Lecture 4: Computer Vision (part 1)

Network Architectures and Transfer Learning

TIES4911 Deep-Learning for Cognitive Computing for Developers
Spring 2024

by:
Dr. Oleksiy Khriyenko
IT Faculty
University of Jyväskylä

Acknowledgement

I am grateful to all the creators/owners of the images that I found from Google and have used in this presentation.

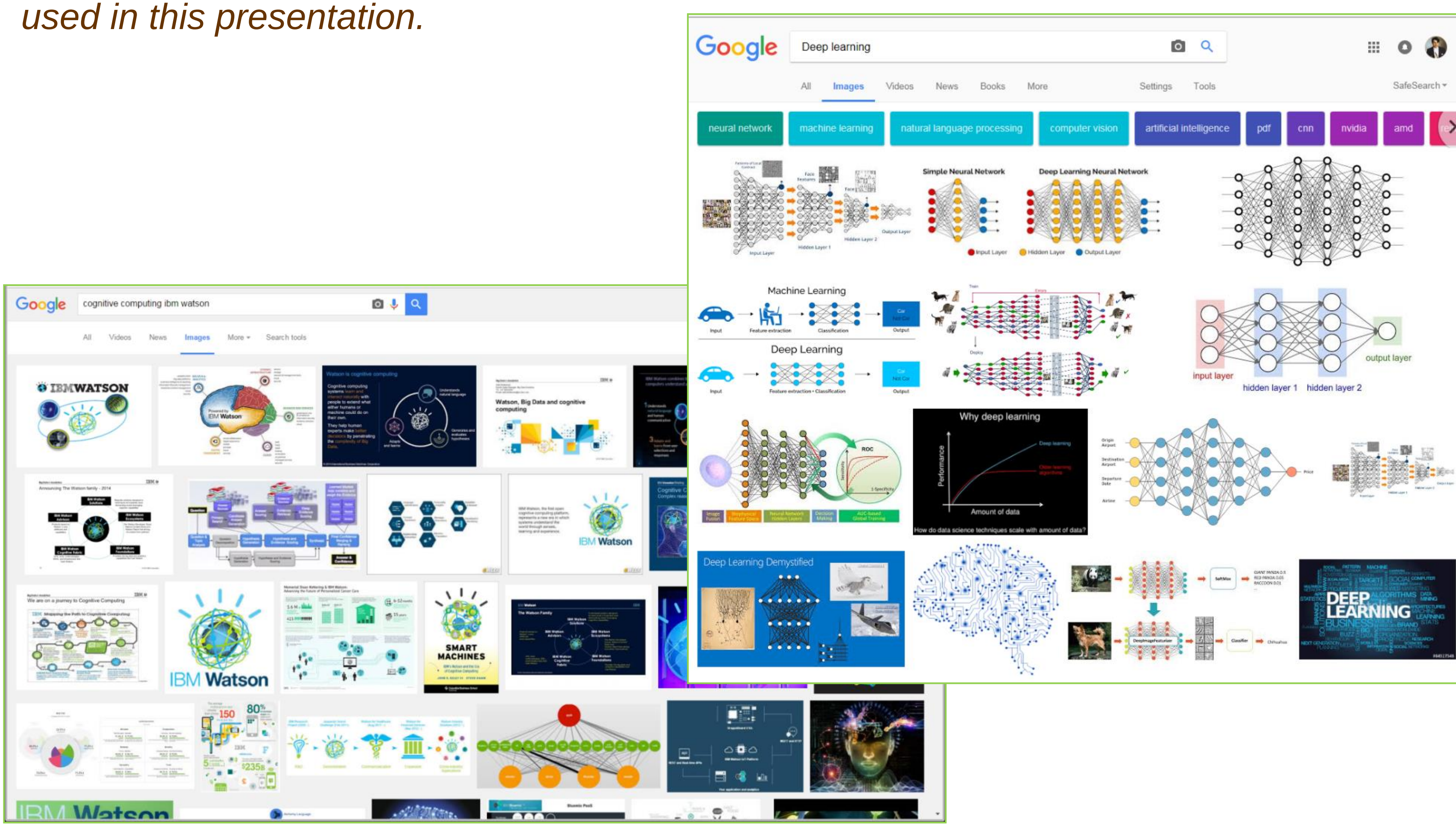


Image Recognition

Deep convolutional neural network can achieve reasonable performance on hard visual recognition tasks, matching or exceeding human performance in some domains.

Imagenet is a project started by Stanford professor ***Fei Fei Li***. It is a large visual database designed for use in visual object recognition software research that contains more than ***14 M*** images from more than ***21K*** different categories. Database organized according to the WordNet hierarchy (currently only the nouns), in which each node of the hierarchy is depicted by hundreds and thousands of images (an average of over five hundred images per node).

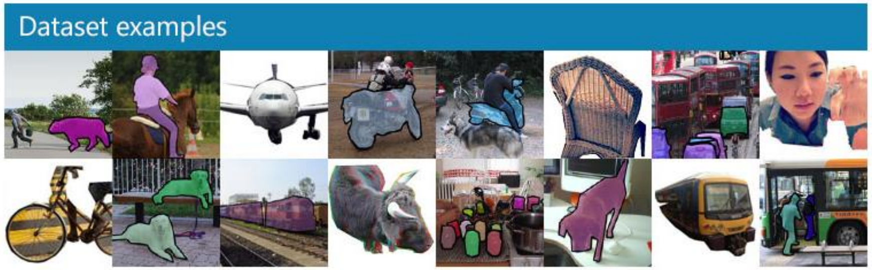
Since 2010, Imagenet runs ***ImageNet Large Scale Visual Recognition Challenge (ILSVRC)*** (*http://image-net.org/*) - an annual competition in visual recognition where participants are provided with 1.2 million images belonging to 1000 different classes from Imagenet data-set. Competition no longer hold after 2017.

Detection Competitions:

- ***Pascal VOC*** (http://host.robots.ox.ac.uk/pascal/VOC/) *project is finished in 2012*
- ***COCO*** (http://cocodataset.org/#home)
- ***ImageNet ILSVRC*** (http://image-net.org)(2010-2017)
- ***Kaggle*** (https://www.kaggle.com/competitions)

VOC: 20 classes

COCO: 200 classes



All the projects manage large-scale object detection, segmentation, and captioning datasets.

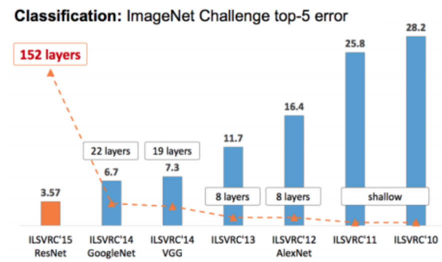
Relevant links:
https://karpathy.github.io/2014/09/02/what-i-learned-from-competing-against-a-convnet-on-imagenet/

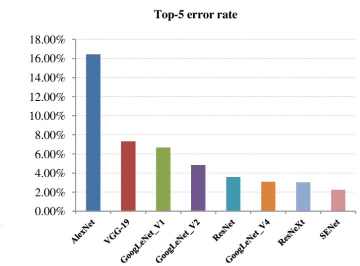
Image Recognition

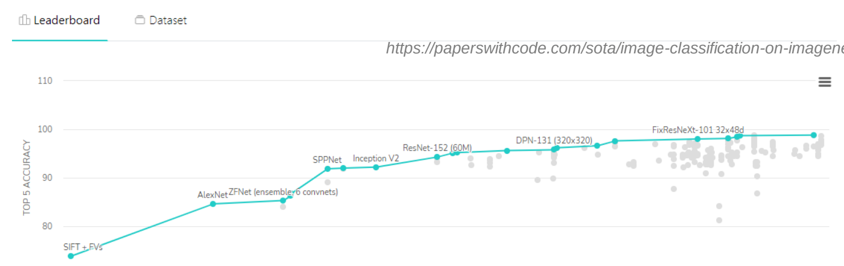
Successive models constantly continue to show improvements

(a top-5 error rate):

- *AlexNet (15.3%, 2012) by Alex Krizhevsky*
- *VGG (7.7%, 2014) by a reasearch group at Oxford*
- *Inception (GoogLeNet) (6.67%, 2014) by Google*
- *Inception-v2 (4.9%)*
- *ResNet (3.57%, 2015) by Microsoft*
- *Inception-v3 (3.57%, 2015)*
- ***Inception-v4(+Residual) (3.08%)***
- *SqueezeNet (~15%) is remarkable for how less computation does it need (pre-trained model on Imagenet has a size of less than 5MB)*
- ***ResNeXt (3.03%)***
- ***SENet (2.25%)** 2017*
- *Andrej Karpathy (5.1%) – attempted to compete against a ConvNet*







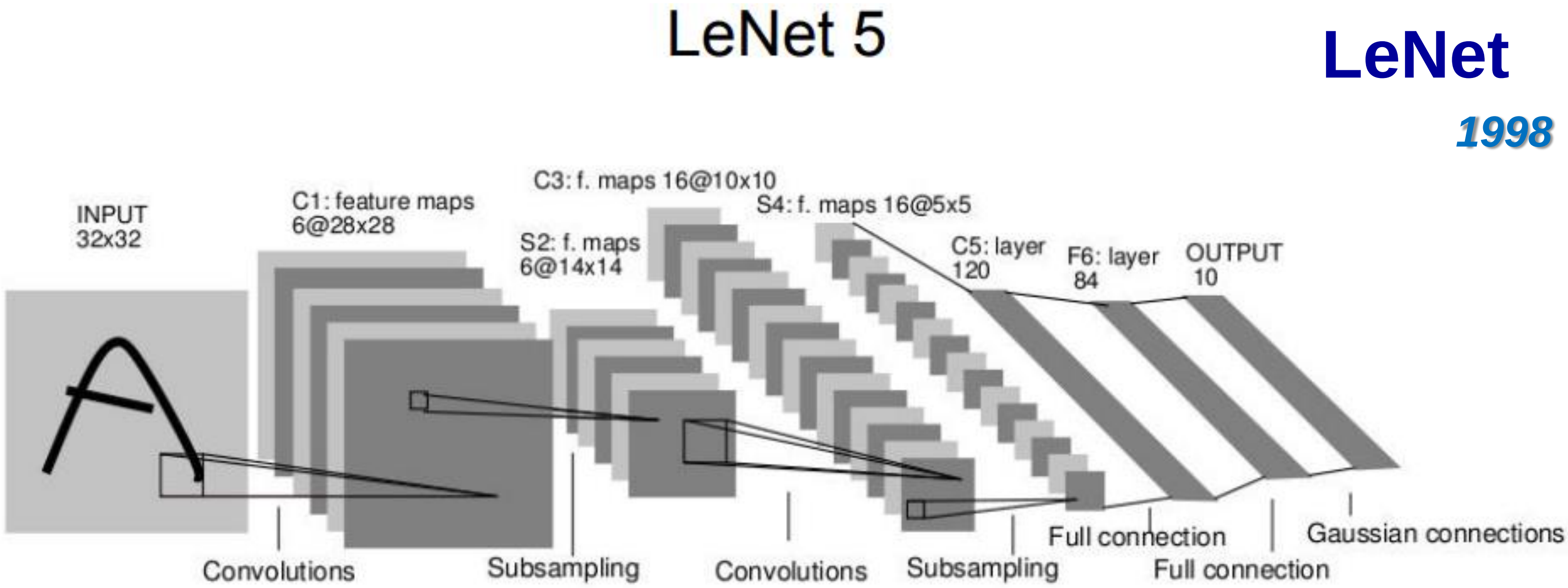
https://paperswithcode.com/sota/image-classification-on-imagenet

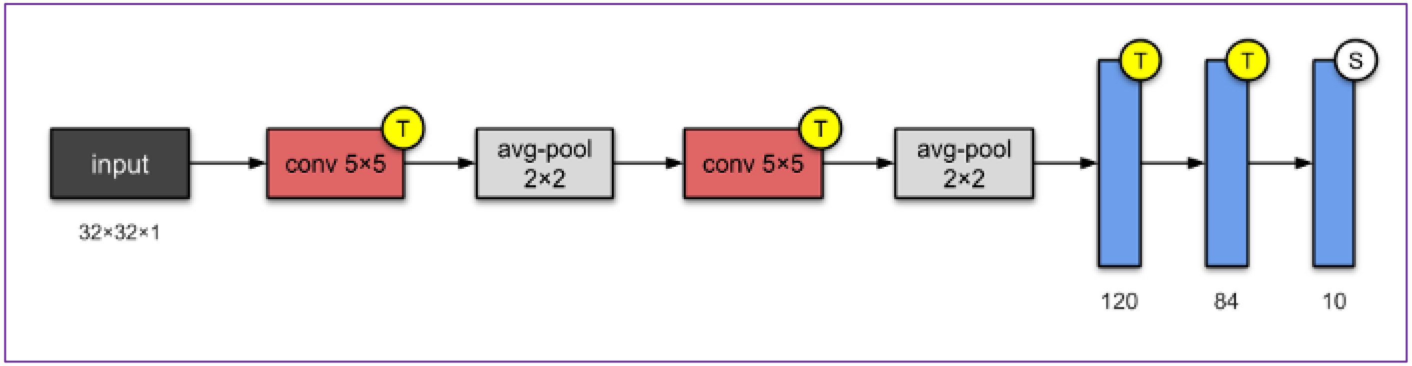
Relevant links:

https://medium.com/@RaghavPrabhu/cnn-architectures-lenet-alexnet-vgg-googlenet-and-resnet-7c81c017b848
https://medium.com/@sidereal/cnns-architectures-lenet-alexnet-vgg-googlenet-resnet-and-more-666091488df5
https://towardsdatascience.com/illustrated-10-cnn-architectures-95d78ace614d
https://journalofbigdata.springeropen.com/articles/10.1186/s40537-021-00444-8
https://karpathy.github.io/2014/09/02/what-i-learned-from-competing-against-a-convnet-on-imagenet/
http://slazebni.cs.illinois.edu/spring17/lec01_cnn_architectures.pdf
http://cv-tricks.com/cnn/understand-resnet-alexnet-vgg-inception/

