Universitatea de Vest din Timişoara

Hybrid Deep Learning Topology for Image Classification

Petru Radu

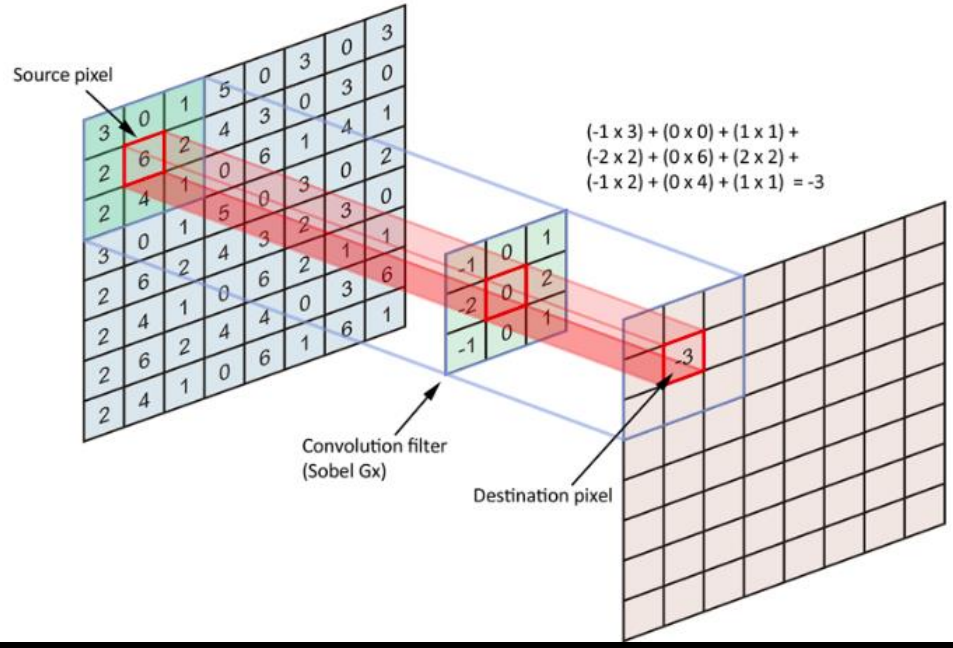
NESS

petru.radu@ness.com

Introduction

Classical Neural Networks employed in image classification have a large number of parameters => impossible to train such a system without overfitting the model due to the lack of a sufficient number of training examples.

But with Convolutional Neural Networks(CNN), the task of training the whole network from the scratch can be carried out using existing large (enough) datasets like ImageNet.



Introduction

One important aspect of deep learning is understanding the underlying working principles of a model that was designed to solve a certain problem.

A very popular deep neural network model is VGG (*). VGG stands for Visual Geometry Group, which is the research group that proposed it.

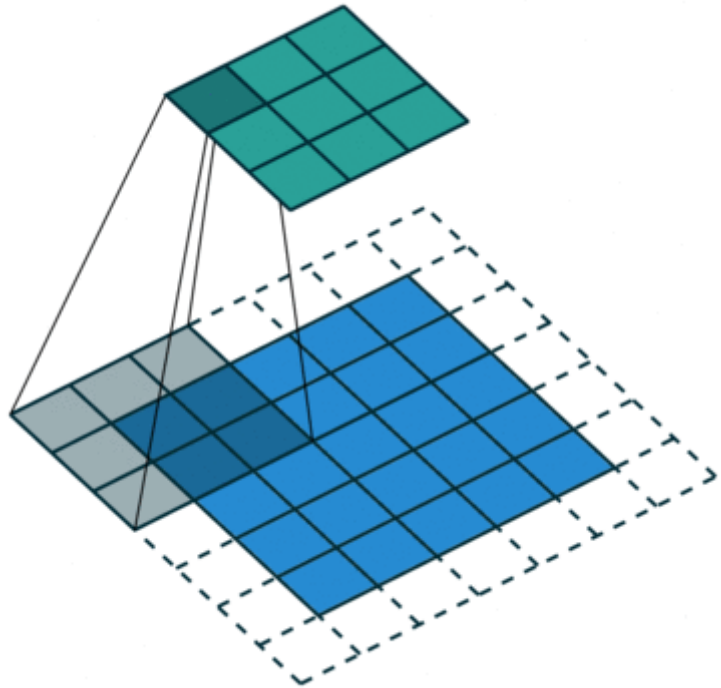
One of the main benchmarks in image classification was *Image Net Large Scale Visual Recognition Challenge* (ILSVRC) (**) was evaluated on ImageNet dataset.

