# Artificial Neural Network Deep Learning

Subject: Image Classification

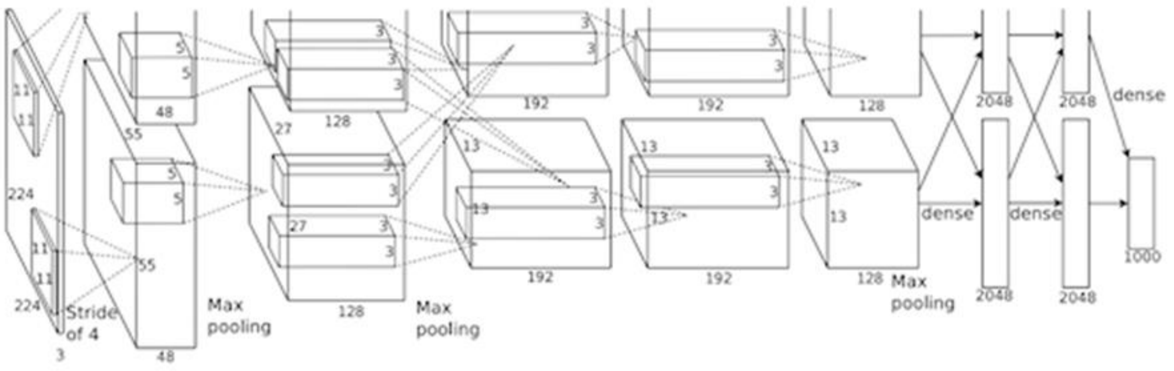
## Contents

# 1 Search For Deep Learning Architecture

## 1.1 AlexNet (2012)

The network was made up of 5 convolution layers, max-pooling layers, dropout layers, and 3 fully connected layers (60 million parameters and 500,000 neurons).

It was the first neural network to outperform the state of the art image classification of that time and it won the 2012 ILSVRC (ImageNet Large-Scale Visual Recognition Challenge).



AlexNet architecture (May look weird because there are two different "streams". This is because the training process was so computationally expensive that they had to split the training onto 2 GPUs)

Main Points:

- Used ReLU (Rectified Linear Unit) for the nonlinearity functions (Found to decrease training time as ReLUs are several times faster than the conventional tanh function).
- Used data augmentation techniques that consisted of image translations, horizontal reflections, and patch extractions.
- Implemented dropout layers in order to combat the problem of overfitting to the training data.
- Trained the model using batch stochastic gradient descent, with specific values for momentum and weight decay.

## 1.2 VGGNet (2014)

VGG Net reduced the size of each layer but increased the overall depth of the network (up to 16 - 19 layers) and reinforced the idea that convolutional neural networks have to be deep in order to work well on visual data.

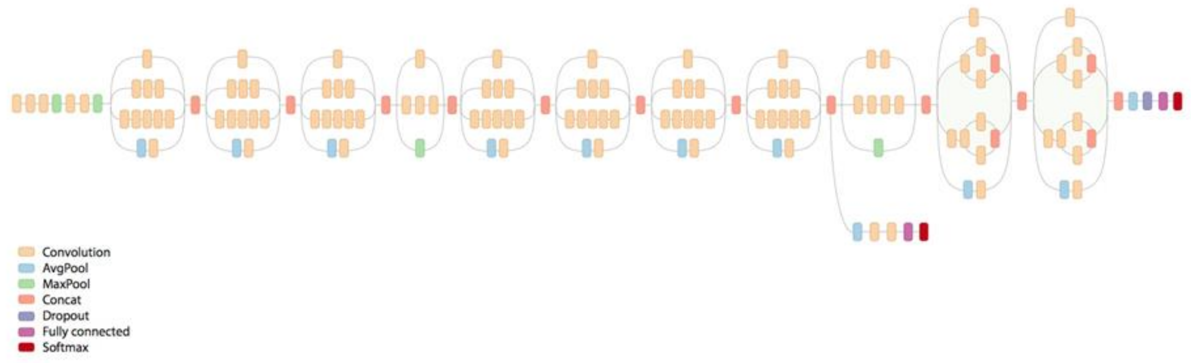
It finished at the first and the second places in the localisation and classification tasks respectively at 2014 ILSVRC.

Main Points

- The use of only 3x3 sized filters is quite different from AlexNet's 11x11 filters in the first layer: the combination of two 3x3 conv layers has an effective receptive field of 5x5.
- It simulates a larger filter while decreasing in the number of parameters.
- As the spatial size of the input volumes at each layer decrease (result of the conv and pool layers), the depth of the volumes increase due to the increased number of filters as you go down the network.
- Interesting to notice that the number of filters doubles after each maxpool layer. This reinforces the idea of shrinking spatial dimensions, but growing depth.

## 1.3 GoogLeNet and the Inception (2015)

GoogLeNet was one of the first models that introduced the idea that CNN layers didn't always have to be stacked up sequentially and creating smaller networks within the network :



Another view of GoogLeNet's architecture.

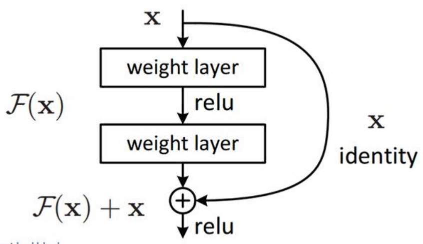
GoogLeNet is a 22 layer CNN and was the winner of ILSVRC 2014

Main Points

- Used 9 Inception modules in the whole architecture, with over 100 layers in total!
- No use of fully connected layers : it use an average pool instead, to go from a 7x7x1024 volume to a 1x1x1024 volume. This saves a huge number of parameters.
- Uses 12x fewer parameters than AlexNet.
- During testing, multiple crops of the same image were created, fed into the network, and the softmax probabilities were averaged to give the final solution.
- There are updated versions to the Inception module.