LeNet

1998

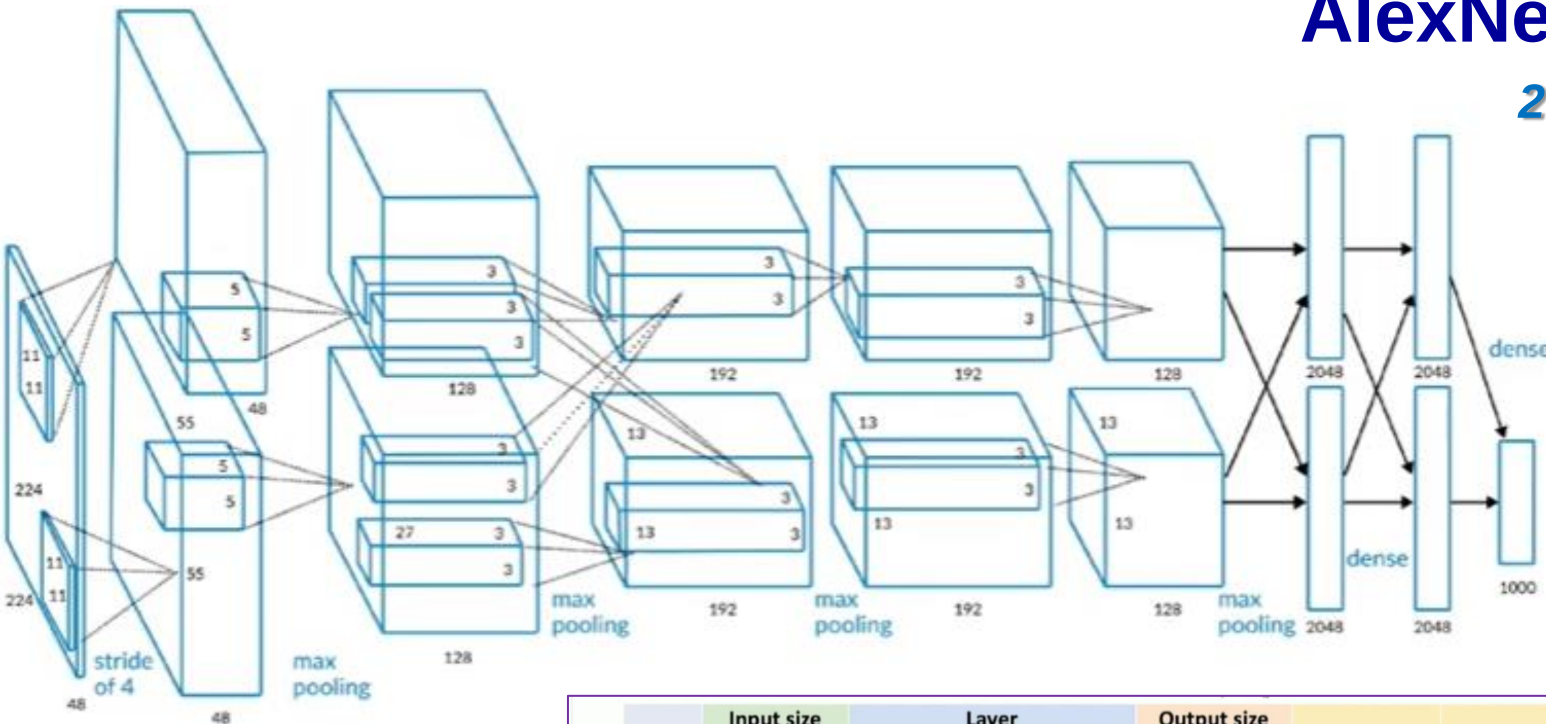


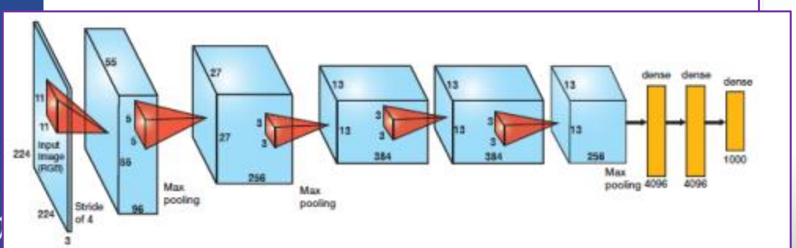


Relevant links:
https://ieeexplore.ieee.org/abstract/document/726791
https://machinelearningmastery.com/review-of-architectural-innovations-for-convolutional-neural-networks-for-image-classification/

AlexNet

2012



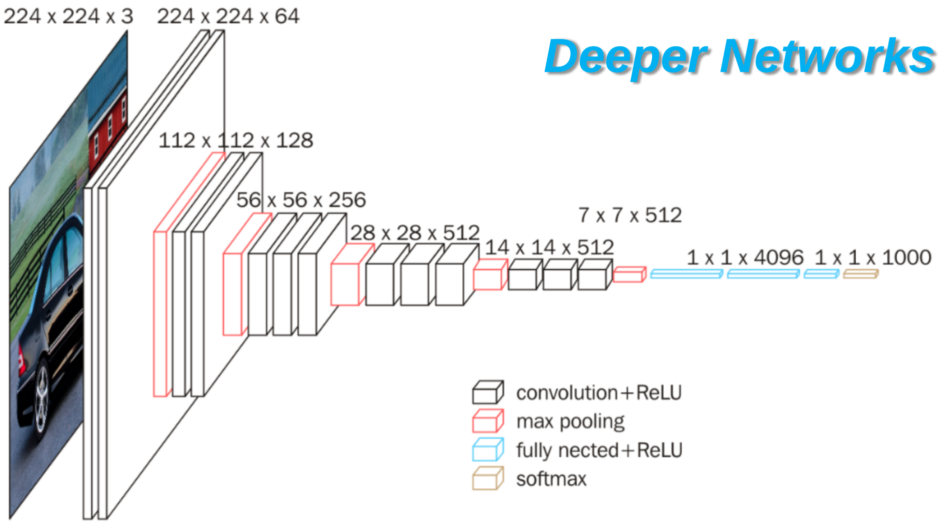


	Input size		Layer				Output size				
Layer	C	H / W	filters	kernel	stride	pad	C	H / W	memory (KB)	params (k)	flop (M)
conv1	3	227	64	11	4	2	64	56	784	23	73
pool1	64	56		3	2	0	64	27	182	0	0
conv2	64	27	192	5	1	2	192	27	547	307	224
pool2	192	27		3	2	0	192	13	127	0	0
conv3	192	13	384	3	1	1	384	13	254	664	112
conv4	384	13	256	3	1	1	256	13	169	885	145
conv5	256	13	256	3	1	1	256	13	169	590	100
pool5	256	13		3	2	0	256	6	36	0	0
flatten	256	6					9216		36	0	0
fc6	9216		4096				4096		16	37,749	38
fc7	4096		4096				4096		16	16,777	17
fc8	4096		1000				1000		4	4,096	4

Relevant links:
https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf
https://machinelearningmastery.com/review-of-architectural-innovations-for-convolutional-neural-networks-for-image-classification/

