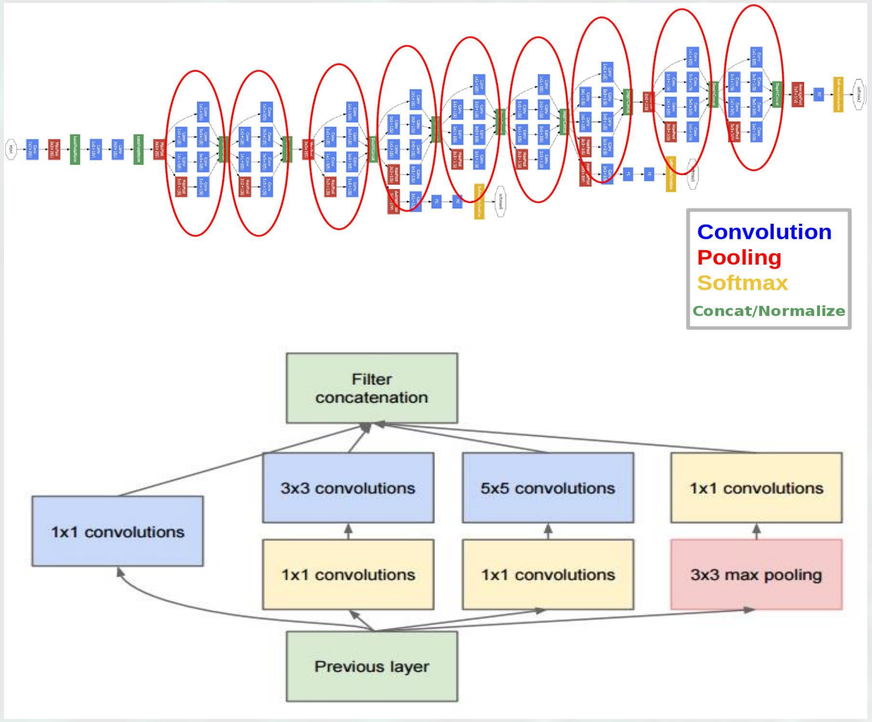

```
class VGG_16:

    @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)))
```