## 1.4 ResNet (2015)

The central idea of Residual Nets is to reformulate the layers as learning residual functions with reference to the layer inputs, instead of learning unreferenced functions : each individual layer learns “x” and not F(x)


x
weight layer
$\mathcal{F}(\mathbf{x})$
relu
x
identity
weight layer
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu

A residual block

That structure is then sequentially stacked several tenth of time (or more).

The winner of the classification task of ILSVRC 2015 is a ResNet network.

Main Points

- On the ImageNet dataset residual nets of a depth of up to 152 layers were used being 8x deeper than VGG nets while still having lower complexity.
- residual networks are easier to optimize than traditional networks and can gain accuracy from considerably increased depth.
- Residual networks have a natural limit : a 1202-layer network was trained but got a lower test accuracy, presumably due to overfitting.

## 1.5 SqueezeNet (2016)

Up to this point, research focused on improving the accuracy of neural networks, the SqueezeNet team took the path of designing smaller models while maintaining the accuracy unchanged.

It resulted into SqueezeNet, which is a convolutional neural network architecture that has 50 times fewer parameters than AlexNet while maintaining its accuracy on ImageNet.

3 main strategies are used while designing that new architecture:

- Replace the majority of 3x3 filters with 1x1 filters (to fit within a budget of a certain number of convolution filters).
- Decrease the number of input channels to 3x3 filters, using dedicated filters names squeeze layers.
- Downsample late in the network so that convolution layers have large activation maps.

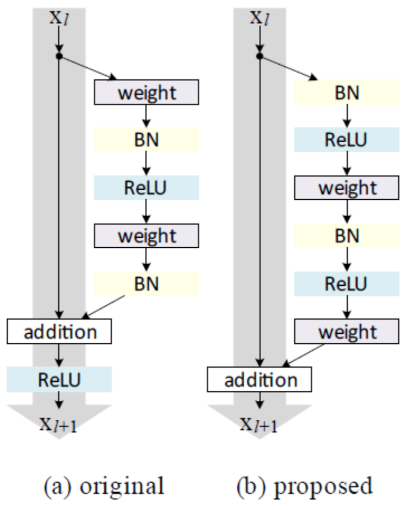
The first 2 strategies are about judiciously decreasing the quantity of parameters in a CNN while attempting to preserve accuracy.

The last one is about maximizing accuracy on a limited budget of parameters.

## 1.6 PreActResNet (2016)

PreActResNet stands for Pre-Activation Residuel Net and is an evolution of ResNet described above.

The residual unit structure is changed from (a) to (b) (BN: Batch Normalization) :



The activation functions ReLU and BN are now seen as “pre-activation” of the weight layers, in contrast to conventional view of “post-activation” of the weighted output.

This changes allowed to increase further the depth of the network while improving its accuracy on ImageNet for instance.

## 1.7 DenseNet (2016)

DenseNet is a network architecture where each layer is directly connected to every other layer in a feed-forward fashion (within each dense block). For each layer:

- the feature maps of all preceding layers are treated as separate inputs,
- its own feature maps are passed on as inputs to all subsequent layers.

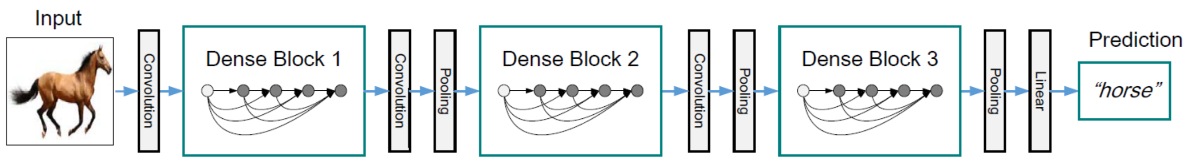


**Figure 2:** A deep DenseNet with three dense blocks. The layers between two adjacent blocks are referred to as transition layers and change feature-map sizes via convolution and pooling.

This connectivity pattern yields on CIFAR10/100 accuracies as good as its predecessors (with or without data augmentation) and SVHN. On the large scale ILSVRC 2012 (ImageNet) dataset, DenseNet achieves a similar accuracy as ResNet, but using less than half the amount of parameters and roughly half the number of FLOPs.

## 1.8 ResNeXt (2016)

The neural network architecture ResNeXt is based upon 2 strategies :

- stacking building blocks of the same shape (strategy inherited from VGG and ResNet)
- the “split-transform-merge” strategy that is derived from the Inception models and all its variations (split the input, transform it, merge the transformed signal with the original input).

That design exposes a new dimension, called "cardinality" (the size of the set of parallel transformations), as an essential factor in addition to the dimensions of depth and width.

It is empirically observed that :

- when keeping the complexity constant, increasing cardinality improves classification accuracy.
- increasing cardinality is more effective than going deeper or wider when we increase the capacity.

ResNeXt finished at the second place in the classification task at 2016 ILSVRC.