(*) K. Simonoyan and A. Zisserman, “Very Deep Convolutional Networks for Large-Scale Image Recognition”, ICLR 2015

(**) Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg and Li Fei-Fei,” *ImageNet Large Scale Visual Recognition Challenge”*. *IJCV,* 2015

Introduction

CONVOLUTIONAL NEURAL NETWORKS ARCHITECTURES			
Architecture	Top-1 Accuracy	Top-5 Accuracy	Year
Alexnet	57.1	80.2	2012
Inception-V1	69.8	89.3	2013
VGG	70.5	91.2	2013
Resnet-50	75.2	93	2015
InceptionV3	78.8	94.4	2016

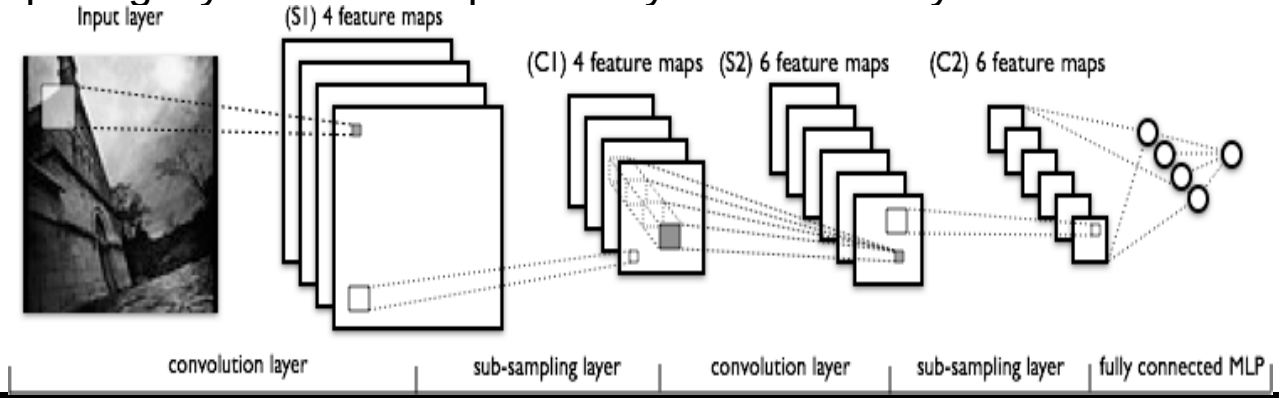


Introduction - LeNet

VGG won the competition in 2014 and obtained high results in both image localization and image classification tasks.

The VGG architecture had as starting the architecture of LeNet model, which is shown in the figure below.

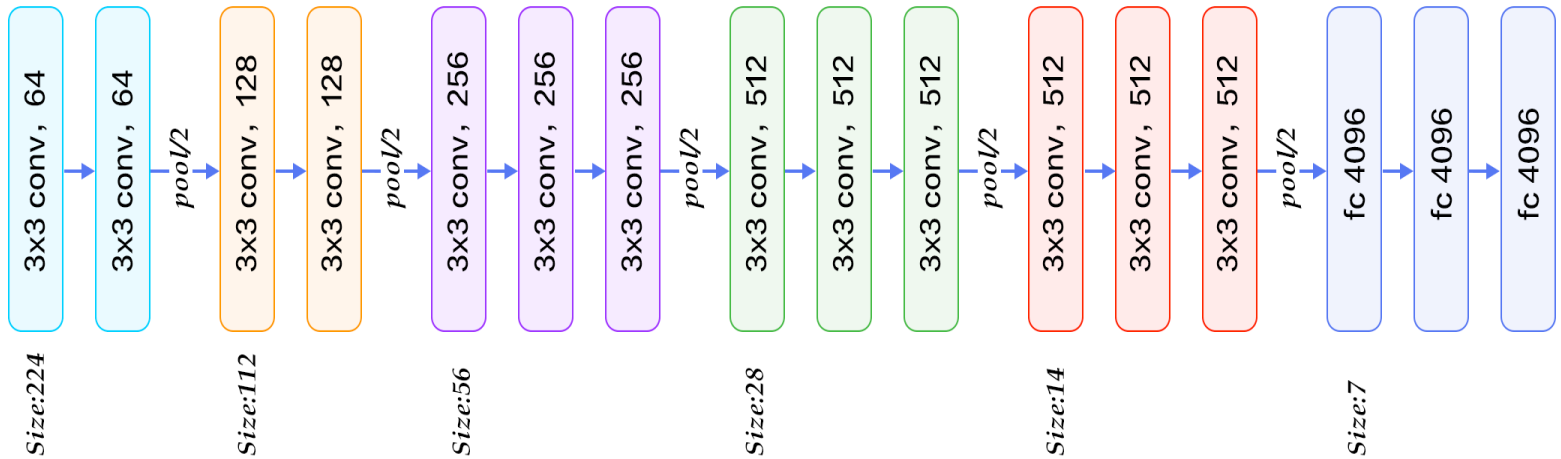
There is a convolution layer followed by a pooling layer then another convolution followed by another pulling layer and a couple of fully connected layers.