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




```
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:

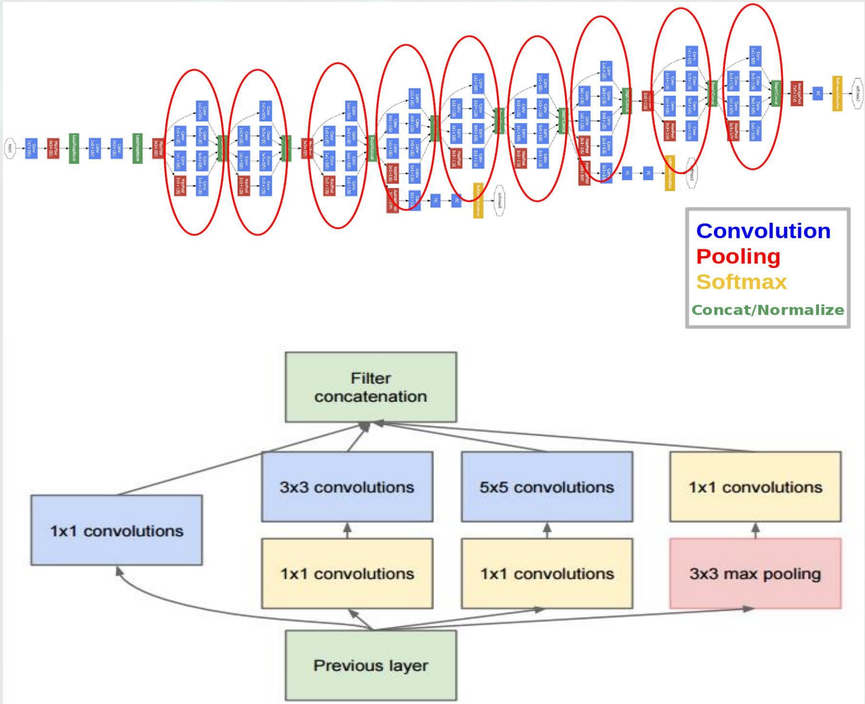
- **8 trainable layers:** 5 convolutional layers and 3 fully connected layers.
- **Max pooling layers** after 1st, 2nd and 5th layer.
- **Rectified Linear Units (ReLUs) (**Nair, V., & Hinton, G. E. 2010).
- **Local Response Normalization.**
- **60 millions parameters, 650 thousands neurons**.
- **Regularizations**: 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.

24

Results:

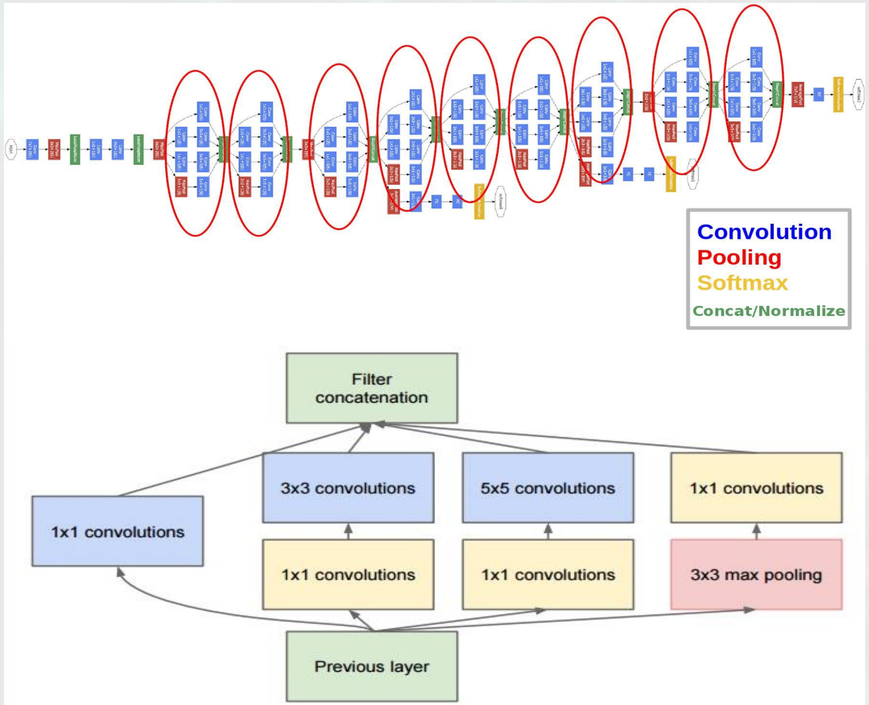
- **Multi ConvNet model :** (D/[256;512]/256,384,512), (E/[256;512]/256,384,512), multi-crop & dense eval: **23.7%** Top-1 Error, **6.8%** Top-5 Error.

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



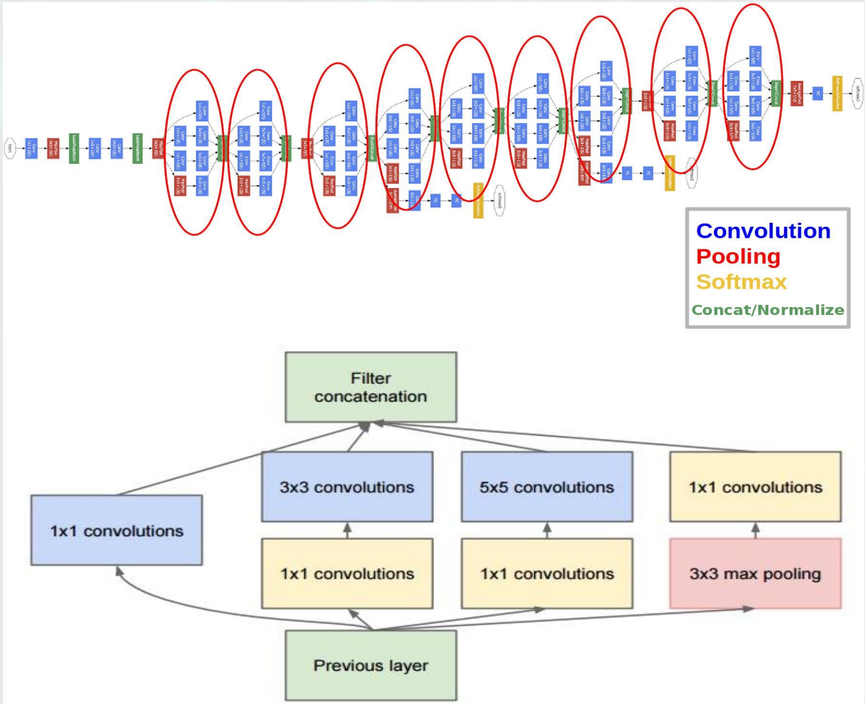

```python
class GoogleNet:

    @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)