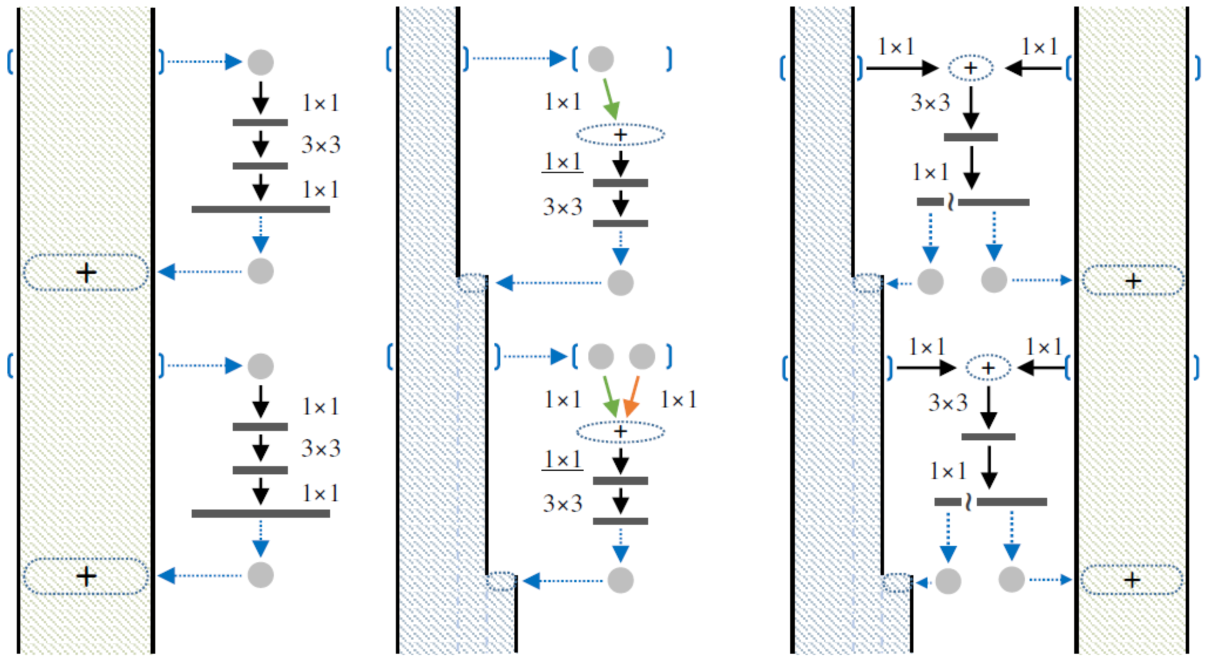
## 1.9 DPN (2017)

DPN is a family of convolutional neural networks that intends to efficiently merge residual networks and densely connected networks to get the benefits of both architecture:

- residual networks implicitly reuse features, but it is not good at exploring new ones,
- densely connected networks keep exploring new features but suffers from higher redundancy.

Consequently, DPN (Dual Path Network) contains 2 path :

- a residual alike path (green path, similar to the identity function),
- a densely connected alike path (blue path, similar to a dense connection within each dense block).



Residual Network Densely Connected Network Dual Path Architecture

## 1.10 NASNet (2017)

NASNet proposes a 2 step approach where :

- First, learn the architecture of the neural network itself,
- Then, train the network found.

The first step is by the Neural Architecture Search (NAS) framework, which uses a reinforcement learning search method to optimize architecture configuration.

To keep the computational effort affordable, the first step is performed on a subset of the complete dataset.

The key principle of this model is the design of a new search space (named the "NASNet search space") which enables transferability from the smallest dataset to the complete one.

## 1.11 SENet (2017)

Typical convolutional neural networks builds a description of its input image by progressively capturing patterns in each of its layers. For each of them, a set of filters are learnt to express local spatial connectivity patterns from its input channels. Convolutional filters captures informative combinations by fusing spatial and channel-wise information together within local receptive fields.

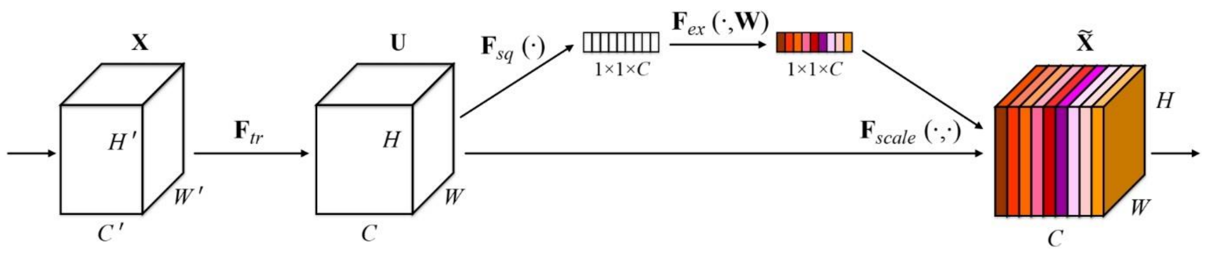
SENet (Squeeze-and-Excitation Networks) focuses on the relation between channels and recalibrates at transformation step its features so that informative features are emphazised and less useful ones suppressed (independently of their spatial location).

To do so, SENet uses a new architectural unit that consists of 2 steps :

- First, squeeze the block input (typically the output of any other convolutional layer) to create a global information using global average pooling,
- Then, "excite" the most informative features using adaptive recalibration.

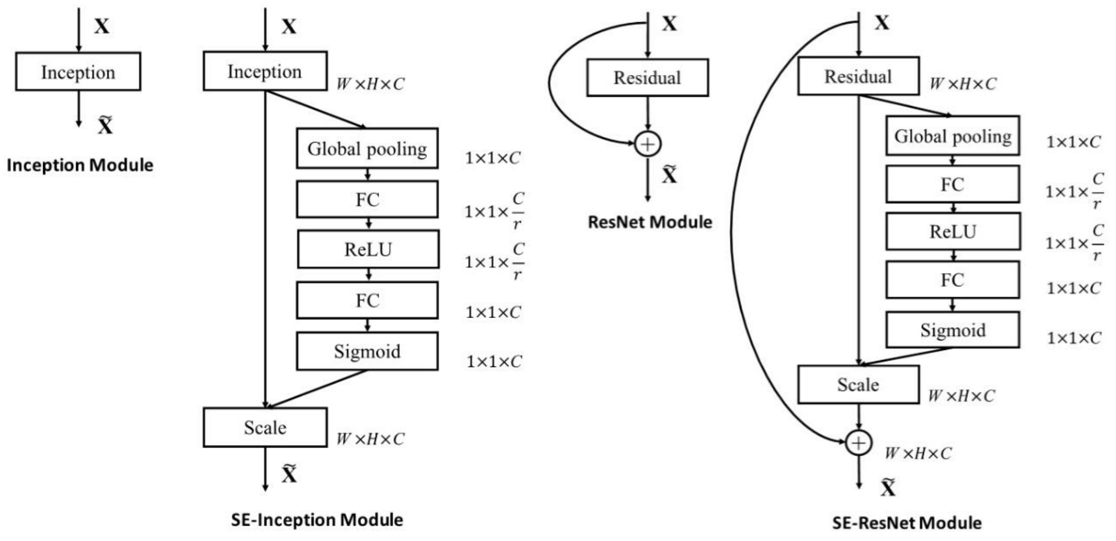
The adaptative recalibration is done as follows :