VGG

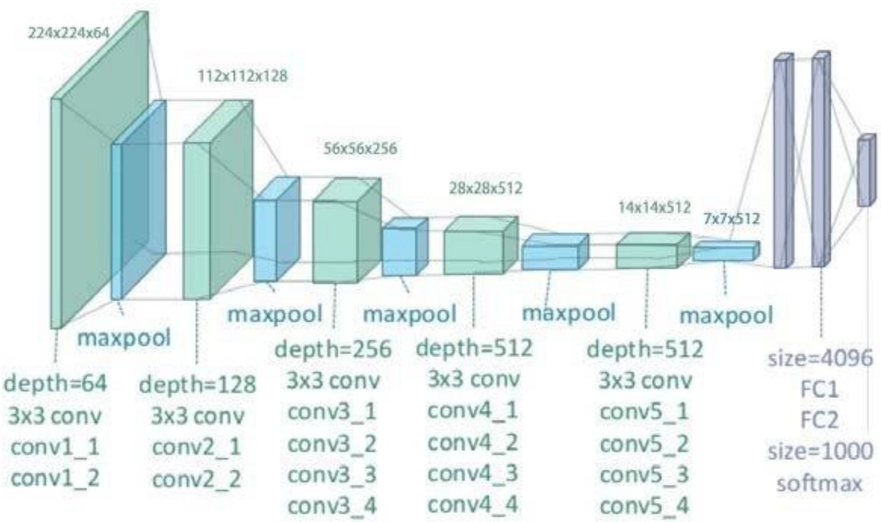
VGG 16



Deeper Networks with Regular Design…

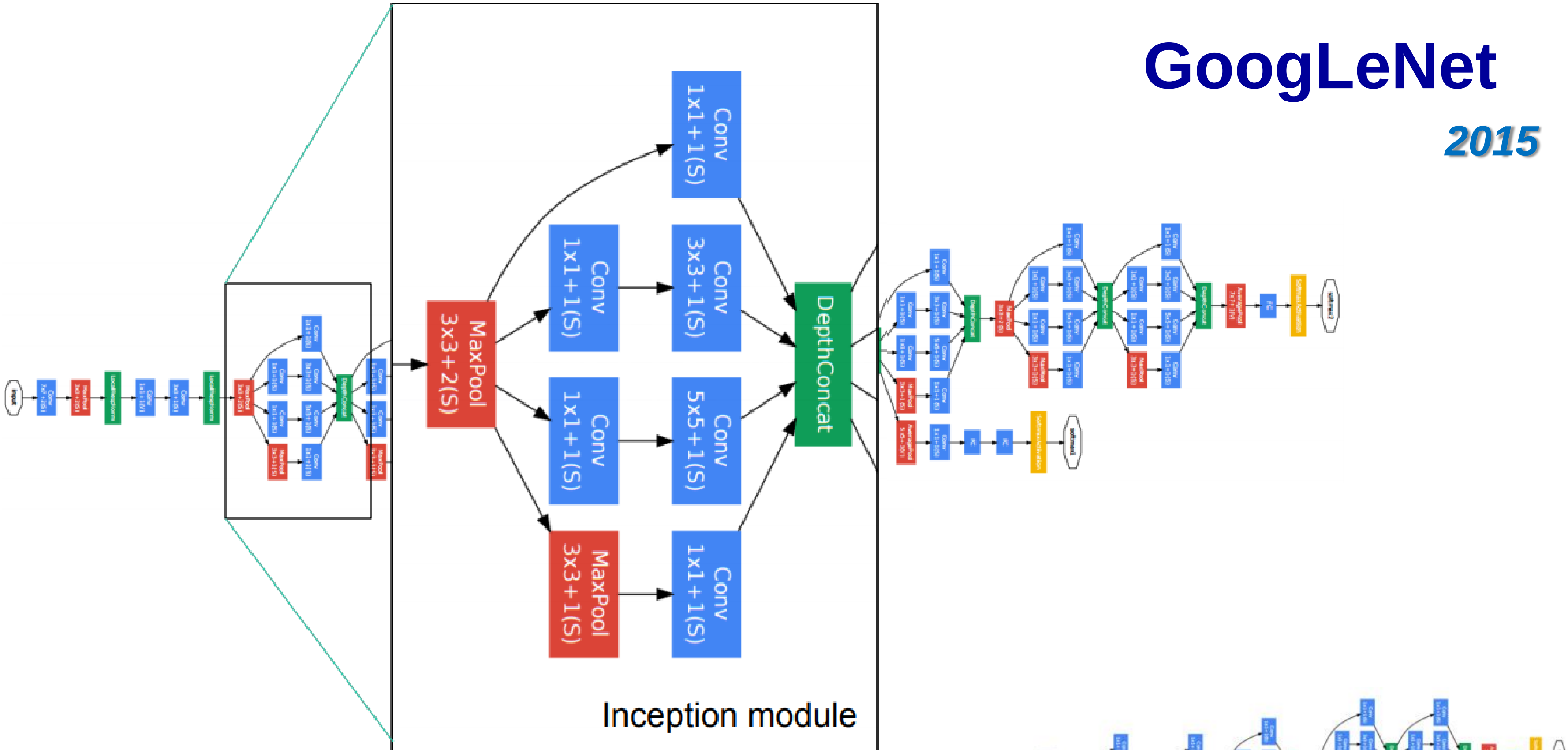
2015

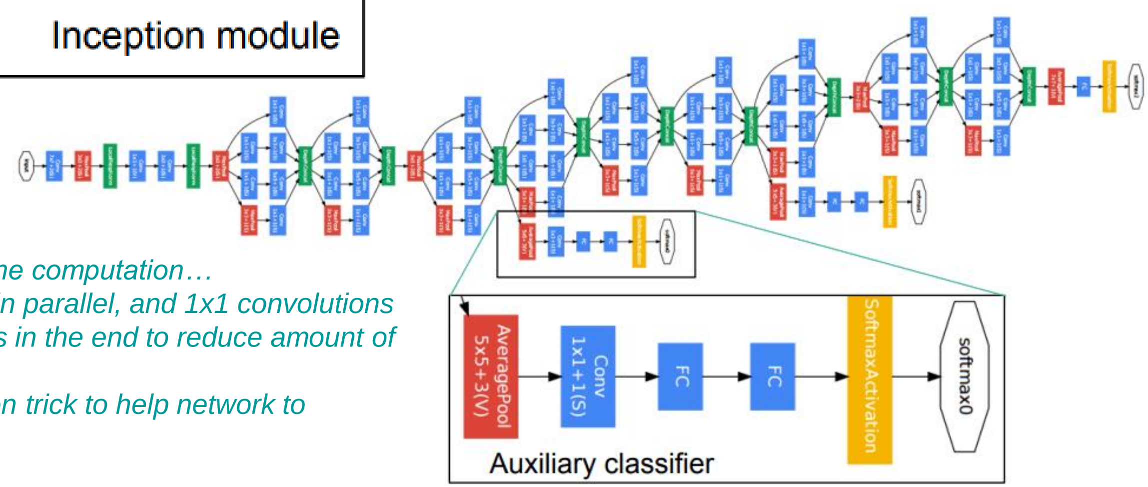
VGG 19



GoogLeNet

2015

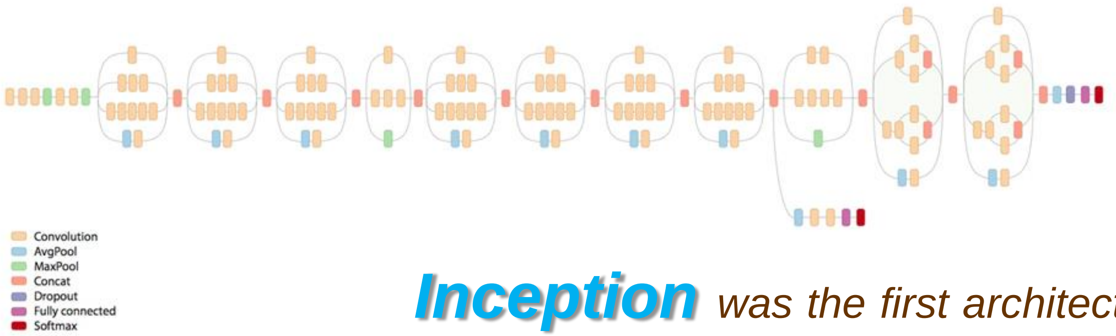




Efficiency…

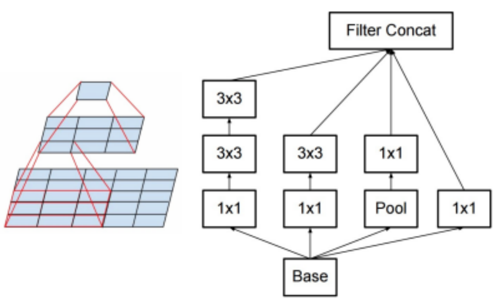
- *aggressively downsamples input to reduce the computation…*
- *Inception module with different kernel sized in parallel, and 1x1 convolutions*
- *Global Average Pooling instead of FC layers in the end to reduce amount of parameters*
- *Auxiliary classifiers, as a kind of normalization trick to help network to converge*

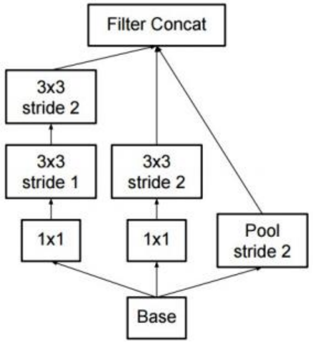
Inception v2, v3

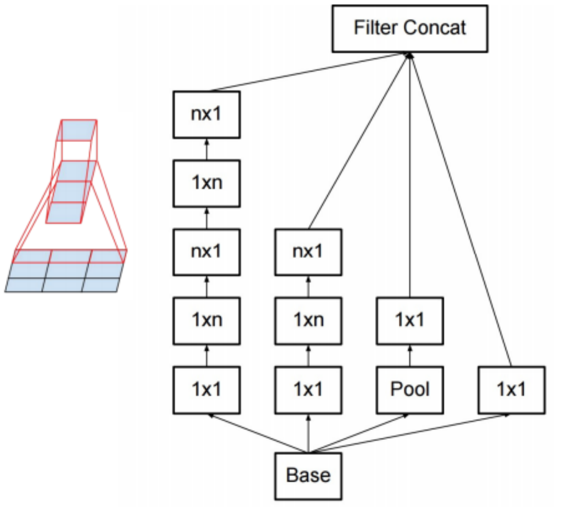


Another view of GoogLeNet's architecture.

Inception *was the first architecture which improved results by design, not by simply going deep!!!*







Relevant links:
https://www.analyticsvidhya.com/blog/2018/10/understanding-inception-network-from-scratch/?utm_source=blog&utm_medium=top4_pre-trained_image_classification_models

ResNet

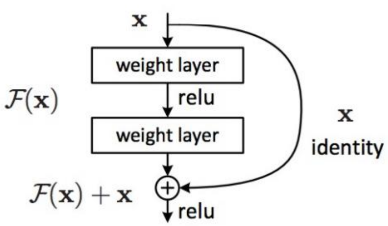
Revolution of depth by **ResNet** **2016**

Thanks to the discovered Batch Normalization, we may go much deeper!!!
But, on practice, more shallow networks show better results than much deeper networks…

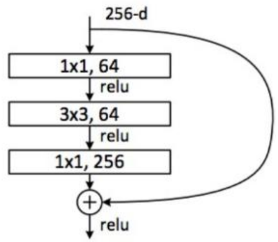
*From 8 layers (AlexNet, 2012), 19 layers (VGG, 2014) and 22 layers (GoogLeNet, 2014) to **152 layers** in year 2015.*

- *Introduces Residual Module (skip or shortcut connection) helps to learn identity function within deeper networks… Basic block is for ResNet18 and ResNet34; and bottleneck block for ResNet50, ResNet101 and ResNet152.*

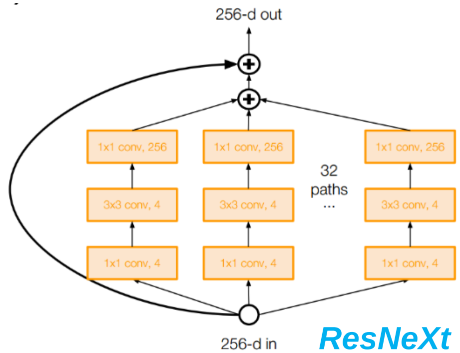
"Basic"



"Bottleneck"

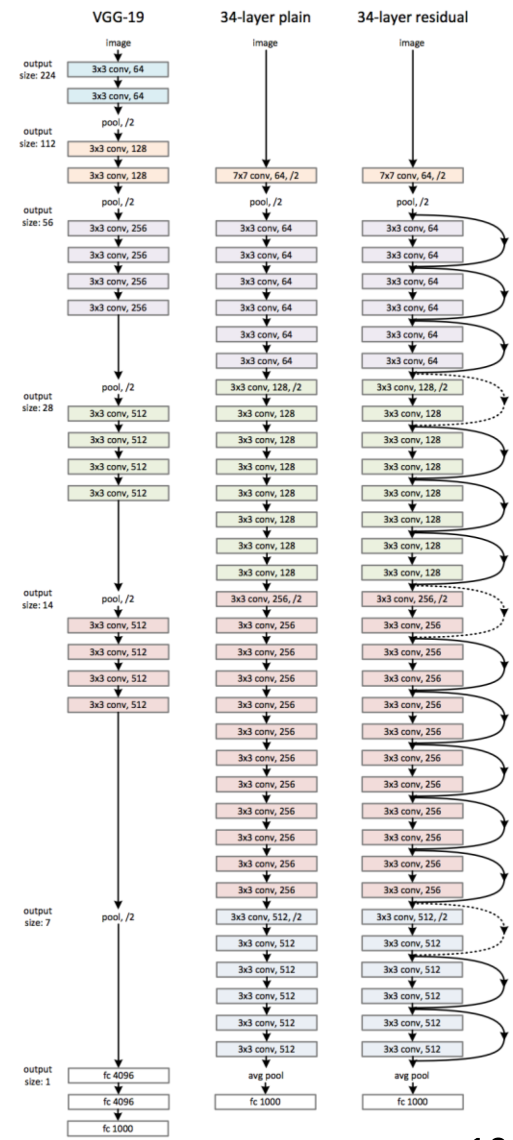


" multiple parallel pathways"



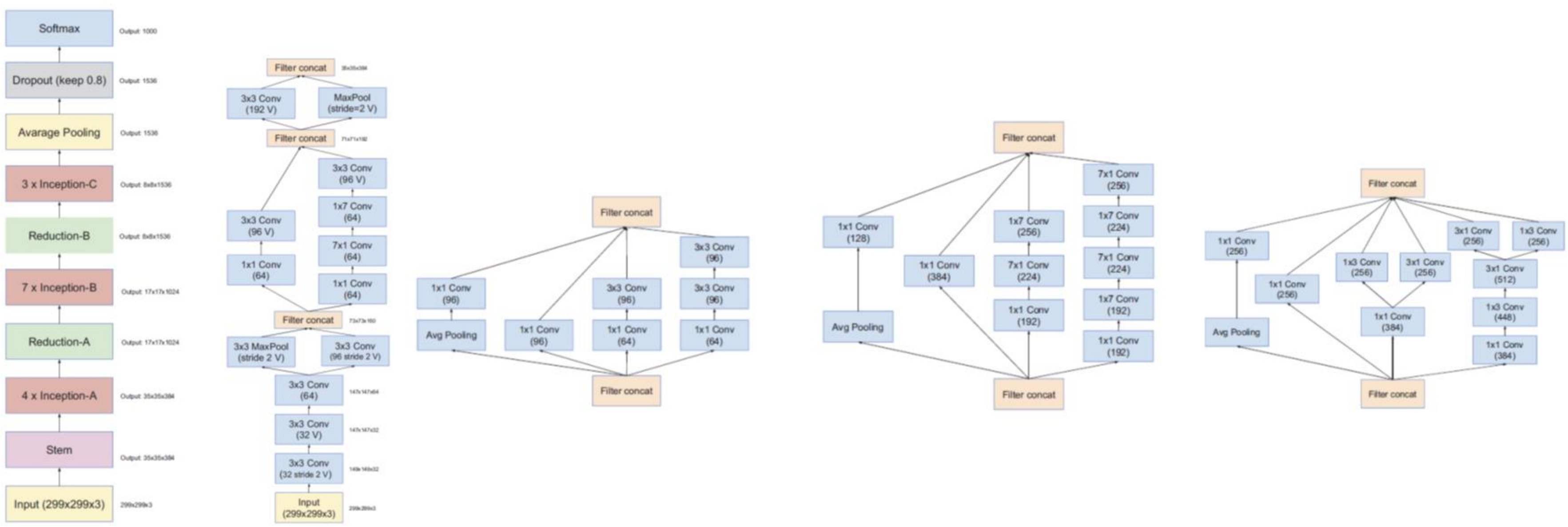
ResNeXt

- *Also uses the aggressive stem in the beginning, and global average pooling in the end as GoogLeNet*



Inception v4

A more uniform simplified architecture and more inception modules than ***Inception-v3***