Introduction - VGG16

As it may be intuitively noticed, there are multiple types of VGGs, depending on customized configurations of the base topology.

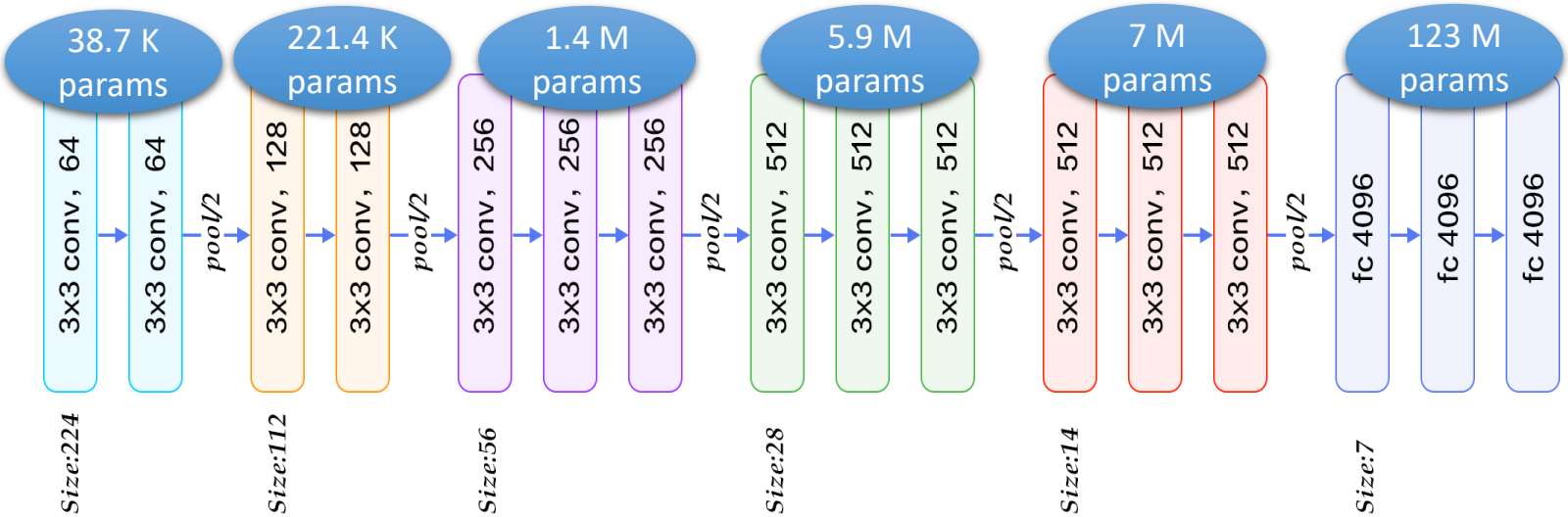
Two of the most known VGG models are VGG16, which has 16 layers and VGG19, which has 19 layers. VGG16:



Introduction - VGG16

As it may be intuitively noticed, there are multiple types of VGGs, depending on customized configurations of the base topology.

VGG16:

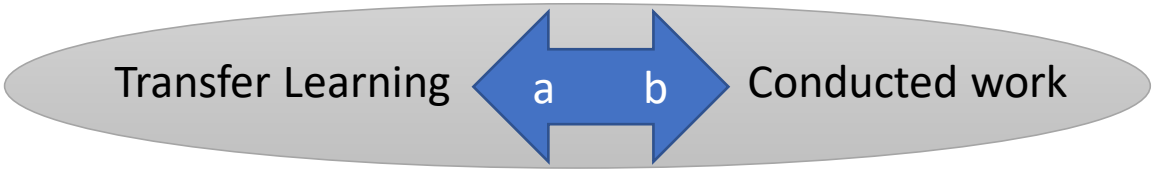


Problem statement

2 of the most significant research questions in AI:

***a**. is there a way to shorten overall development time of a deep learning model for new classes?*

***b**. is there a method to reduce the number of parameters of the deep network whilst maintaining its accuracy?*



Assume the database that needs to be used contains images of cats and dogs. However, if the VGG returns the label of a house, the engineer knows for sure that this is false if the network has not been trained on cats and dogs.

Transfer learning

One could think of a deep neural network as a combination between 2 pieces: a feature transformer and a linear model that works on those transformed features.