- reduce the dimension of its input using a fully connected layer (noted FC below),
- go through a non-linearity function (ReLU function),
- restore the dimension of its data using another fully connected layer,
- use sigmoid function to transform each output into a scale parameter between 0 and 1,
- linearly rescale each original input of the SE unit according to the scale parameters.



| | |
|---|---|
| X | : input of the block that will be enhanced by the squeeze and excitation method |
| $F_{tr}$ | : original convolutional operator to be enhanced |
| U | : output of $F_{tr}$ |
| $F_{sq}$ | : squeeze function |
| $F_{ex}$ | : excitation function (creates the scaling parameters) |
| $F_{scale}$ | : scaling function (scale the original output of $F_{tr}$ according to the SENet calibration ouput) |
| X̃ | : recalibrated $F_{tr}$ output |

Below are 2 examples of existing blocks enhanced with an SE unit:



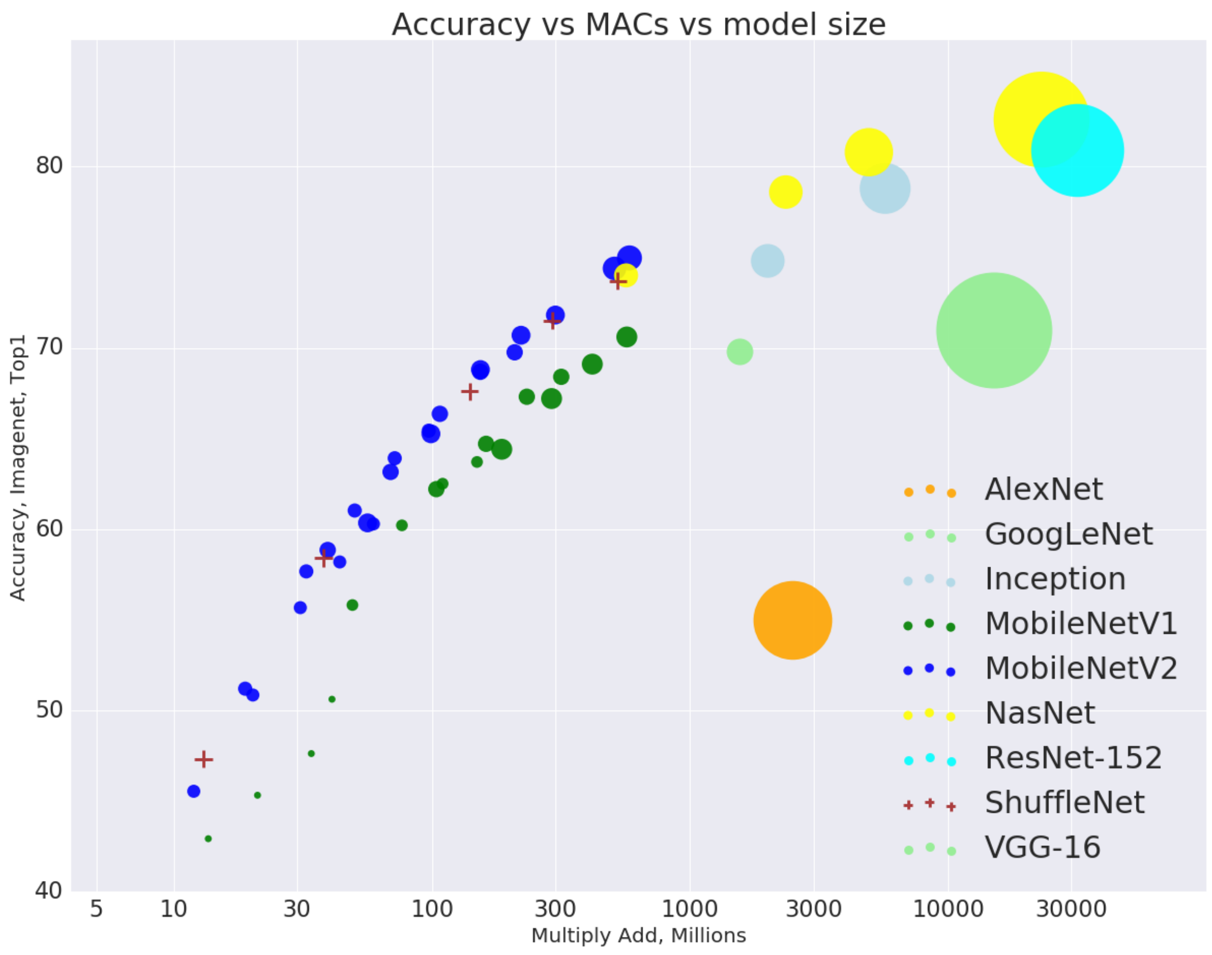
The winner of the classification task of ILSVRC 2017 is a modified ResNeXt integrating SE blocks.

## 1.12 MobileNet v1/v2

There has been also recently some effort to adapt neural networks to less powerfull architecture such as mobile devices, leading to the creation of a class of networks names MobileNet.

The diagram below illustrates that accepting a (slightly) lower accuracy than the state of the art, it is possible to create networks much less demanding in terms of resources (note that the multiply/add axis is on a logarithmic scale).

However since this study focuses on state of the art performances, those networks are not studied further.