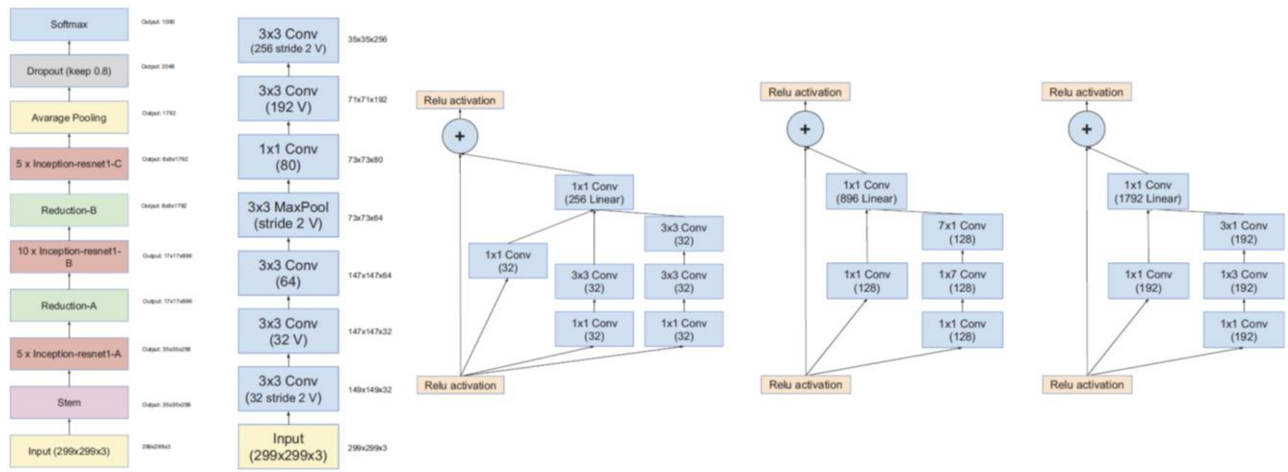


Relevant links:
https://towardsdatascience.com/review-inception-v4-evolved-from-googlenet-merged-with-resnet-idea-image-classification-5e8c339d18bc

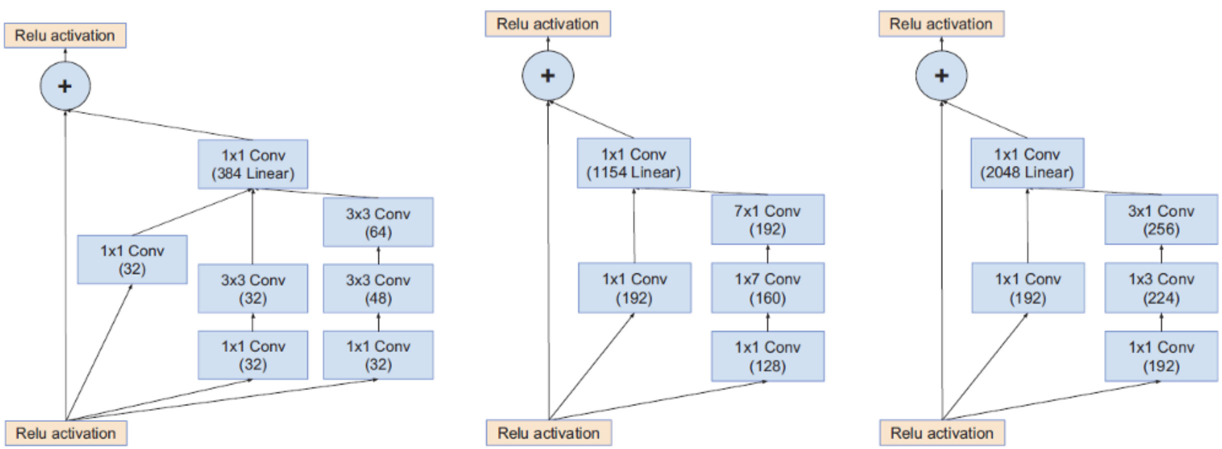
Inception - ResNet

Combine *Inception* and *Residual Modules*

Inception-ResNet-v1



Inception-ResNet-v2

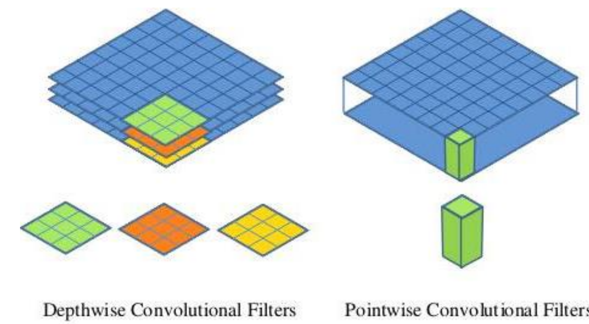


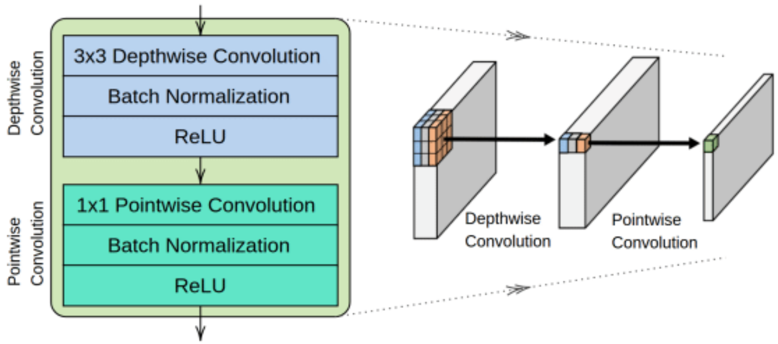
Relevant links:

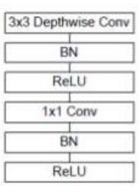
https://towardsdatascience.com/review-inception-v4-evolved-from-googlenet-merged-with-resnet-idea-image-classification-5e8c339d18bc

Tiny networks (MobileNet, ShuffleNet)

MobileNets with the *depthwise separable convolutions process*, which consists of *depthwise convolution* and *pointwise convolution*. The batch normalization layer and the rectified linear unit are added at the end of every convolutional layer.

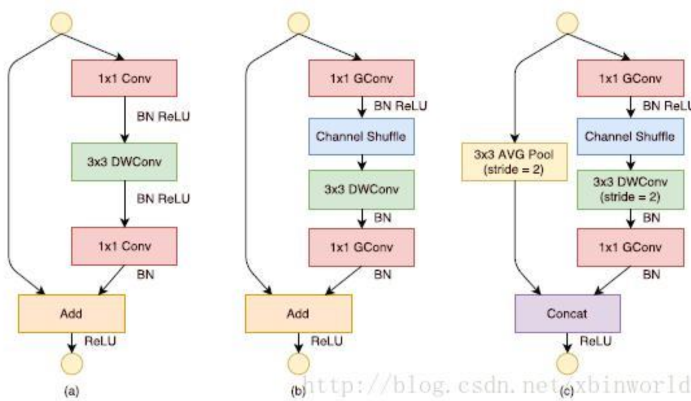




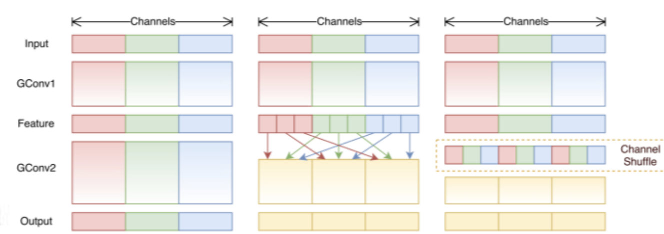


Depthwise Separable Convolution

ShuffleNet also uses the *depthwise convolution*, *grouped convolution and channel shuffle*.



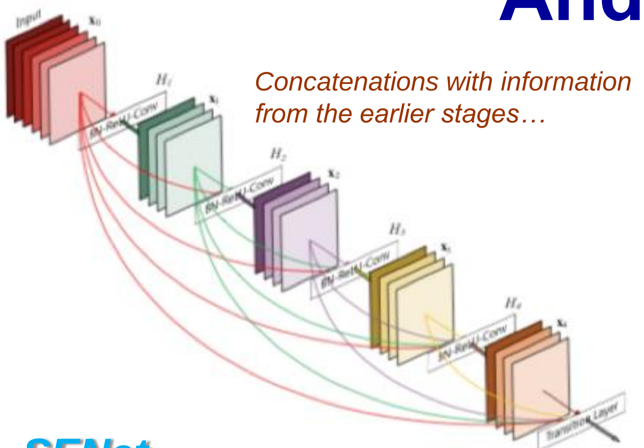
These networks have not that high accuracy, but are computationally efficient to be used on mobile and embedded devices…



Relevant links:
https://arxiv.org/abs/1704.04861
https://www.programmersought.com/article/7227832762/
https://towardsdatascience.com/mobilenetv2-inverted-residuals-and-linear-bottlenecks-8a4362f4ffd5

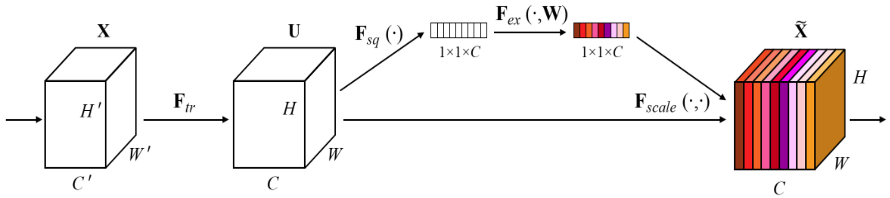
And many other in the future…

DenseNet



Concatenations with information from the earlier stages…

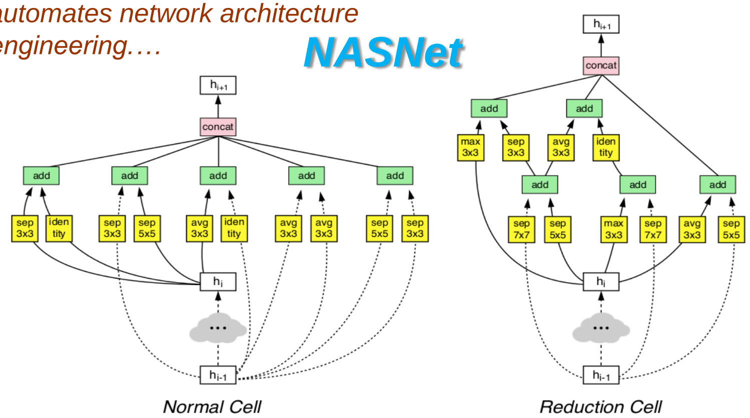
SENet



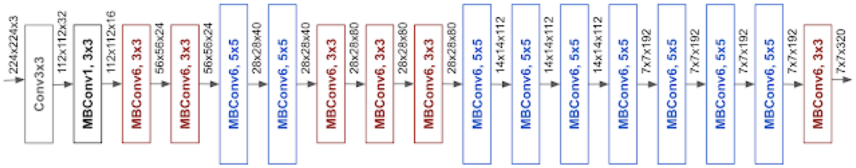
Learns relevance of feature maps depending on the content… Extra trainable module allows rescaling of channels depending on input.

Neural Search Architecture (NAS) - automates network architecture engineering.…

NASNet



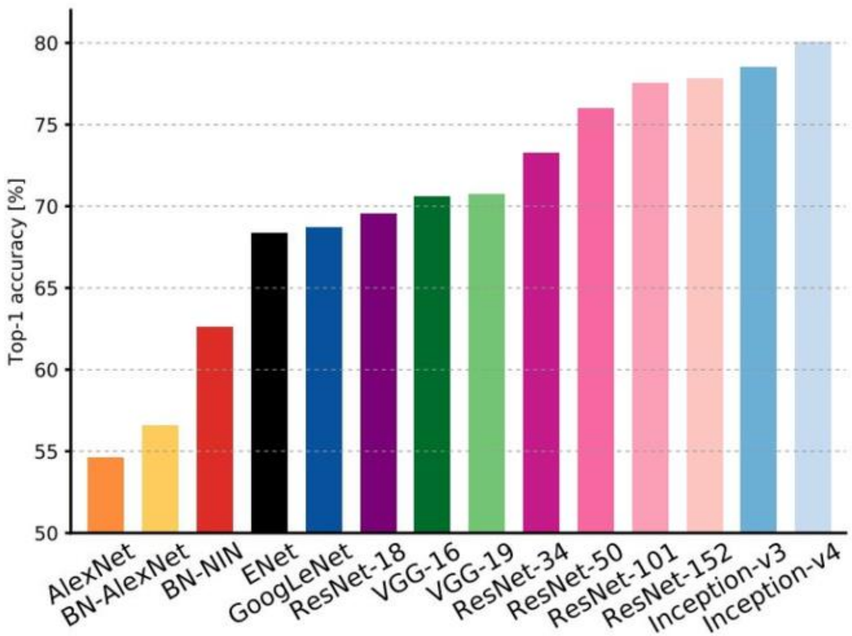
EfficientNet

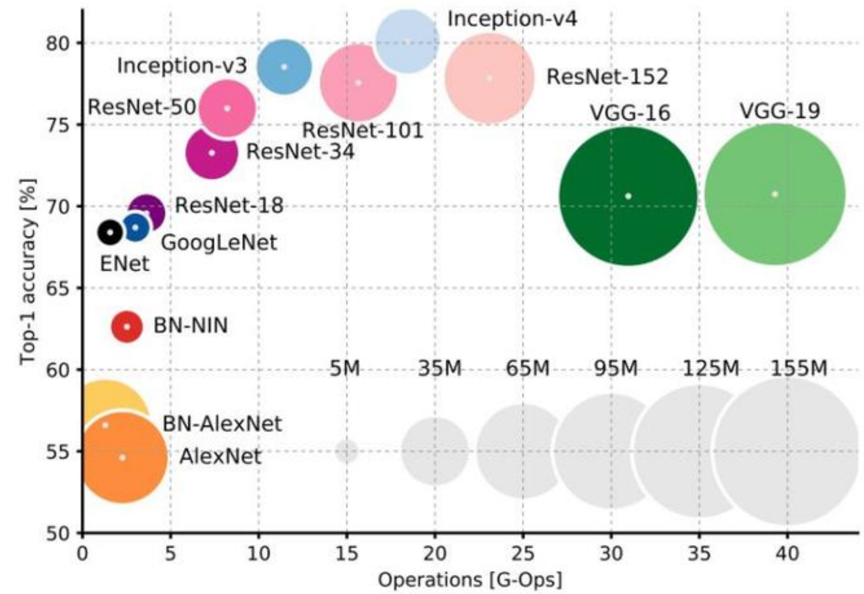


Applies Stochastic Depth and uses scaling method that uniformly scales all dimensions of depth/width/resolution using a compound coefficient.….