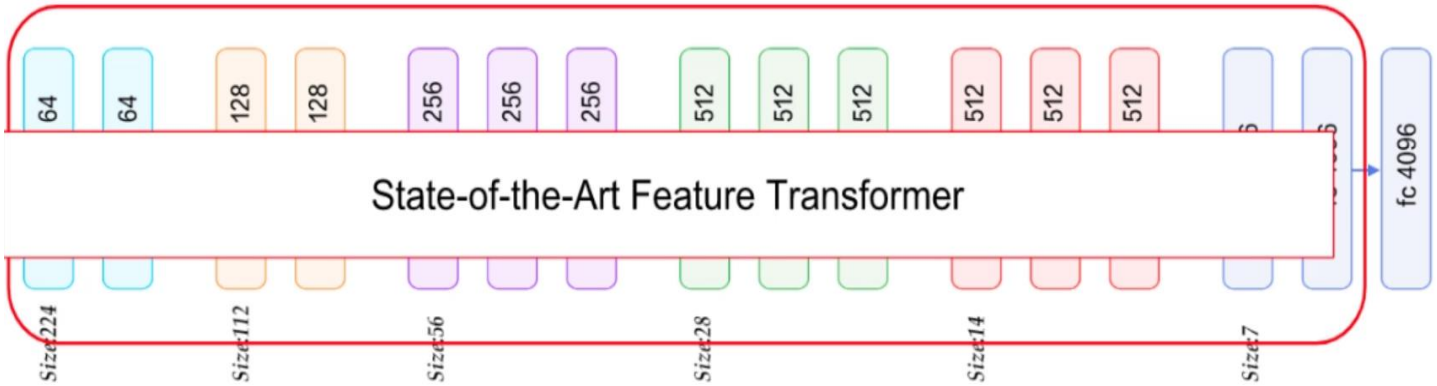
By retraining the final part of the network, i.e. the classifier, on the original data by augmenting new classes of images, the task of *transfer learning* is achieved.

In the case of VGG16, training only the final 1-3 dense layers, while keeping the feature transformer weights fixed achieves the capability of adding new classes to the output labels.

Transfer learning

The underlying assumption is that the weights of the feature transformer are highly optimized on millions of images and therefore do not need to be re-trained.

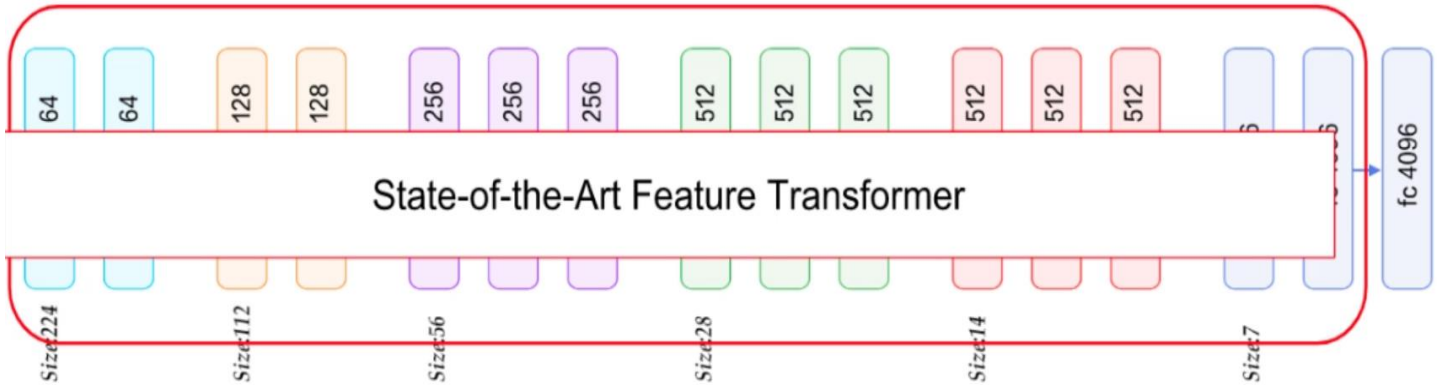
Only the final part, which contains the fully connected layers needs to be retrained



Transfer learning

The underlying assumption is that the weights of the feature transformer are highly optimized on millions of images and therefore do not need to be re-trained.