# 2 Search For Scientific Papers

## 2.1 Scientific papers describing the neural networks

Here are the list of scientific papers describing the neural networks described in step 1:

| Date | Name | Paper URL |
|---|---|---|
| 2012 | AlexNet | https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks) |
| April-2014 | VGGNet | https://arxiv.org/abs/1409.1556 |
| 2015 | GoogLeNet/Inception | https://www.cv-foundation.org/openaccess/content_cvpr_2015/papers/Szegedy_Going_Deeper_With_2015_CVPR_paper.pdf |
| Dec-2015 | ResNet | https://arxiv.org/abs/1512.03385 |
| Feb-2016 | SqueezeNet | https://arxiv.org/abs/1602.07360 |
| March-2016 | PreActResNet | https://arxiv.org/abs/1603.05027 |
| August-2016 | DenseNet | https://arxiv.org/abs/1608.06993 |
| Nov-2016 | ResNeXt | https://arxiv.org/abs/1611.05431 |
| July-2017 | DPN | https://arxiv.org/abs/1707.01629 |
| July-2017 | NASNet | https://arxiv.org/abs/1707.07012 |
| Sept-2017 | SENet | https://arxiv.org/abs/1709.01507 |

## 2.2 Dataset used to test the neural networks

The table below summarizes the most common dataset found in those publications:

| Name | #Class | #Train | #Test | URL | Freely Available |
|---|---|---|---|---|---|
| CIFAR-10 | 10 | 50000 | 50000 | https://www.cs.toronto.edu/~kriz/cifar.html | Yes |
| CIFAR-100 | 100 | 50000 | 50000 | https://www.cs.toronto.edu/~kriz/cifar.html | Yes |
| ILSVRC | 1000 | 1200000 | 150000 | http://www.image-net.org/ | * |
| SVHN | 10 | 600000 | | http://ufldl.stanford.edu/housenumbers/ | Yes |
| NMIST | 10 | 60000 | 10000 | http://yann.lecun.com/exdb/mnist/ | Yes |
| PASCAL VOC | 20 | 11530 | | http://host.robots.ox.ac.uk/pascal/VOC/ | * |
| COCO | 91 | 328000 | | http://cocodataset.org/ | Yes |

* accessing the dataset requires a specific approval

Some comments about those data set:

- CIFAR datasets have been very commonly used in almost all papers,
- ILSVRC is the de facto standard in all recent publications but it is a huge database and accessing it requires a preliminary approval.
- NMIST is not used anymore as it is seen nowadays as "too easy".
- SVHN images in "format 2" are house numbers from Google street view in a NMIST like format.
- PASCAL VOC and COCO are mostly used in object detection and localization challenges as they often contains several objects (i.e. not well adapted to image classification).

## 2.3 Performance metrics

The classification performance is evaluated using two measures:

- the top-1 error,
- the top-5 error.

### 2.3.1 Top 1 error

The top 1 error is a multi-class classification error: it is the proportion of image that does not match the ground truth, i.e. the proportion of images incorrectly classified by the model.

### 2.3.2 Top 5 error

The top-5 error rate compares the ground truth against the top 5 predicted classes: an image is deemed correctly classified if the ground truth is among the top-5, regardless of its rank in them.

The top 5 error rate is then computed as the proportion of images such that the ground-truth category is outside of the top-5 predicted categories.

The ILSVRC challenge uses the top-5 error rate to rank the submission it receives.

## 2.4 Performances and execution times

### 2.4.1 Neural network performances

The table below summarizes the performances of the neural network listed in the first section:

| Name | CIFAR-10 | CIFAR-100 | ILSVRC top-1 | ILSVRC top-5 | SVHN |
|---|---|---|---|---|---|
| AlexNet | | | 45% | 16,4% | |
| VGG (ILSVRC'14) | | | 24,8% | 7,32% | |
| GoogLeNet (ILSVRC'14) | | | - | 6,66% | |
| ResNet (ILSVRC'15) | 6,43% | 27.22% | 19,38% | 3.57% | 2,01% |
| SqueezeNet | | | 42,5% | 19,7% | |
| PreActResNet | 4,62% | 22,71% | 20,1% | 4,8% | |
| DenseNet | 3,46% | 17,18% | 20,8% | 5,29% | 1,59% |
| ResNeXt | 3.58% | 17,31% | 20,4% | 5,3% | |
| ResNet (320x320) | | | 19,1% | 4,4% | |
| DPN | | | 19,93% | 5,12% | |
| NASNet | | 2,4% | 17,3% | 3,8% | |
| SENet (ILSVRC '17) | | | 18,68% | 4,47% | |
| SENet (320x320) | | | 16,88% | 3,58% | |

Except for ILSVRC that uses both top-1 and top-5 error rates, all other only uses top-1 error rate.

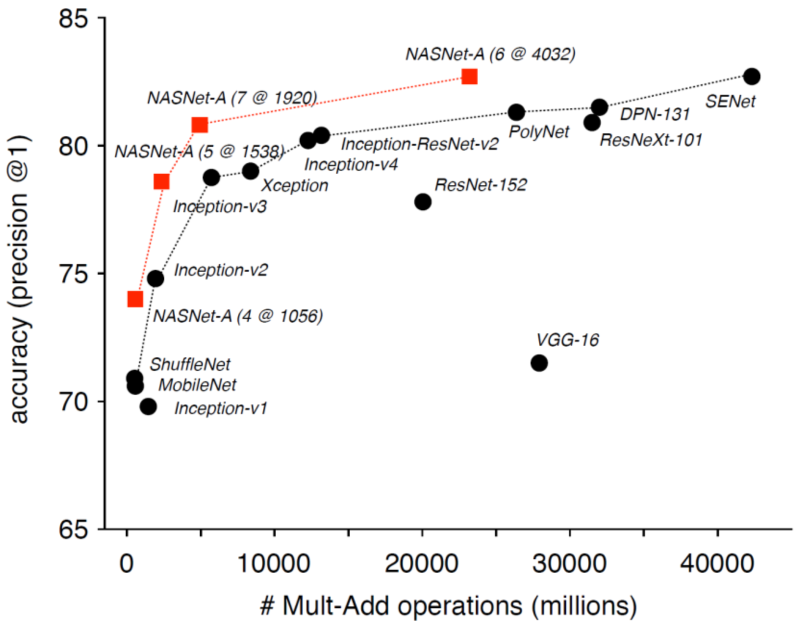
### 2.4.2 Hardware used and execution time