Relevant links:
https://arxiv.org/abs/1608.06993v5
https://towardsdatascience.com/understanding-and-visualizing-densenets-7f688092391a
https://arxiv.org/pdf/1707.07012.pdf
https://sh-tsang.medium.com/review-nasnet-neural-architecture-search-network-image-classification-23139ea0425d
https://towardsdatascience.com/review-senet-squeeze-and-excitation-network-winner-of-ilsvrc-2017-image-classification-a887b98b2883
https://arxiv.org/pdf/1709.01507.pdf
https://towardsdatascience.com/squeeze-and-excitation-networks-9ef5e71eacd7
https://arxiv.org/pdf/1905.11946.pdf

Network Architectures

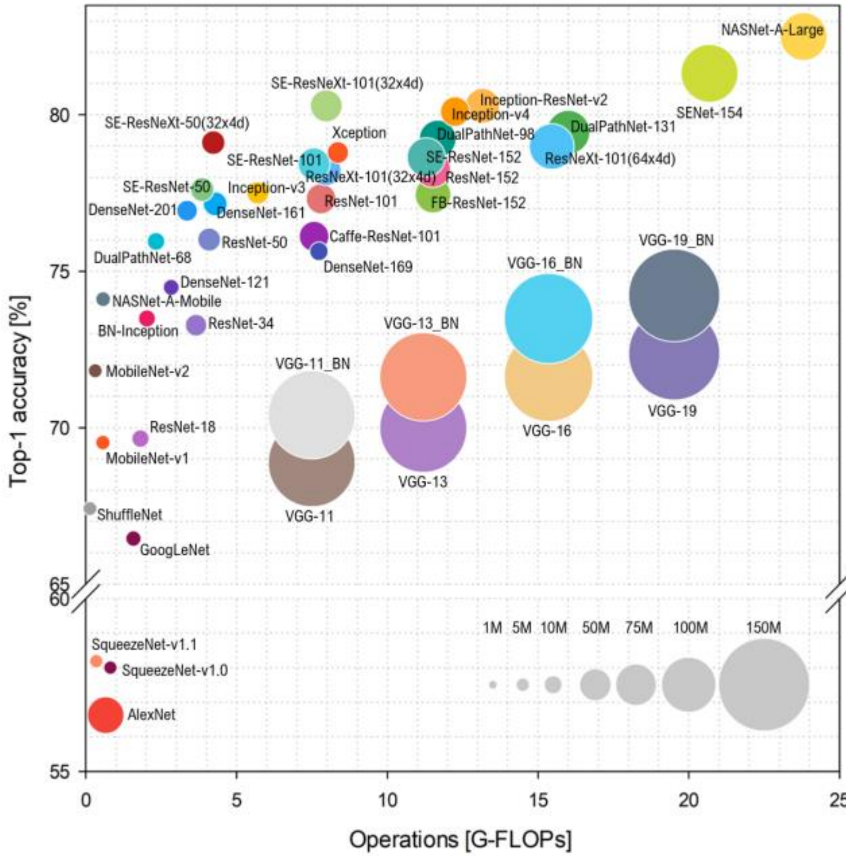


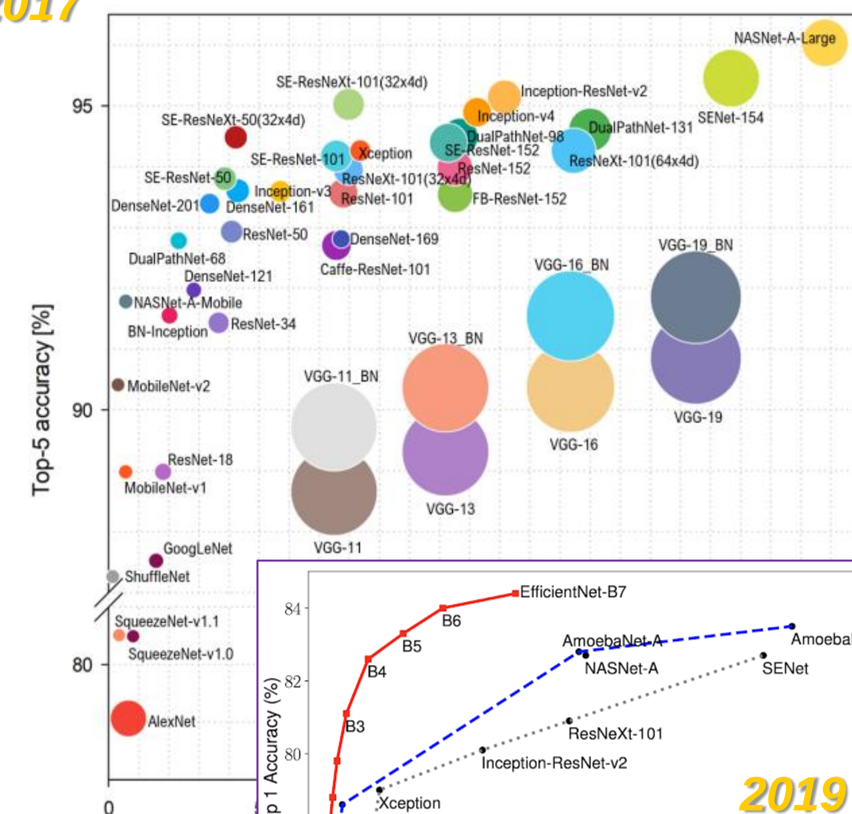


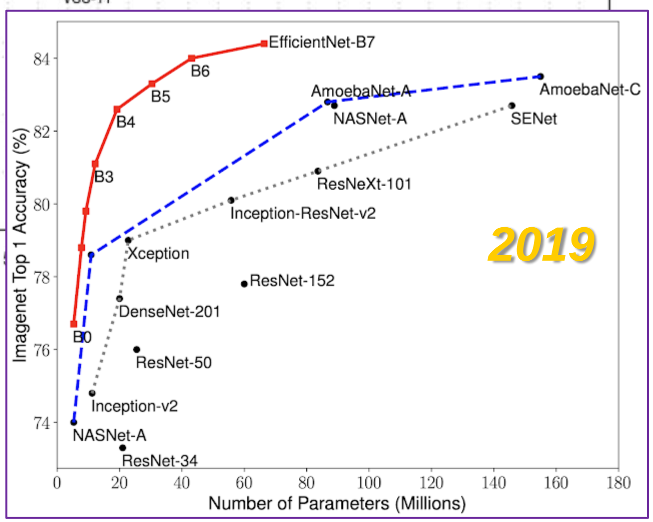
Relevant links:
https://tariq-hasan.github.io/concepts/computer-vision-cnn-architectures/

Network Architectures

2017

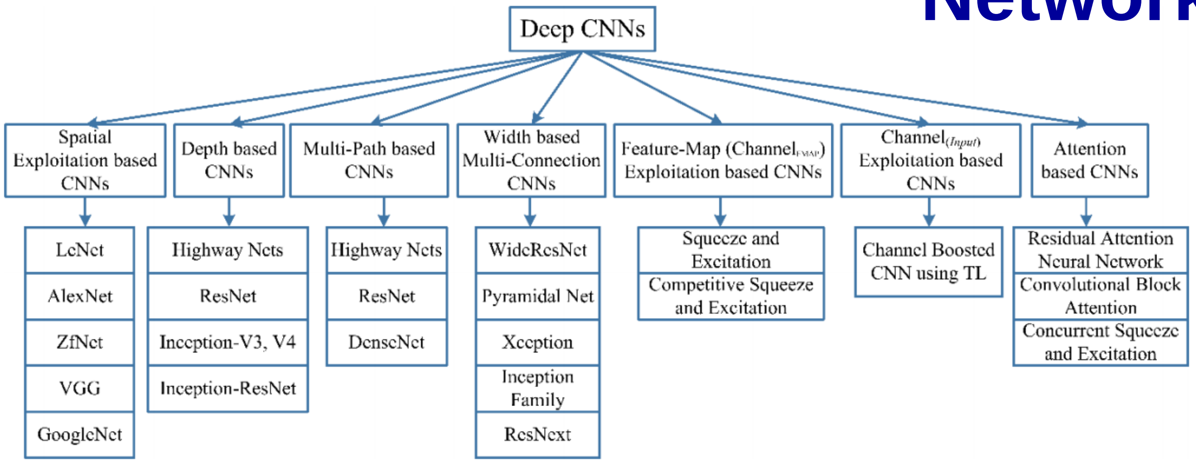




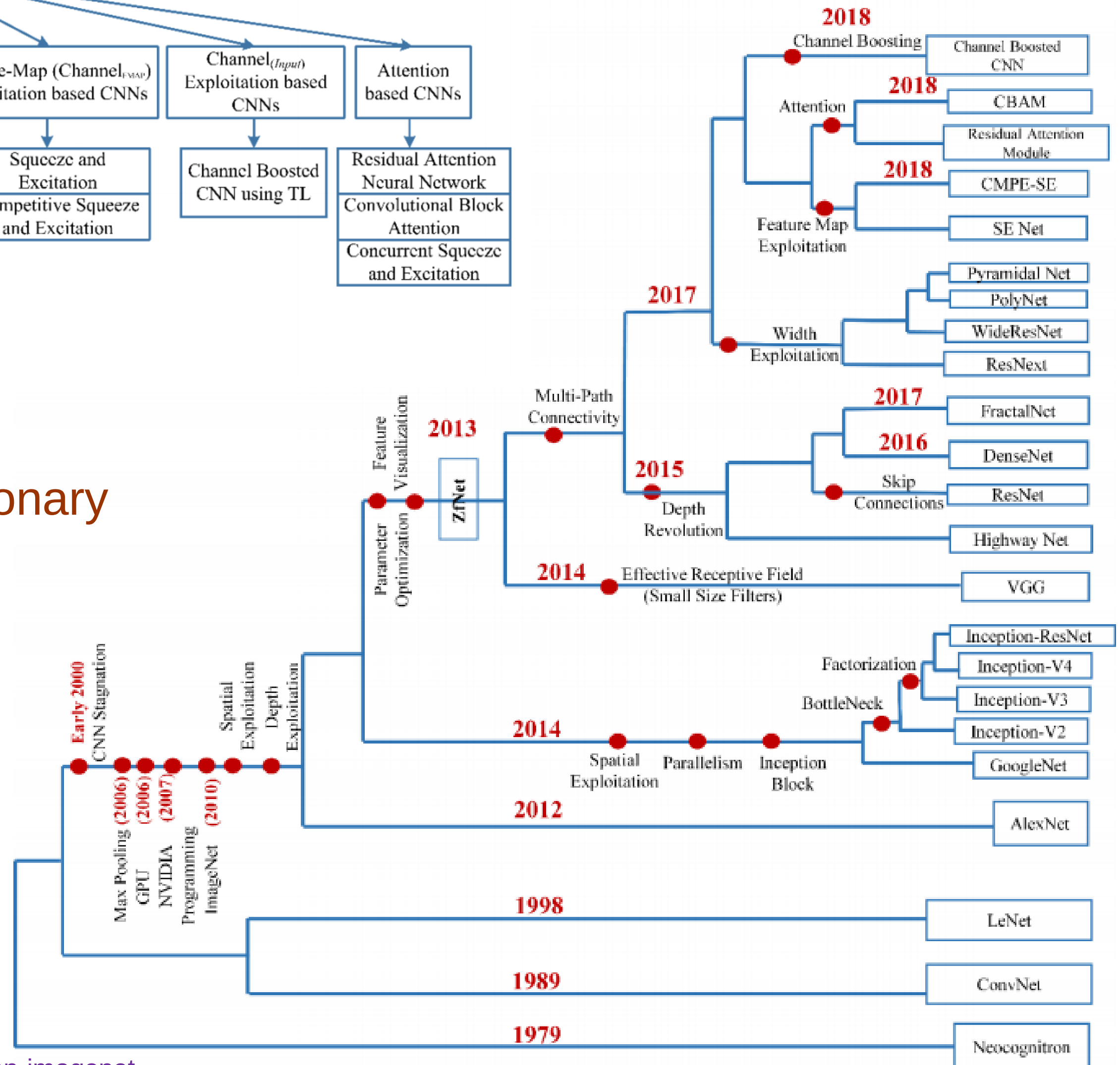


Relevant links:
https://paperswithcode.com/sota/image-classification-on-imagenet
https://arxiv.org/ftp/arxiv/papers/1901/1901.06032.pdf
https://arxiv.org/pdf/1810.00736.pdf
https://arxiv.org/pdf/1905.11946.pdf