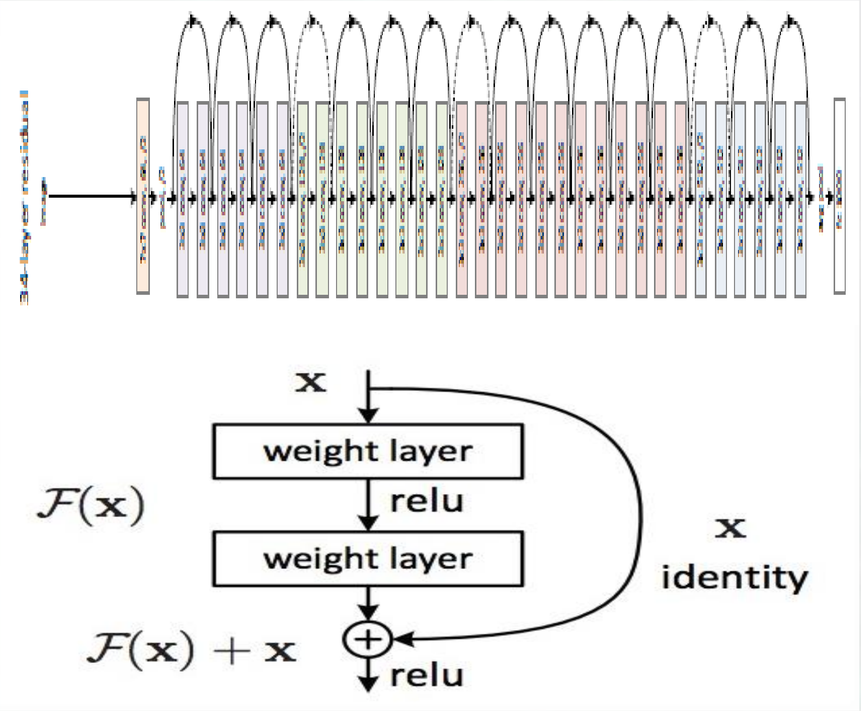

```
# Second Inception Module
inception_3b_1x1 = Convolution2D(128,1,1,border_mode='same',activation='relu',name='inception_3b/1x1',W_regularizer=l2(0.0002))(inception_3a_output)
inception_3b_3x3_reduce = Convolution2D(128,1,1,border_mode='same',activation='relu',name='inception_3b/3x3_reduce',W_regularizer=l2(0.0002))(inception_3a_output)
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

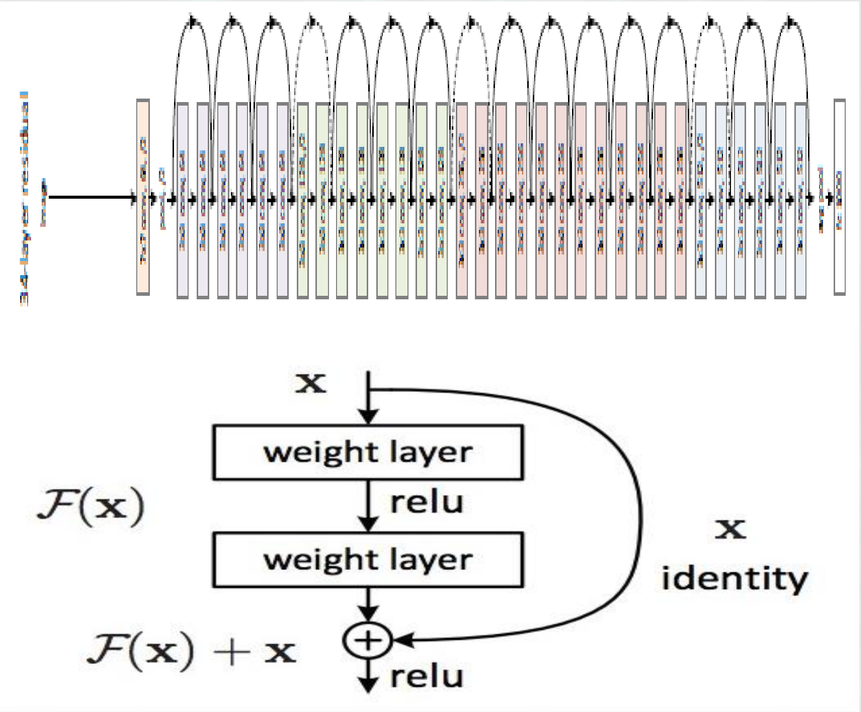
- **Degradation Problem:** Stacking more and more layers **IS NOT** better. With the network