The table below summarizes the performances of the neural network listed in the first section:

| Name | Hardware | duration |
|---|---|---|
| AlexNet | 2 x GTX 580 3GB | 5-6 days |
| VGG (ILSVRC'14) | 4 x NVIDIA Titan Black GPU | 2-3 week |
| GoogLeNet (ILSVRC'14) | few high-end GPU | about 1 week |
| ResNet (ILSVRC'15) | 2 GPUs | |
| SqueezeNet | | |
| PreActResNet | CIFAR : 2 GPUs | CIFAR : 27 hours |
| | ImageNet: 8 GPUs | ImageNet: 3 weeks |
| DenseNet | | |
| ResNeXt | | |
| ResNet (320x320) | | |
| DPN | 4 x K80 GPU | |
| NASNet | 500 x P100 (pool of worker) | 4 days (2000 GPU-hour) |
| SENet (ILSVRC '17) | | |
| SENet (320x320) | 64 x NVIDIA Titan X card | |

### 2.4.3 Special case of NASNet

The first step of NASNet that searches for the network architecture itself can be quite expensive in terms of computational cost (which is why a pool of 500 GPU is used). However, once found, the resulting neural networks have a lower computational cost (diagram below is based on ImageNet 2012 ILSVRC dataset) :



## 3 Search For Challenges / scientific competitions

The most important image classification challenge is the classification task of the ImageNet Large Scale Visual Recognition Competition (ILSVRC).

## 4 Search For Implementations

The table below lists the PyTorch implementation for the models listed above found in github. The table also indicates the names of the models directly available in PyTorch when they exist.

| torchvision. models | Name | Github repo *(official repo cited in arxiv in italic)* | Watch | Star | Fork | Sco-re (*) |
|---|---|---|---|---|---|---|
| alexnet | **AlexNet** | | | | | |
| vgg* | **VGGNet** | | | | | |
| inception_v3 | **GoogLeNet/ Inception** | | | | | |
| resnet* | **ResNet** | | | | | |
| squeezenet* | ***SqueezeNet*** | *https://github.com/DeepScale/SqueezeNet* | ***93*** | ***1360*** | ***483*** | **1936** |
| | ***PreActResNet*** | *https://github.com/KaimingHe/resnet-1k-layers* | *51* | *518* | *165* | 734 |
| | **PreActResNet** | https://github.com/kuangliu/pytorch-cifar | **37** | **654** | **272** | **963** |
| densenet* | **DenseNet** | https://github.com/andreasveit/densenet-pytorch | 6 | 147 | 52 | 205 |
| | **DenseNet** | https://github.com/bamos/densenet.pytorch | 11 | 391 | 73 | 475 |
| | **DenseNet** | https://github.com/gpleiss/efficient_densenet_pytorch | 24 | 515 | 107 | 646 |
| | ***DenseNet*** | *https://github.com/liuzhuang13/DenseNet* | ***143*** | ***2471*** | ***602*** | **3216** |
| | **ResNeXt** | https://github.com/prlz77/ResNeXt.pytorch | 9 | 171 | 35 | 215 |
| | **ResNeXt** | https://github.com/D-X-Y/ResNeXt-DenseNet | 8 | 200 | 36 | 244 |
| | ***ResNeXt*** | *https://github.com/facebookresearch/ResNeXt* | ***59*** | ***1168*** | ***183*** | **1410** |
| | **DPN** | https://github.com/oyam/pytorch-DPNs | 5 | 60 | 13 | 78 |
| | **DPN** | https://github.com/rwightman/pytorch-dpn-pretrained | 6 | 98 | 17 | 121 |
| | **DPN** | https://github.com/cypw/DPNs | **28** | **367** | **119** | **514** |
| | **NASNet** | https://github.com/wandering007/nasnet-pytorch | 1 | 6 | 1 | 8 |
| | ***SENet*** | *https://github.com/moskomule/senet.pytorch* | ***6*** | ***168*** | ***32*** | **206** |

(*) Score is the arbitrary sum of Watch, Star and Fork.

There is no PyTorch implementation in github for the oldest models. However, this is not an issue as they are directly available as part of the Torchvision package.

# 5 Evaluate implementations

## 5.1 General observations

As the accurary of the models globally increase, it makes sense to focus on the most recent networks.

Most of the modern papers points to a github repository and when several implementations are available, it is that one that will have the highest “Score” (since it has a large visibility due to the original publication).

The most common dataset is ImageNet from ILSVRC and it would therefore make sense to consider it, but :

- Accessing to it requires a discretionary approval (my request was left unanswered),
- it is a huge dataset and training a model on it typically requires several hundred GPU hours.

The alternative is CIFAR-10/CIFAR-100 because to up very recently it was very commonly used. However, it is worth noting that some of the lastest papers do not publish results for that dataset anymore.

## 5.2 Hardware used

All the experiments below are run on AWS using a p3.2xlarge instance running the Amazon Image “Deep Learning AMI (Ubuntu)” Version 8.0 (ami-5d7c5024).

The reason for choosing that instance is that is the “smallest” one having a Tesla V100 GPU (with 16GB of RAM).

The software environment selected from the Amazon image is PyTorch 0.4 and Python3.6.

## 5.3 NASNet

Nasnet is the latest publication and its classification results seem to be the state of the art, but its “official” implementation is on tensorflow and there is no “complete” implementation in PyTorch :

- The PyTorch repository has a very low “Score” of 8,
- It only allows to evaluate imagenet accuracy through a pretrained model from tensorflow.

For all those reasons, the study of NASNet is not pursued further, the general feeling being that it is not yet mature enough in PyTorch.

## 5.4 SENet

SENet is the next most promising network after NASNet when looking at its performance.