Only the final part, which contains the fully connected layers needs to be retrained

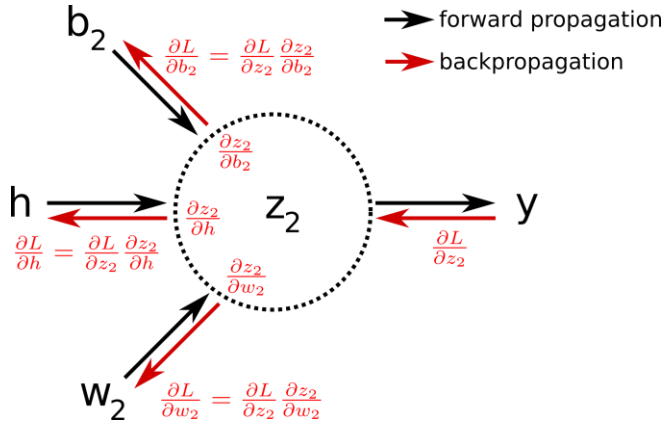


F.C. layers:
123 M params

BUT: The final part (i.e. the fully connected layers) contain a significant amount of the weights of a deep learning architecture due to the *flattening* operation

NN Training

- Deep learning's success is largely due to the ability of using the backpropagation algorithm to efficiently calculate the gradient of an objective function over each model parameter.
- There are many problems where the backpropagation algorithm is sub-optimal (*)
- Even where calculating the gradient is possible, the issue of getting stuck in a local optimum remains.



(*) https://towardsdatascience.com/the-problem-with-back-propagation-13aa84aabd71

NN Training

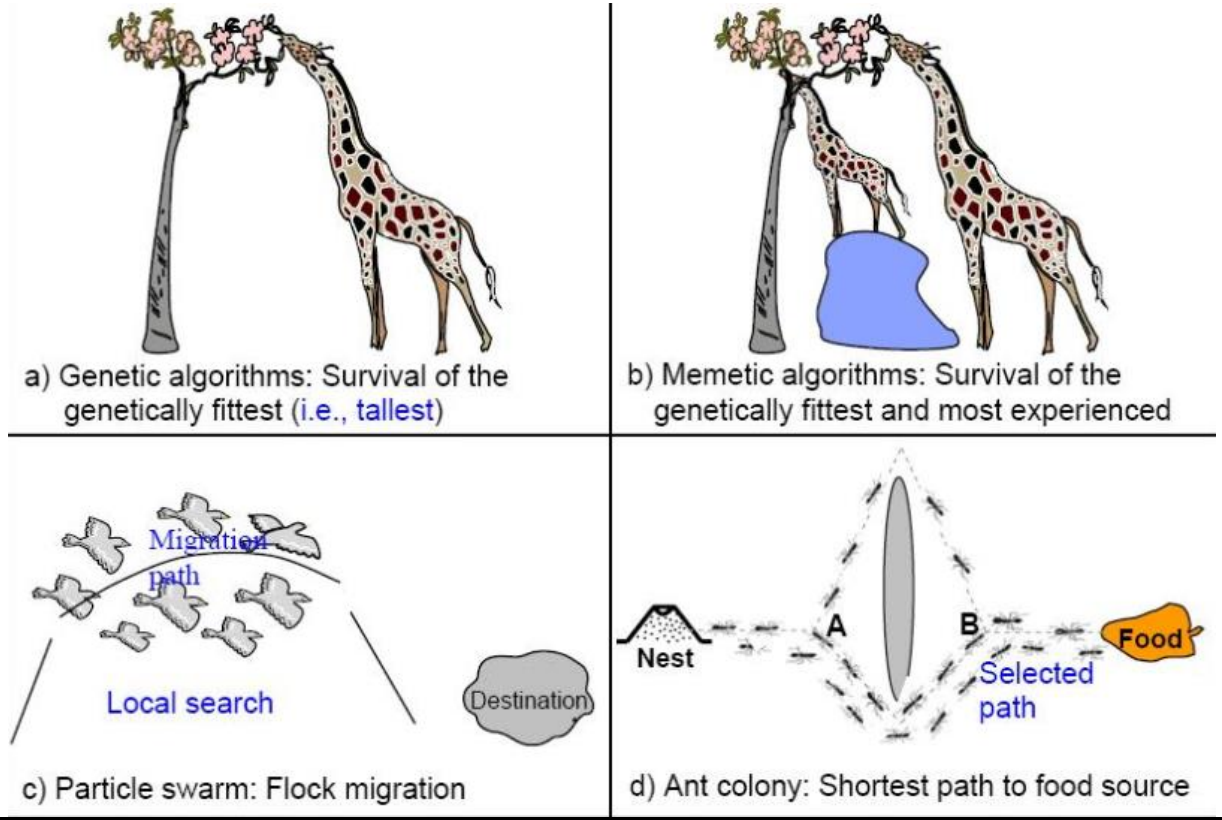
- An alternative to backpropagation is represented by evolutionary algorithms (*)
- Evolutionary computation methodologies have been applied to 3 main attributes of NN:
  - network connection weights
  - network architecture (network topology, transfer function)
  - network learning algorithms

(*) A. Lee et al, “Determination of Optimal Initial Weights of an ANN by Using the Harmony Search Algorithm: Application to Breakwater Armor Stones”, Applied Sciences, 2016