Network Architectures



Taxonomy and evolutionary history of Deep CNNs



Relevant links:
https://arxiv.org/ftp/arxiv/papers/1901/1901.06032.pdf
https://arxiv.org/pdf/1810.00736.pdf
https://paperswithcode.com/sota/image-classification-on-imagenet

Network Architectures

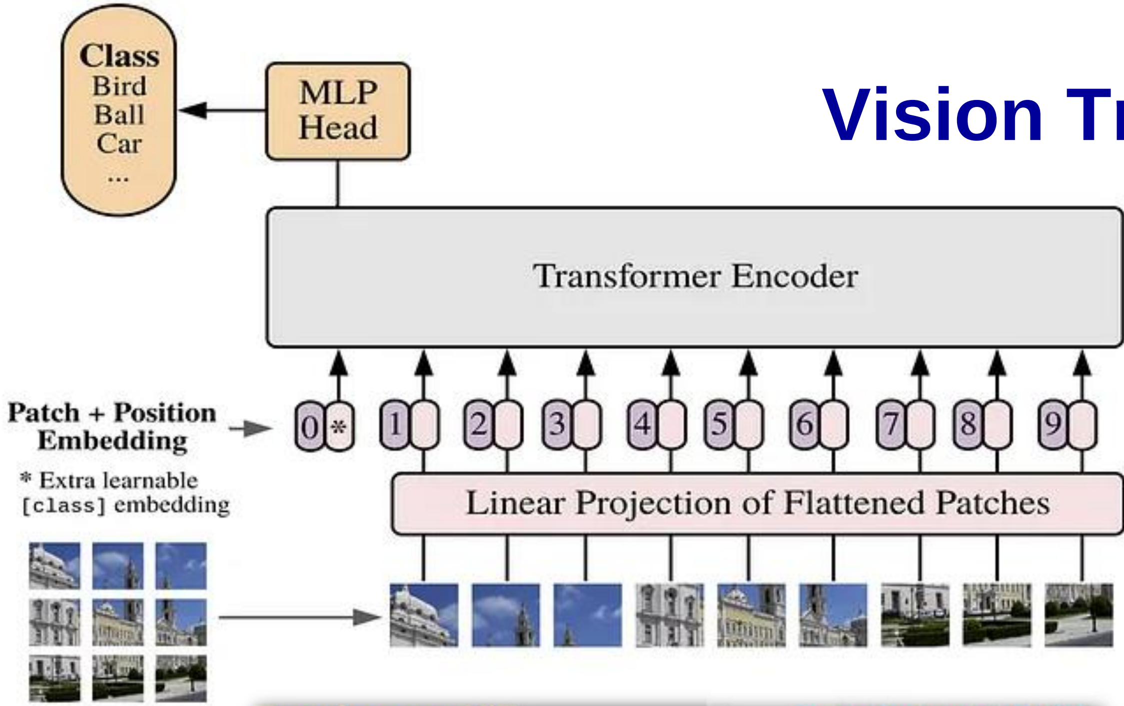
Short summary:

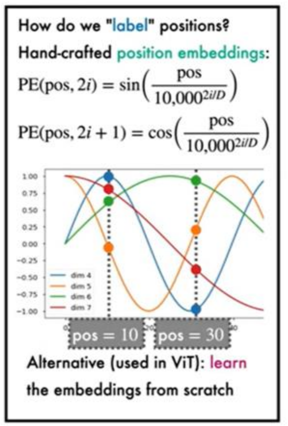
- 1x1 filters to reduce number of parameters and add regularization
- Inception layers
- Residual connections
- Learnable architectures
- Rise of deeper models from 5 layers to more than 1000
- However, a smaller net is often sufficient. There is still competition deep vs. wide layers, and dependence on the amount of training data.

- ImageNet results for classification are typically *<5%* in most of the latest submissions. Therefore, to show significant improvement we need another dataset.
- There is a need for new general datasets, as well as for particular specific problem domains. Some are already generated: MS COCO (*http://cocodataset.org*), Visual Genome Dataset (*https://visualgenome.org*)
- Additional research directions are aimed on improvement of speed and size of networks on mobile platforms.

Vision Transformer (ViT)

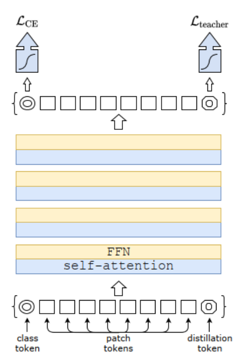
for Image Classification

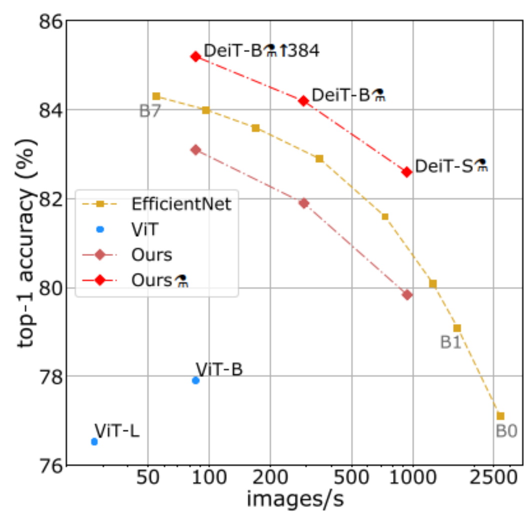




Data-Efficient Image Transformer (DeiT)

is a type of Vision Transformer for image classification tasks. It is like "ViT but trained with a procedure (initialization, optimization, data-augmentation, regularization and distillation) more adapted to a data starving regime." The model is trained using a teacher-student strategy specific to transformers. It relies on a ***distillation token*** ensuring that the student learns from the teacher through attention.



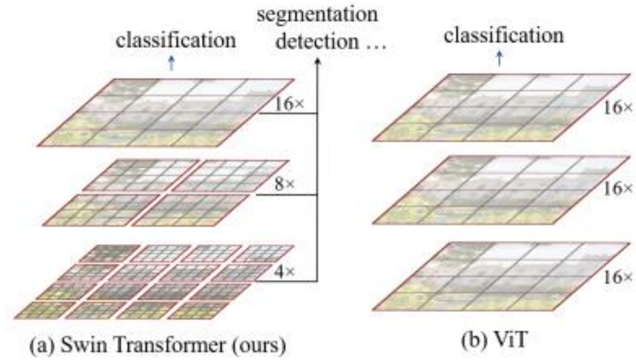


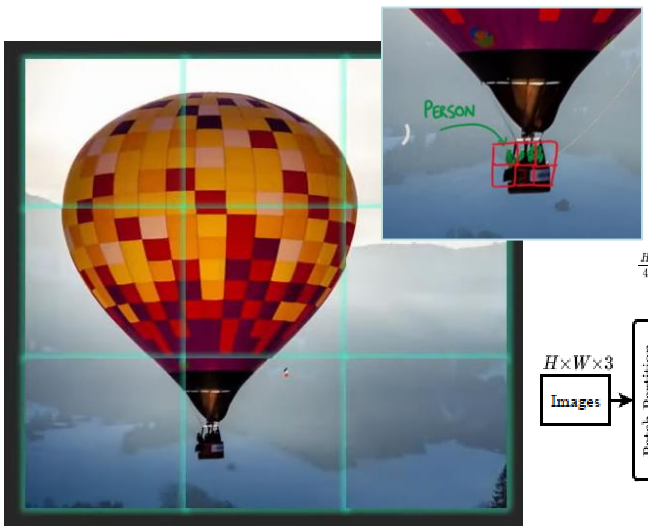
Relevant links:
https://arxiv.org/abs/2010.11929
https://arxiv.org/abs/2012.12877v2

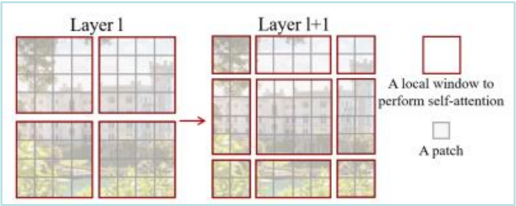
Swin Transformer (Swin-T)

Swin Transformer - *Hierarchical Vision Transformer using Shifted Windows (Shifted Windows base Self-Attention)*

It is a vision Transformer that capably serves as a general-purpose backbone for computer vision. Challenges in adapting Transformer from language to vision arise from differences between the two domains, such as large variations in the scale of visual entities and the high resolution of pixels in images compared to words in text. To address these differences, authors propose a hierarchical Transformer whose representation is computed with ***Shifted windows***. The shifted windowing scheme brings greater efficiency by limiting self-attention computation to non-overlapping local windows while also allowing for cross-window connection.







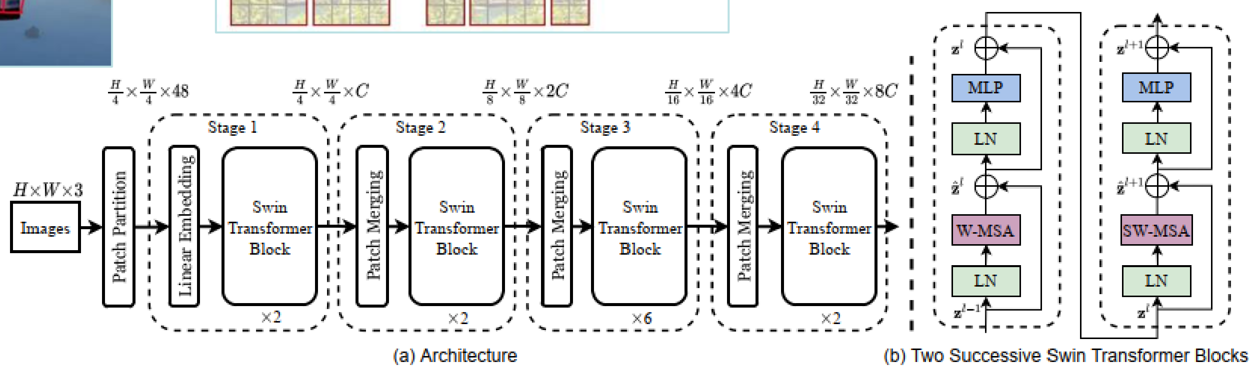


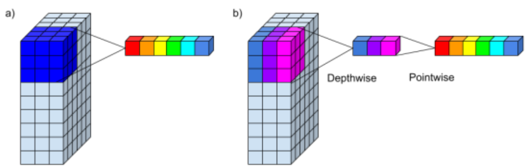
Figure 3. (a) The architecture of a Swin Transformer (Swin-T); (b) two successive Swin Transformer Blocks (notation presented with Eq. (3)). W-MSA and SW-MSA are multi-head self attention modules with regular and shifted windowing configurations, respectively.

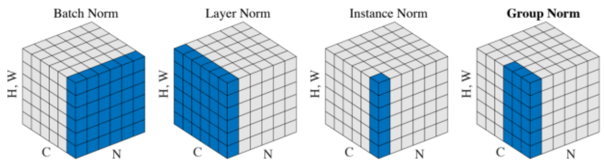
Relevant links:
https://arxiv.org/pdf/2103.14030v2.pdf
https://paperswithcode.com/paper/swin-transformer-hierarchical-vision

ConvNeXt and ConvNeXt v2

ConvNeXt is a pure ConvNet model that was proposed in the paper “A ConvNet for the 2020s”. It is constructed entirely from standard ConvNet modules, and it can be used for image classification, object detection, and segmentation tasks. ConvNeXt is similar to other ConvNet models in the sense that no new design is implemented, but it has better accuracy, performance, and scalability than Vision Transformers.

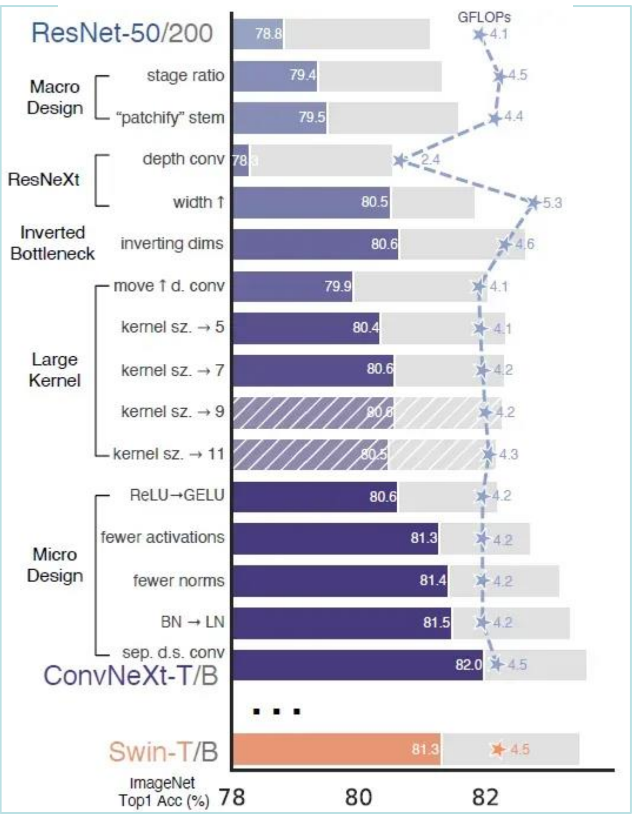
ConvNeXt is a pure ConvNet model that incorporates concepts from Vision Transformers (VITs) but does not directly use transformers. It focuses on using depth-wise convolution, Layer Normalization and the ResNext family of Convolutional Neural Networks for efficient image processing, while VIT relies on transformers and self-attention mechanisms for visual understanding.