### 5.4.1 Github analysis

The chosen repository for that network is https://github.com/moskomule/senet.pytorch, and it is because:

- It is pointed at by the repository referenced in the official arXiv paper,
- It has 168 stars and 32 forks.

The README of the repository explains that it contains the implementation of a SE-ResNet20 ready to be run of the dataset CIFAR-10.

As explained in section 1, a SENet is in fact a "classic" neural network to which is added an SE block in order to improve it.

In this implementation, the author chose to improve a ResNet20 by adding an SE block to it.

In terms of performance, the README says:

| | **ResNet20** | **SE-ResNet20 (reduction 4 or 8)** |
|---|---|---|
| max. test accuracy | 92% | 93% |

The official ResNet arXiv paper gives the following results for CIFAR-10 (in section "4.2. CIFAR10 and Analysis"):

| | #layers | #params | Error (%) |
|---|---|---|---|
| **ResNet** | **20** | **0.27M** | **8.75** |
| ResNet | 32 | 0.46M | 7.51 |
| ResNet | 44 | 0.66M | 7.17 |
| ResNet | 56 | 0.85M | 6.97 |
| ResNet | 110 | 1.7M | 6.43 |
| ResNet | 1202 | 19.4M | 7.93 |

The arXiv papers says that the error rate is 8.75%, meaning that the accuracy is 91.25% which is comparable to the 92% of the repository README.

### 5.4.2 Code analysis

The following files of the github repository are used when using the CIFAR-10 dataset:

- baseline.py : implementation of already known models:
  - resnet20,32,56,110,
  - preact_resnet20,32,56,110.

  It is worth noting that all those networks rely on the same macro architecture (both has 6*n layers+2 where n is the number of repetition of the basic block). That property appears in the implementation: each family of network has a single constructor that receives the number of repetition of the basic block ("n") as a parameter.

- cifar.py : contains the dataloader that loads the CIFAR-10 dataset. The role of the dataloader is to transform the specific format of the dataset to one that suits the PyTorch framework.
  It also contains the main part of the program (from a controlling stand point) :

```
optimizer = optim.SGD(params=model.parameters(), lr=1e-1,
                         momentum=0.9, weight_decay=1e-4)
    scheduler = StepLR(optimizer, 80, 0.1)
    trainer = Trainer(model, optimizer, F.cross_entropy)
    trainer.loop(200, train_loader, test_loader, scheduler)
```

- se_module.py : contains the squeeze and excitation block as described in section 1:
    - the global pooling (nn.AdaptiveAvgPool2d),
    - a Fully Connected layer to reduce the dimension (nn.Linear),
    - the non linearity function ReLU (nn.ReLU)
    - another Fully Connected layer to restore the dimension (size exchanged),
    - the sigmoid function (nn.Sigmoid) that rescales between 0 and 1 the feature calibration parameters.

```
class SELayer(nn.Module):
    def __init__(self, channel, reduction=16):
        super(SELayer, self).__init__()
        self.avg_pool = nn.AdaptiveAvgPool2d(1)
        self.fc = nn.Sequential(
                nn.Linear(channel, channel // reduction),
                nn.ReLU(inplace=True),
                nn.Linear(channel // reduction, channel),
                nn.Sigmoid()
        )
```

- se_resnet.py : implementation of the SENet familly following the official arXiv paper for imageNet and CIFAR dataset.
- utils.py : utility classes said to be from PyTorch (Trainer that takes care of training and testing the network for as may epoch as required, StepLR that takes care of regularly decreasing the learning rate).
  Below is the main loop that runs the dataset (training or test) mini batch per mini batch on the network:

```
   for data, target in tqdm(data_loader, ncols=80):
      if self.cuda:
          data, target = data.cuda(), target.cuda()
      data, target = Variable(data, volatile=not is_train),
                     Variable(target, volatile=not is_train)
      output = self.model(data)
      loss = self.loss_f(output, target)
      loop_loss.append(loss.data[0] / len(data_loader))
      correct.append(
          float((output.data.max(1)[1] ==
                 target.data).sum()) / len(data_loader.dataset))
      if is_train:
          self.optimizer.zero_grad()
          loss.backward()
          self.optimizer.step()
  mode = "train" if is_train else "test"
  print(">>>[{}] loss: {:.2f}/accuracy: {:.2%}".
        format(mode, sum(loop_loss), sum(correct)))
  return loop_loss, correct
```

  Some observations:
    - the for loop is used to iterate over the batches,
    - 'self.model(data)' is the forward pass (common to training or testing),
    - 'loss = self.loss_f(output, target)' : is the computation of the loss
    - if in training backward propagation happens 'loss.backward()'
    - The accurary that is displayed is a top-1 accuracy; summed over the whole batch/dataset: 'output.data.max(1)[1] == target.data).sum()'

### 5.4.3 Test results

Running the code only requires installing an additional package named ‘tqdm’ that is not deep learning related (it is a fancy progress bar).

The program trains the network for 200 epochs and the final test accuracy is 92.43%, which corresponds to an error rate of 7.57%. Training the network took 1h25.

The improvement from 8.75% to 7.57% is of 1.18% which corresponds to the description of the repository README, from 92% to 93%.

However, It is worth noted that on CIFAR-10, the arXiv papers says that the best performance is obtained with the more complex network ResNet110 (see table above).

Since the only change between the difference version of ResNet is the number of repetition of the basic blocks (which is shown also in the existing network constructor), the following lines of code are added in seresnet.py and cifar.py to test a SE-ResNet110.

```
def se_resnet110(**kwargs):
    """Constructs a ResNet-108 model.
    """
    model = CifarSEResNet(CifarSEBasicBlock, 18, **kwargs)
    return model


model = se_resnet110(num_classes=10, reduction=reduction)
```

The average error rate of a ResNet110 is 6.43% (see table above copied the the official arXiv paper)

When running the modified code:

- it reaches an accuracy of 94.47%, that is to say an error rate of 5.53%. It can be observed that the addition the SE blocks lower by 1.1% the error rate which is about the same value as for the ResNet20.
- It took 2h30 to run 200 epochs.