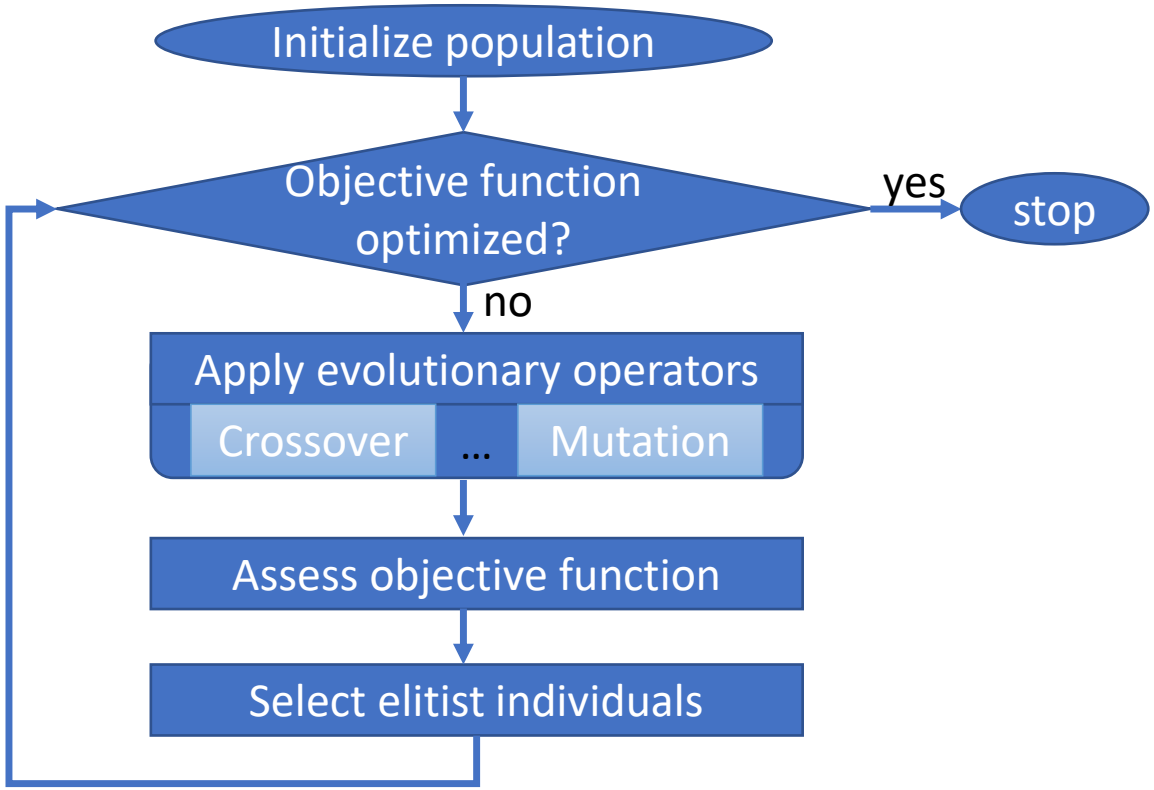
Simple Evolution Strategies

- Evolutionary algorithms are stochastic search methods that mimic natural biological evolution
- A very simple evolution strategy may sample a set of solutions from a normal distribution.
- The mean would consist of the past generation's best solution.
- The algorithm would follow the process of updating the mean and resampling for a new population over a number of generations.
- For more effective evolution strategies, populations are composed through crossover and mutation, as well recurring elite members.

Simple Evolution Strategies



Simple Evolution Strategies



Covariance-Matrix Adaptation Evolution Strategy (CMA-ES)

- CMA-ES is an algorithm that can take the results of each generation and adaptively increase or decrease the search space for the next generation.
- It does this by focusing only on a number of the best solutions in the existing population
- Mean computation in next iteration is the average of N_{best} selected points from the current iteration:

$$\mu_x^{(g+1)} = \frac{1}{N_{best}} \sum_{i=1}^{N_{best}} x_i \qquad \mu_y^{(g+1)} = \frac{1}{N_{best}} \sum_{i=1}^{N_{best}} y_i$$

- CM terms:

$$\sigma_x^{2,(g+1)} = \frac{1}{N_{best}} \sum_{i=1}^{N_{best}} (x_i - \mu_x^{(g)})^2 \qquad \sigma_y^{2,(g+1)} = \frac{1}{N_{best}} \sum_{i=1}^{N_{best}} (y_i - \mu_y^{(g)})^2 \qquad \sigma_{xy}^{(g+1)} = \frac{1}{N_{best}} \sum_{i=1}^{N_{best}} (x_i - \mu_x^{(g)})(y_i - \mu_y^{(g)})$$