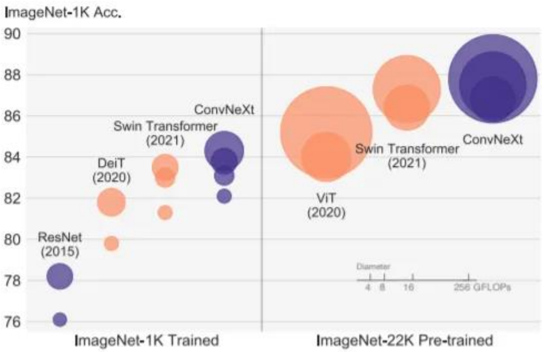




ConvNeXt V2 is a purely convolutional architecture that, after pretraining and fine-tuning, achieved state-of-the-art performance on ImageNet. ConvNeXt V2 improves upon ConvNeXt, which updated the classic ResNet.

The Design Journey Of ConvNeXt





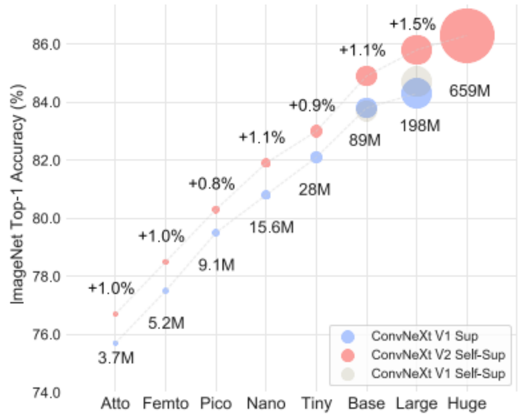


Figure 1. **ConvNeXt V2 model scaling**. The ConvNeXt V2 model, which has been pre-trained using our fully convolutional masked autoencoder framework, performs significantly better than the previous version across a wide range of model sizes.

Relevant links:

https://arxiv.org/abs/2201.03545v2
https://arxiv.org/abs/2301.00808v1
https://www.tensorflow.org/api_docs/python/tf/keras/applications/convnext

InceptionNeXt

InceptionNeXt

To speed up ***ConvNeXt***, authors build ***InceptionNeXt*** by decomposing the large kernel depthwise convolution into four parallel branches along the channel dimension (with Inception style). Thus, ***InceptionNeXt-T*** enjoys both ResNet-50's speed and ConvNeXt-T's accuracy.

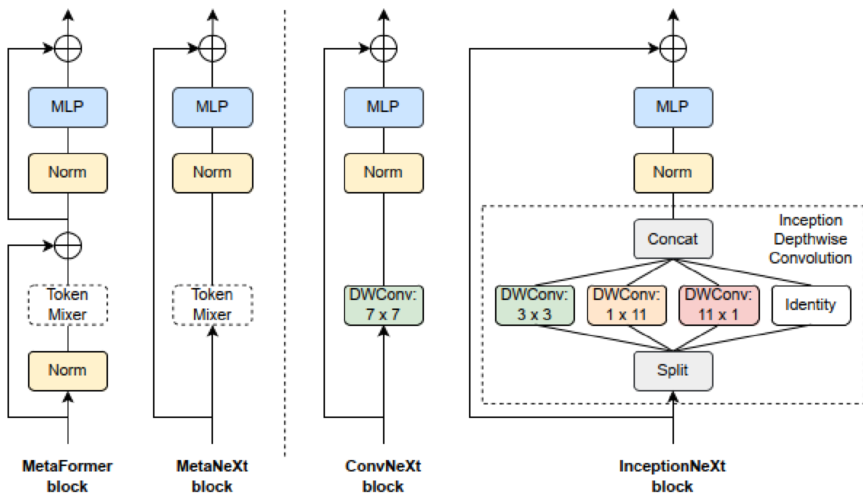


Figure 2: **Block illustration of MetaFormer, MetaNext, ConvNeXt and InceptionNeXt.** Similar to MetaFormer block [74], MetaNeXt is a general block abstracted from ConvNeXt [38]. MetaNeXt can be regarded as a simpler version obtained from MetaFormer by merging two residual sub-blocks into one. It is worth noting that the token mixer used in MetaNeXt cannot be too complex (*e.g.* self-attention [63]) or it may fail to train to converge. By specifying the token mixer as depthwise convolution or Inception depthwise convolution, the model is instantiated as ConvNeXt or InceptionNeXt block. Compared with ConvNeXt, InceptionNeXt is more efficient because it decomposes expensive large-kernel depthwise convolution into four efficient parallel branches.

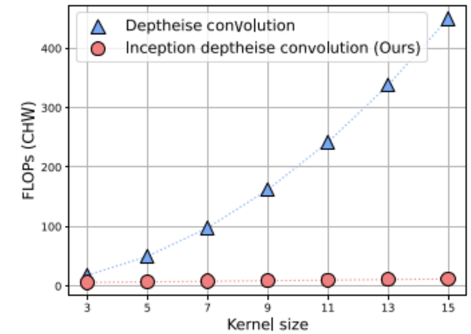


Figure 3: **Comparison of FLOPs between depthwise convolution and Inception depthwise convolution.** Inception depthwise convolution is much more efficient than depthwise convolution as kernel size increases.

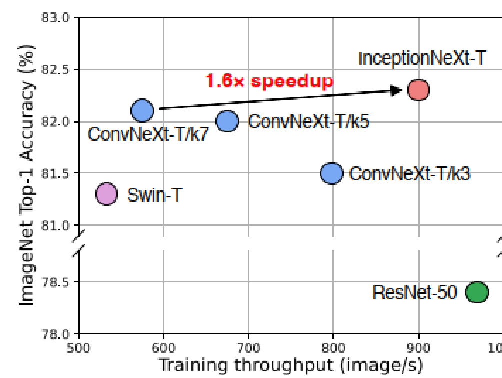


Figure 1: **Trade-off between accuracy and training throughput.** All models are trained under the DeiT training hyperparameters [61, 37, 38, 69]. The training throughput is measured on an A100 GPU with batch size of 128. ConvNeXt-T/kn means variants with depthwise convolution kernel size of $n \times n$. **InceptionNeXt-T enjoys both ResNet-50's speed and ConvNeXt-T's accuracy.**

Relevant links:
https://arxiv.org/pdf/2303.16900v1.pdf

Image Recognition

Keras Applications - are deep learning models that are made available alongside pre-trained weights. These models can be used for prediction, feature extraction, and fine-tuning. (https://keras.io/api/applications/) on the ImageNet validation dataset.

Weights are downloaded automatically when instantiating a model. They are stored at ~/.keras/models/.

Models for image classification with weights trained on *ImageNet*:

- *Xception*
- *VGG16*
- *VGG19*
- *ResNet50*
- *InceptionV3*
- *InceptionResNetV2*
- *MobileNet*
- *DenseNet*
- *NASNet*
- *EfficientNet*
- *EfficientNetV2*
- *ConvNeXt*

Model	Size	Top-1 Accuracy	Top-5 Accuracy	Parameters	Depth
Xception	88 MB	0.790	0.945	22,910,480	126
VGG16	528 MB	0.713	0.901	138,357,544	23
VGG19	549 MB	0.713	0.900	143,667,240	26
ResNet50	98 MB	0.749	0.921	25,636,712	-
ResNet101	171 MB	0.764	0.928	44,707,176	-
ResNet152	232 MB	0.766	0.931	60,419,944	-
ResNet50V2	98 MB	0.760	0.930	25,613,800	-
ResNet101V2	171 MB	0.772	0.938	44,675,560	-
ResNet152V2	232 MB	0.780	0.942	60,380,648	-
InceptionV3	92 MB	0.779	0.937	23,851,784	159
InceptionResNetV2	215 MB	0.803	0.953	55,873,736	572
MobileNet	16 MB	0.704	0.895	4,253,864	88
MobileNetV2	14 MB	0.713	0.901	3,538,984	88
DenseNet121	33 MB	0.750	0.923	8,062,504	121
DenseNet169	57 MB	0.762	0.932	14,307,880	169
DenseNet201	80 MB	0.773	0.936	20,242,984	201
NASNetMobile	23 MB	0.744	0.919	5,326,716	-
NASNetLarge	343 MB	0.825	0.960	88,949,818	-
EfficientNetB0	29 MB	-	-	5,330,571	-
EfficientNetB1	31 MB	-	-	7,856,239	-
EfficientNetB2	36 MB	-	-	9,177,569	-
EfficientNetB3	48 MB	-	-	12,320,535	-
EfficientNetB4	75 MB	-	-	19,466,823	-
EfficientNetB5	118 MB	-	-	30,562,527	-
EfficientNetB6	166 MB	-	-	43,265,143	-
EfficientNetB7	256 MB	-	-	66,658,687	-

EfficientNetV2B0	29	78.7%
EfficientNetV2B1	34	79.8%
EfficientNetV2B2	42	80.5%
EfficientNetV2B3	59	82.0%
EfficientNetV2S	88	83.9%
EfficientNetV2M	220	85.3%
EfficientNetV2L	479	85.7%
ConvNeXtTiny	109.42	81.3%
ConvNeXtSmall	192.29	82.3%
ConvNeXtBase	338.58	85.3%
ConvNeXtLarge	755.07	86.3%
ConvNeXtXLarge	1310	86.7%

The top-1 and top-5 accuracy refers to the model's performance on the ImageNet validation dataset.

Image Recognition

*Classify ImageNet classes with **ResNet50** model…*

```
Import tensorflow as tf
from tf.keras.applications.resnet50 import ResNet50
from tf.keras.preprocessing import image
from tf.keras.applications.resnet50 import preprocess_input, decode_predictions
import numpy as np

model = ResNet50(weights='imagenet')
img_path = 'elephant.jpg'
img = image.load_img(img_path, target_size=(224, 224))
x = image.img_to_array(img)
x = np.expand_dims(x, axis=0)
x = preprocess_input(x)
preds = model.predict(x)
# decode the results into a list of tuples (class, description, probability)
# (one such list for each sample in the batch)
print('Predicted:', decode_predictions(preds, top=3)[0])
# Predicted: [(u'n02504013', u'Indian_elephant', 0.82658225), (u'n01871265', u'tusker', 0.1122357), (u'n02504458', u'African_elephant', 0.061040461)]
```

In case of reading from URL…

```
…
import cv2
import urllib
…
img_path = 'http://www.helpmykidlearn.ie/images/uploads/daffodil_larger.jpg'
x = urllib.request.urlopen(img_path)
x = np.asarray(bytearray(x.read()), dtype="uint8")
x = cv2.imdecode(x, cv2.IMREAD_COLOR)
x = cv2.resize(x, (224, 224))
x = np.expand_dims(x, axis=0).astype(np.float32)
x = preprocess_input(x)
…
```

In case of reading from gDrive in Google Colaboratory…

```
from google.colab import drive
import os

drive.mount('/content/gdrive')
data_dir = '/content/gdrive/MyDrive/…/'
os.makedirs(data_dir, exist_ok=True)
```

or

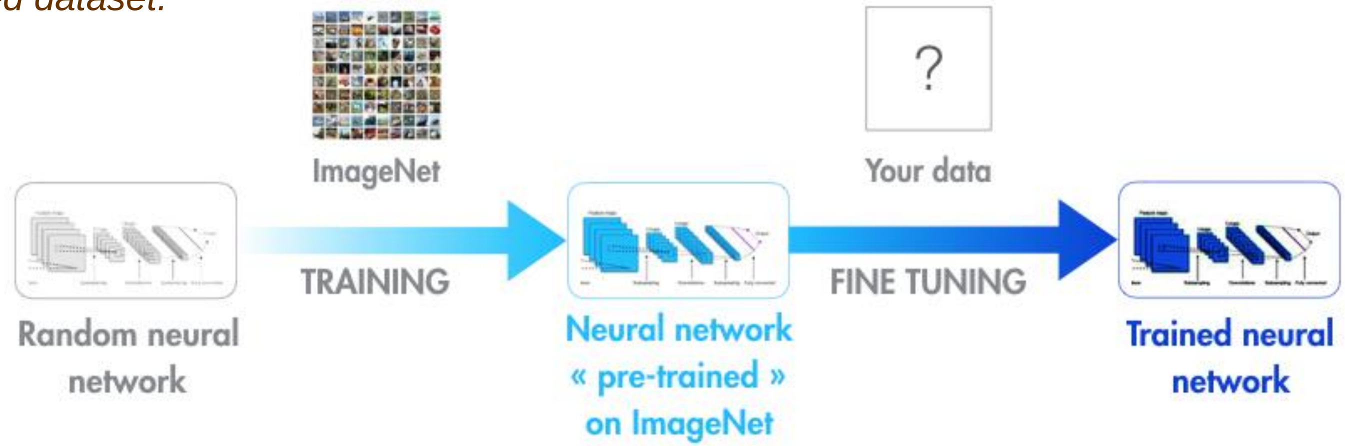
```
!pip install -U -q PyDrive ## you will have install for every colab session
```

```
from pydrive.auth import GoogleAuth
from pydrive.drive import GoogleDrive
from google.colab import auth
from oauth2client.client import GoogleCredentials
# Authenticate and create the PyDrive client to access gDrive.
auth.authenticate_user()
gauth = GoogleAuth()
gauth.credentials =
GoogleCredentials.get_application_default()
drive = GoogleDrive(gauth)
```