## 5.5 DPN (Dual Path Networks)

CIFAR dataset is not addressed by the DPN arXiv paper and it is the same for the github repositories.

Consequently that model is not studied further.

## 5.6 ResNeXt

ResNeXt is the next most promising network after DPN when looking at its performance. It worth noted that in 2016, it is a ResNeXt based network that finished second at the ILSVRC classification task.

### 5.6.1 Github analysis

The chosen repository for that network is https://github.com/prlz77/ResNeXt.pytorch, and it is because:

- It is pointed at by the repository referenced in the official arXiv paper,
- It has 172 stars and 36 forks.

The README of the repository explains that it contains an implementation ready to be run of the dataset CIFAR-10 and that it should reach an accuracy of 3.65% (value from the official arXiv paper which corresponds to a ResNeXt-29, 8x64d).

### 5.6.2 Code analysis

The following files of the github repository are used when using the CIFAR-10 dataset:

- train.py : single file taking care of the training of the model. Below is the code that construct the network and its parameters matches a ResNeXt-29, 8x64d:

```
net = CifarResNeXt(
        args.cardinality=8,     // number of convolution groups
        args.depth=29,          // number of layers
        nlabels=10,             // number of classes of the dataset
        args.base_width=64,     // base number of channels in each group
        args.widen_factor=4)    // factor to adjust the channel
                                // dimensionality. 4 -> 64
```

The rest of the code is very similar to the other models:

- the dataset is read through a dataloader and is then augmented,
- the training is done per batch,
- a test is performed after each epoch (calculation a top-1 accuracy rate). The current accuracy is saved in a log file and the best accuracy obtained so far is printed on screen.

Below is the detail of the training loop, it contains all the expected steps, i.e. forward and backward pass :

```
def train():
     net.train()
     loss_avg = 0.0
     for batch_idx, (data, target) in enumerate(train_loader):
         data, target = torch.autograd.Variable(data.cuda()),
                        torch.autograd.Variable(target.cuda())

         # forward
         output = net(data)

         # backward
         optimizer.zero_grad()
         loss = F.cross_entropy(output, target)
         loss.backward()
         optimizer.step()
```

### 5.6.3 Test results

The following commands were used to install the CIFAR-10 dataset and to run the network (command derived from the README repository) :

```
cd /mnt/hdd
wget https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
mkdir cifar.python
cd cifar.python/
tar xvf ../cifar-10-python.tar.gz
cd /mnt/hdd/ResNeXt.pytorch
python train.py /mnt/hdd/cifar.python cifar10 -s ./snapshots --log ./logs -
-ngpu 1 --learning_rate 0.05 -b 128
```

The program trains the network for 300 epochs and the final test accuracy is 96%, which corresponds to an error rate of 4%. Training the network took 12h50.

The average error rate of a ResNeXt given in the official arXiv paper is 3.58% which is similar to the value obtained in the test above.

## 5.7 DenseNet

### 5.7.1 Github analysis

The github repository chosen for DenseNet is https://github.com/gpleiss/efficient_densenet_pytorch as it is pointed at by the official repository (as it does not support PyTorch).

It has the highest ranking after the official repository (in terms of Watch/Star/Fork).

The repository README insists of the efficiency of the implementation in terms of memory consumption (at the cost of 15% to 20% of speed).

### 5.7.2 Code analysis

The code is organized in 2 files :

- models/densenet.py : constructor of the DenseNet network following the official arXiv paper
- demo.py : that file runs the network (using similar PyTorch calls as a SENet). Here is an extract of its main testing loop:

```
for batch_idx, (input, target) in enumerate(loader):
    ..
    # compute output
    output = model(input_var)
    loss = torch.nn.functional.cross_entropy(output, target_var)

    # measure accuracy and record loss
    batch_size = target.size(0)
    _, pred = output.data.cpu().topk(1, dim=1)
    error.update(torch.ne(pred.squeeze(),
                 target.cpu()).float().sum() / batch_size, batch_size)
    losses.update(loss.data[0], batch_size)
```

It can be seen that the program uses a top-1 error rate sum over each batch and then accumulated over the entire dataset.

### 5.7.3 Test results

The program trains the network for 300 epochs and the final error rate was 5.20%. Training the network took 6h15.

By default the program runs a DenseNet-BC (k = 12) using data augmentation; that use case is described in the official arXiv paper and it should give a top 1 error rate of 4.51%.

The results obtained by running the program are a little higher than expectation but still not too far.

# 6 Conclusion

In this study, we evaluated 3 convolutional neural networks SENet, ResNeXt and SENet on the dataset CIFAR-10. The table below summarizes the results:

| Neural Network | Top-1 Error Rate | Training Time (Tesla V100) |
|---|---|---|
| SENet | 5.53% | 2h30 |
| ResNeXt | 4% | 12h50 |
| DenseNet | 5.2% | 6h15 |

From those results, there are 2 possible options :

- ResNeXt because it has the lowest error rate,
- SENet because its results is no far but computational cost is much lighter (ratio of 1:5).

DenseNet is now not the best choice anymore as SENet gives similar results while having only a third of its computational needs.

However, we also need to take into account the dataset used : CIFAR-10 has the advantage of being relatively fast to run while having also been widely used in most papers, but it only consists of very small images of 16x16 pixels.

On the other hand, the ImageNet dataset that consists of much larger images seems to be closer to a real use case; and in that case SENet has slightly better results than ResNeXt.

That along with the much lower computational cost would point at SENet to start a real industrial project. One should then start by confirming those good results with the real images coming from the project, using a pre-trained network as a starting point. If results are not up to expectation, ResNeXt could be checked, but care should be taken regarding its computational cost.

Alternatively, if the computational cost is too high even for SENet and if some compromises can be made on the performances, mobilenet could be investigated as a fallback solution.