- It can therefore opt to cast a larger or smaller net depending on the closeness of the best solutions to each other.
- Performance: $O(N^2)$

Particle Swarm Optimisation (PSO)

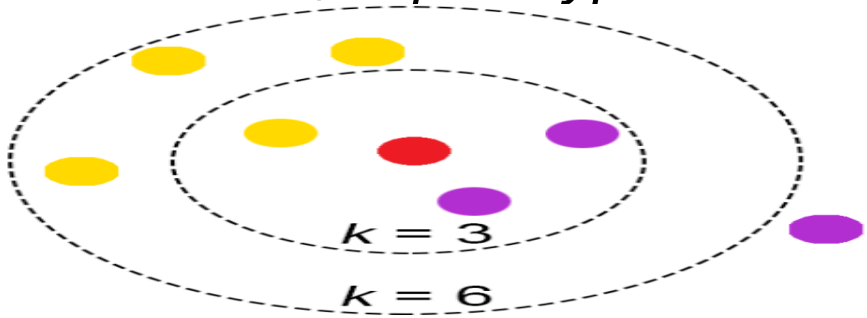
- Population based stochastic optimisation technique
- Instead of evolutionary operators, PSO uses a set of particles moving through the solution space
- The direction followed by the particles is a function of the best position found so far and the optimal particle position occupied at the current step.
- $v[] = v[] + c_1 * rand() * (p_{best}[] - present[]) + c_2 * rand() * (g_{best}[] - present[])$
- $present[] = present[] + v[]$

Evolutionary algorithms and CNNs

- Optimising for millions of parameters (in the case of a sizable NN) is highly unlikely to converge to the global minimum.
- A generic classifier can be used as a substitute to fully connected layers at the end of the network.
- Optimize subsets of convolutional kernels employing training strategies:
  - *Serial optimisation*: convolution filters are optimised individually, one at a time, while other weights remain constant.
  - *Parallel optimisation*: multiple convolutional kernels are optimized in parallel and get plugged into the network architecture when the process is finished.

k-NN Slicing

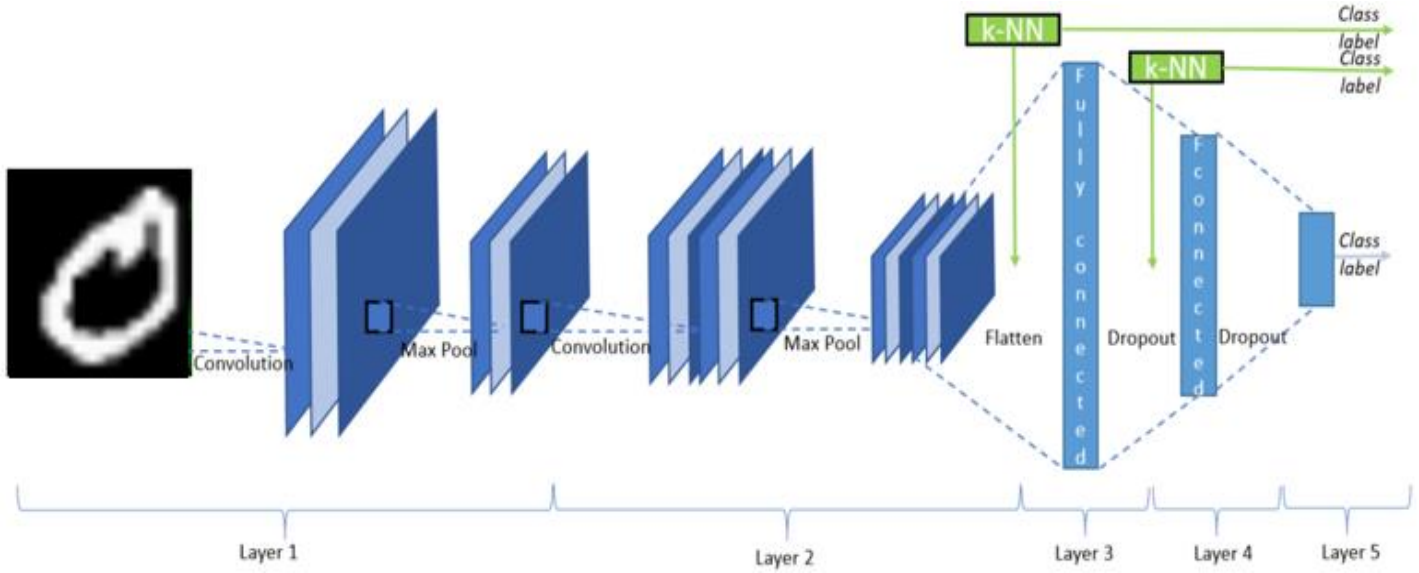
- The *k-NN (k-Neareast Neighbor)* is a classification mechanism which assigns a class label to an input feature according to the class label that the majority of a reference set, or *prototypes features* have