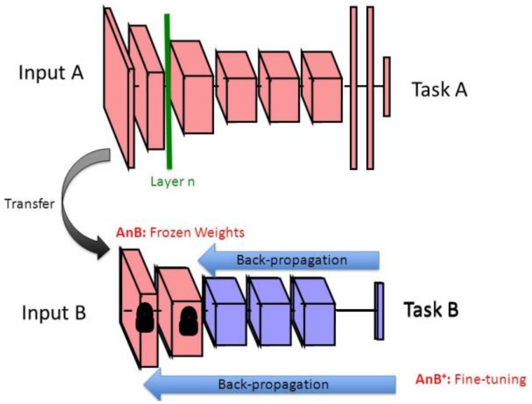
```
…
img_path = 'el_01.jpg'
img_file = drive.CreateFile({'id':'file ID from the sharable link of the file on your Google Drive'})
img_file.GetContentFile(img_path)
img = image.load_img(img_path, target_size=(224, 224))
x = image.img_to_array(img)
x = np.expand_dims(x, axis=0)
x = preprocess_input(x)
…
```

Transfer Learning

Been pre-trained based on a large training set, the model (with the weights and parameters of a network) that is capable to recognize basic features can be further "fine-tuned" for specific task based on customized dataset.



- *Using the pre-trained model as a feature extractor, the idea is to train the model by replacing the last layer of the network with customized classifier. It is important to freeze (not change) the weights of all the other layers during gradient descent/optimization.*
- *If task specific dataset is quite different from the dataset used for the original model, then more high layers suppose to be trained and only a couple of the low layers will be frozen.*



Relevant links:
https://en.wikipedia.org/wiki/Transfer_learning and http://cs231n.github.io/transfer-learning/
https://medium.com/owkin/transfer-learning-and-the-rise-of-collaborative-artificial-intelligence-41f9e2950657
https://arxiv.org/pdf/1411.1792v1.pdf and http://arxiv.org/pdf/1403.6382.pdf and https://arxiv.org/pdf/1310.1531.pdf
https://arxiv.org/pdf/1705.07706.pdf and https://arxiv.org/pdf/1707.09872.pdf

Image Recognition


Transfer Learning…

Build a classifier on top of the pre-trained VGG16 network for two similar classes of flowers (Daffodil and Galanthus Nivalis) from *flower17* dataset. It is a 17 category flower dataset with 80 images for each class. Due to limited images quantity, we need to do image data augmentation to abstract all the elements of any species.

```
Import tensorflow as tf
from tf.keras import applications, optimizers
from tf.keras.preprocessing.image import ImageDataGenerator
from tf.keras.models import Sequential, Model, load_model
from tf.keras.layers import Dropout, Flatten, Dense,
                              GlobalAveragePooling2D
from tf.keras import backend as k
from tf.keras.callbacks import ModelCheckpoint, LearningRateScheduler,
                                 TensorBoard, EarlyStopping
import cv2
from io import BytesIO
import numpy as np
import urllib
from PIL import Image

img_width, img_height = 224, 224
train_data_dir = "train"
validation_data_dir = "validation"
nb_train_samples = 120
nb_validation_samples = 40
batch_size = 16
epochs = 20
```

```
model = applications.VGG16(weights = "imagenet",
         include_top=False, input_shape = (img_width, img_height, 3))

# Freeze the layers which you don't want to train. Here the first 5 layers are
frosen.
for layer in model.layers[:5]:
   layer.trainable = False

#Adding custom Layers
x = model.output
x = Flatten()(x)
x = Dense(1024, activation="relu")(x)
x = Dropout(0.5)(x)
x = Dense(1024, activation="relu")(x)
predictions = Dense(2, activation="softmax")(x)

# creating the final model
model_final = Model(inputs = model.input, outputs = predictions)

# compile the model
model_final.compile(loss = "categorical_crossentropy", optimizer =
optimizers.SGD(learning_rate=0.0001, momentum=0.9),
metrics=["accuracy"])
```

Image Recognition

Transfer Learning…

Create the image data augmentation object for the training and testing dataset (Daffodil and Galanthus Nivalis). https://s3.amazonaws.com/italia18/transfer_learning_dataset.zip

```
# Initiate the train and test generators with data Augmentation
train_datagen = ImageDataGenerator(
                    rescale = 1./255,
                    horizontal_flip = True,
                    fill_mode = "nearest",
                    zoom_range = 0.3,
                    width_shift_range = 0.3,
                    height_shift_range=0.3,
                    rotation_range=30)
test_datagen = ImageDataGenerator(
                    rescale = 1./255,
                    horizontal_flip = True,
                    fill_mode = "nearest",
                    zoom_range = 0.3,
                    width_shift_range = 0.3,
                    height_shift_range=0.3,
                    rotation_range=30)

train_generator = train_datagen.flow_from_directory(
                    train_data_dir,
                    target_size = (img_height, img_width),
                    batch_size = batch_size,
                    class_mode = "categorical")

validation_generator = test_datagen.flow_from_directory(
                    validation_data_dir,
                    target_size = (img_height, img_width),
                    class_mode = "categorical")
```

```
# Save the model according to the conditions
callbacks = [ EarlyStopping(monitor='val_accuracy', min_delta=0,
                    patience=10, verbose=1, mode='auto'),
          ModelCheckpoint(os.path.join(data_dir,'DL-L4_02_model.h5'),
                    monitor='val_loss', save_best_only=True)
]# Fit the new final layers for the model
model_final.fit(
                    train_generator,
                    steps_per_epoch = nb_train_samples//batch_size,
                    epochs = epochs,
                    validation_data = validation_generator,
                    validation_steps = nb_validation_samples //batch_size,
                    callbacks = callbacks)
# test
im = cv2.resize(cv2.imread('test/galan.jpg'), (img_width, img_height))
im = np.expand_dims(im, axis=0).astype(np.float32)
im=preprocess_input(im)
print (im.shape)
out = model_final.predict(im)
model_classes=["Daffodil","Galanthus Nivalis"]
print (model_classes[np.argmax(out)])
print (out)
print ("Probability: ", out[0][np.argmax(out)])
```

```
(1, 224, 224, 3)
Galanthus Nivalis
[[7.9571405e-38 1.0000000e+00]]
Probability:  1.0
```

Image Recognition

Transfer Learning…

```
def show_result(im):
  im = cv2.resize(im, (img_width, img_height))
  im = np.expand_dims(im, axis=0).astype(np.float32)
  im=preprocess_input(im)
  out = model_final.predict(im)
  model_classes=["Daffodil","Galanthus Nivalis"]
  print (model_classes[np.argmax(out)])
  print (out)
  print ("Probability: ", out[0][np.argmax(out)])

def run_visualization(url):
  try:
    resp = urllib.request.urlopen(url)
    image = np.asarray(bytearray(resp.read()), dtype="uint8")
    orignal_im = cv2.imdecode(image, cv2.IMREAD_COLOR)
  except IOError:
    print('Cannot retrieve image. Please check url: ' + url)
    return
  print('running model on image %s...' % url)
  show_result(orignal_im)

# test
image_url = 'http://www.helpmykidlearn.ie/images/uploads/daffodil_larger.jpg'
run_visualization(image_url)
```

```
running model on image
http://www.helpmykidlearn.ie/images/uploads/daffodil_larger.jpg...
Daffodil
[[1. 0.]]
Probability:  1.0
```

Transfer Learning & Fine-tune

```
from tensorflow.keras.applications.inception_v3 import InceptionV3
from tensorflow.keras.preprocessing import image
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D

# create the base pre-trained
base_model = InceptionV3(weights='imagenet', include_top=False)

# add a global spatial average pooling layer
x = base_model.output
x = GlobalAveragePooling2D()(x)
# let's add a fully-connected layer
x = Dense(1024, activation='relu')(x)
# and a logistic layer -- let's say we have 200 classes
predictions = Dense(200, activation='softmax')(x)

# this is the model we will train
model = Model(inputs=base_model.input, outputs=predictions)

# first: train only the top layers (which were randomly initialized)
# i.e. freeze all convolutional InceptionV3 layers
for layer in base_model.layers:
    layer.trainable = False

# compile the model (should be done *after* setting layers to non-
trainable)
model.compile(optimizer='rmsprop', loss='categorical_crossentropy')

# train the model on the new data for a few epochs
model.fit(...)
```

```
# at this point, the top layers are well trained, and we can start fine
# tuning convolutional layers from inception V3. We will freeze the
# bottom N layers and train the remaining top layers.

# let's visualize layer names and layer indices to see how many layers
# we should freeze:
for i, layer in enumerate(base_model.layers):
   print(i, layer.name)

# we chose to train the top 2 inception blocks, i.e. we will freeze
# the first 249 layers and unfreeze the rest:
for layer in model.layers[:249]:
   layer.trainable = False
for layer in model.layers[249:]:
   layer.trainable = True

# we need to recompile the model for these modifications to take
# effect we use SGD with a low learning rate
from tensorflow.keras.optimizers import SGD
model.compile(optimizer=SGD(lr=0.0001, momentum=0.9),
                    loss='categorical_crossentropy')

# we train our model again (this time fine-tuning the top 2 inception
# blocks alongside the top Dense layers
model.fit(...)
```

Relevant links:
https://keras.io/api/applications/

Image Recognition

Customization of image classifier via retraining pre-trained model…

Transfer Learning with TensorFlow Hub…

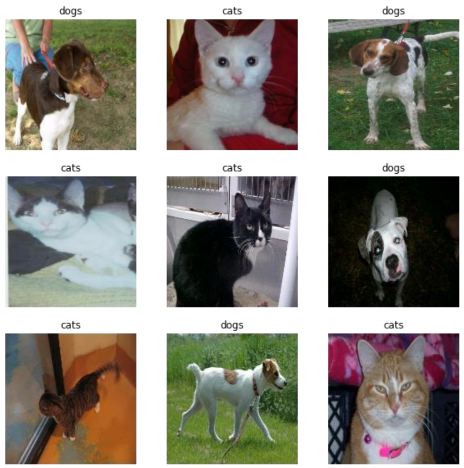
https://www.tensorflow.org/tutorials/images/transfer_learning_with_hub
https://tfhub.dev/

*Pre-trained model (e.g. MobileNet or Inception) doesn't know how to tell a tulip from a daisy. We can retrain existing model based on image collection of 5 different types/classes of flowers (**daisy**, **sunflowers**, **dandelion**, **tulips** and **roses**)*

Transfer Learning with Keras…

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

*Build a classifier on top of the pre-trained MobileNet network for recognize **dogs** and **cats**.*



Relevant links:
https://www.tensorflow.org/guide/keras/transfer_learning

Image Recognition

Wrap classification into Restful service using ***Flask framework*** (https://www.fullstackpython.com/flask.html):

Take an initial template for your restful service and modify it by adding classification functionality from *label_image_ws.py* and corresponding handling of HTTP POST request and response (*webapp.py*)

```
import label_image_ws as cl
from flask import Flask, request
from flask_restful import Api, Resource

app = Flask(__name__)
api = Api(app)

class Classification(Resource):
  def post(self):
    try:
      data = request.get_json()
    except:
      return {'errorMessage': 'Wrong request...'}, 500
    if(len(data)>0):
      response, err = cl.classify(data['imageURL'])
    else:
      return {'errorMessage': 'Please, provide a fileName...'}, 500
    if response:
      return response, 200
    else:
      return {'errorMessage': err}, 404

api.add_resource(Classification, '/classify')
if __name__ == "__main__":
    app.run()
```

Implement classification functionality in *label_image_ws.py*

Install Flask ...

```
pip install flask
```

```
pip install flask_restful
```

Run you web service (*webapp.py*)

```
python webapp.py
```

Image Recognition

Wrap classification into Restful service using ***Flask framework***:

After you run the service, system will tell you the address to access it…

- *Prepare a HTTP POST request with corresponding input in the body and execute it (e.g. http://127.0.0.1:5000/classify)*
- *You may use Postman as a restful service client (https://www.getpostman.com/downloads).*