depth increasing, accuracy gets saturated and then degrades rapidly! It's an issue of "solvers".
- **Solves the "Degradation problem":** by fitting a residual mapping which is easier to optimize.
- **Shortcut 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.**

28

Results:

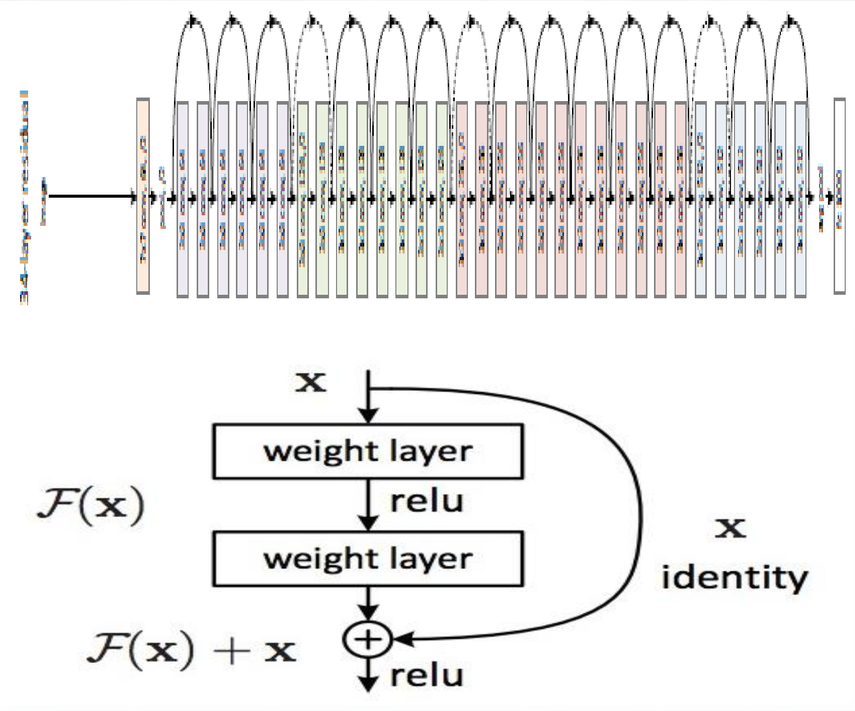
- **ResNet : 3.57%** Top-5 Error.
- **CNNs** show superhuman abilities at Image Recognition! **5%** Human estimated Top-5 error (Johnson, R. C., 2015).

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



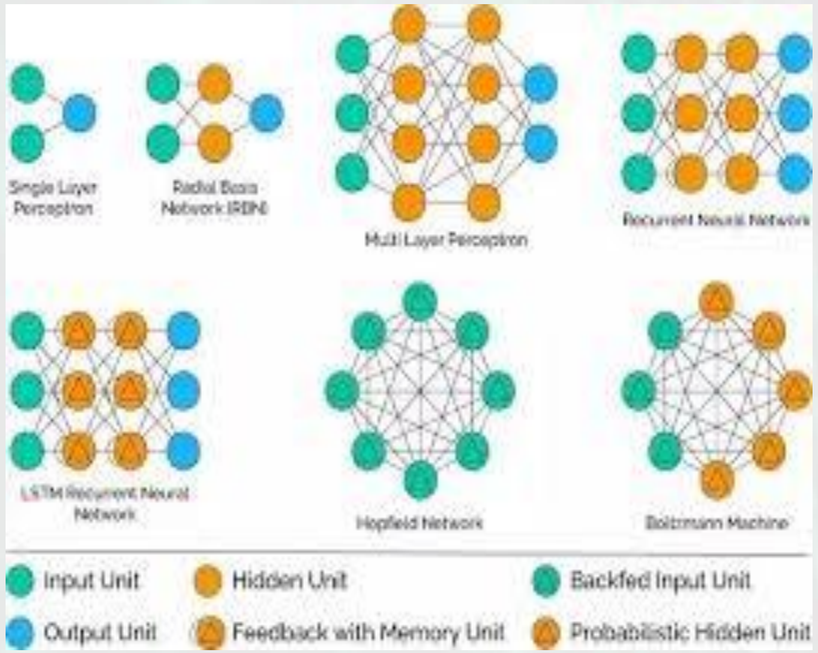

```
class WideResNet:

    @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
```

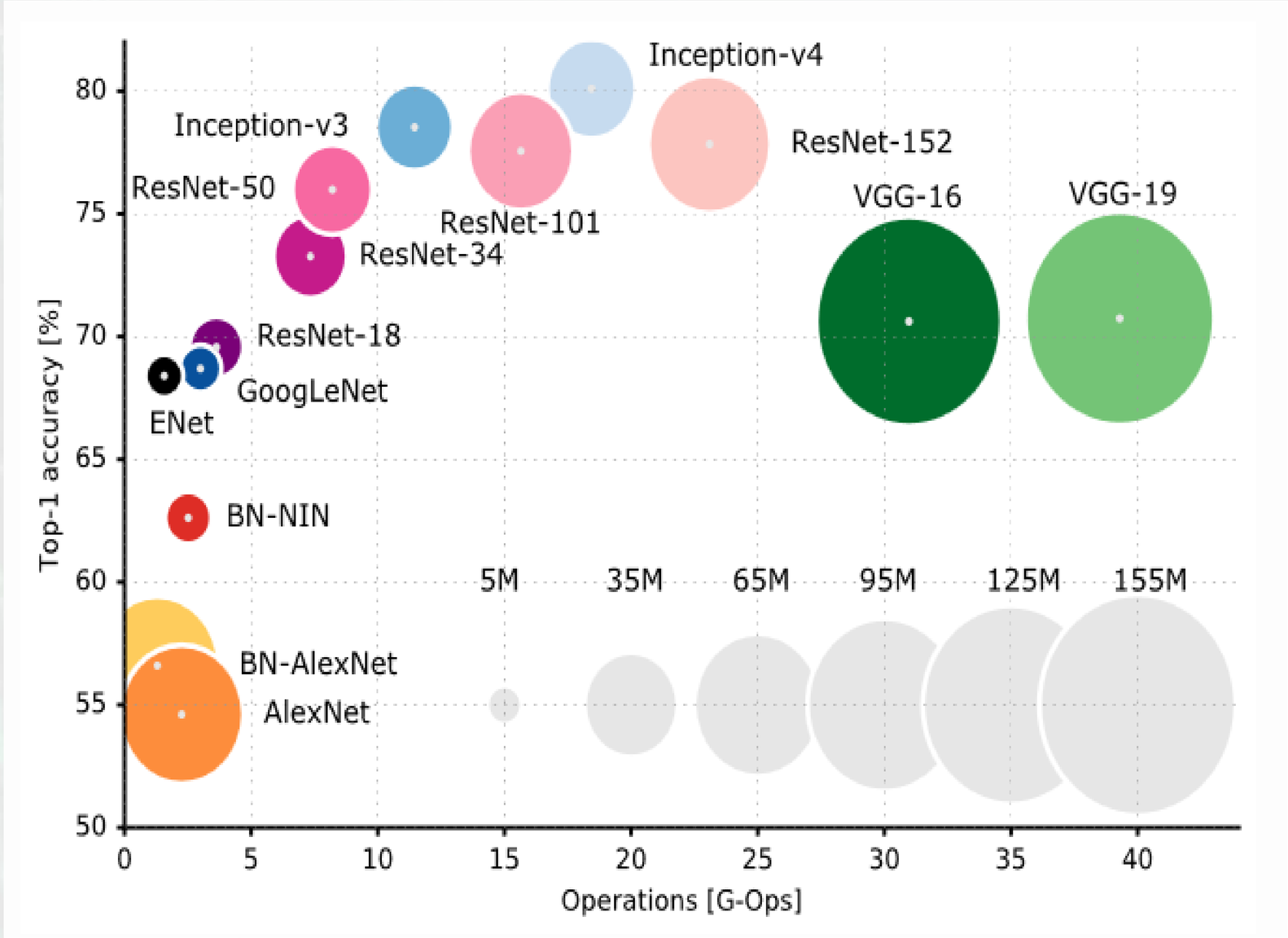
Residual Networks – Microsoft (He, K., et al., 2016)




```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 = BatchNormalisation(axis=1)(incoming)
        y = Activation('relu')(y)
        y = ZeroPadding2D((1, 1))(y)
        y = Conv2D(nb_bottleneck_filter, (3, 3), strides=subsample, kernel_initialiser='he_normal', padding='valid')(y)

        # conv3x3
        y = BatchNormalisation(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_initialiser='he_normal', padding='valid')(y)

        return add([incoming, y])

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

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

        # conv3x3
        y = BatchNormalisation(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_initialiser='he_normal', padding='valid')(y)

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

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

Neural Network Architectures





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.

DEEP LEARNING LESSONS

Deep Learning for Computer Vision (CV)

Francesco Pugliese, PhD
Data Scientist at ISTAT
francesco.pugliese@istat.it

Thank You