- The size of the reference set is denoted by *k*
- KNN is an *non parametric and lazy learning* algorithm. By non parametric, it means that it does not make any assumptions on the underlying data distribution.

k-NN Slicing

- Proposed image classification method using k-NN and CNN. Inserting a k-NN classifier into a deep network architecture to reduce its complexity. In this example, the CNN has 5 layers: 2 convolutional layers and 3 fully connected layers. The CNN – k-NN hybrid approach reduces the number of layers



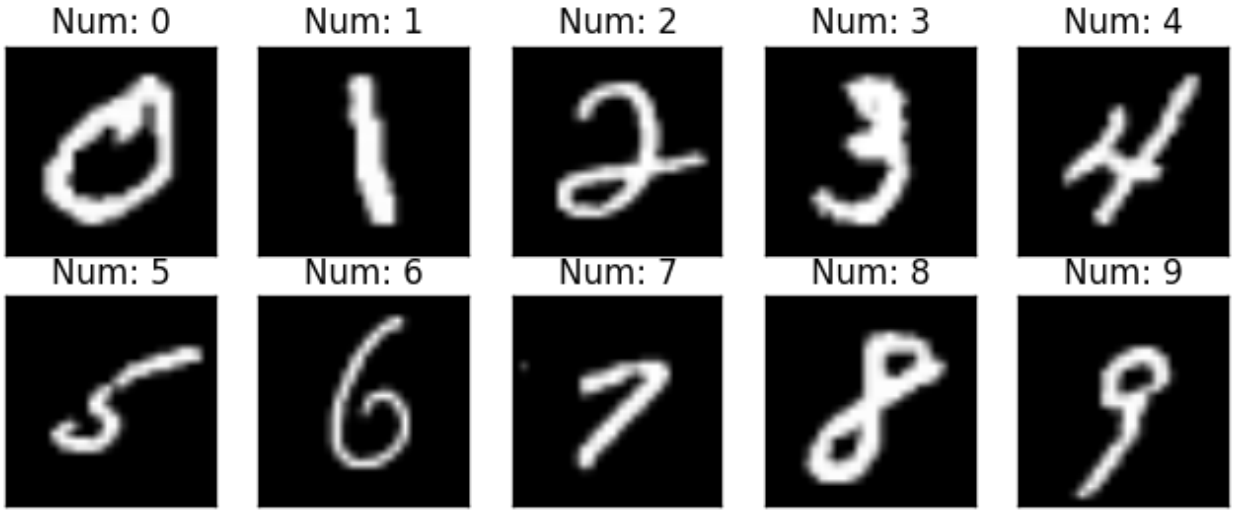
k-NN Slicing

- By employing evolutionary algorithms in fine-tuning the weights of the deep network architecture, the classical approach of back propagation approach is avoided.
- This leads to the possibility of skipping the training of the entire architecture and simply plugging-in the k-NN classifier to output the final decision.
- The genetic algorithm will create optimized features for the k-NN classifier, which are usually created using classical image processing techniques.

Experiments

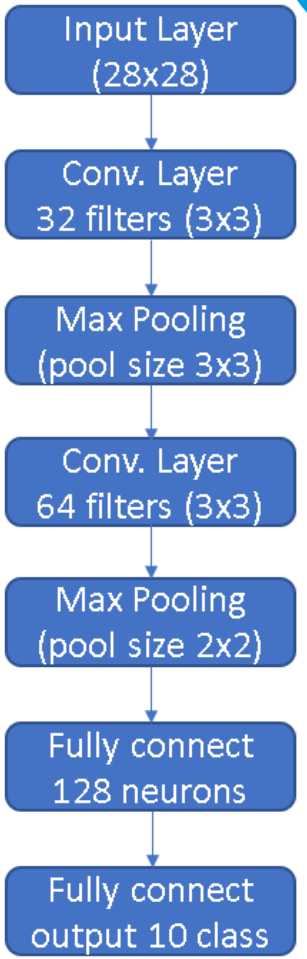
- The dataset used in the experiments is MNIST which has a training set of 60,000 examples, and a test set of 10,000 examples.

- The digits have been size-normalized and centered in a fixed-size image. One image is of size 28x28 pixels



Experiments

- The software library used for experiments is DEAP. DEAP stands for Distributed Evolutionary Algorithms in Python.
- For handling / training the convolutional neural network model, the Python Keras library with Tensorflow backend have been employed.
- The figure below details the architecture of the employed CNN architecture for classifying the MNIST dataset.



Experiments

- The architecture employed contains a total of 1.2 million parameters to be optimized.
- This number is prohibitive to be used as number of parameters in evolutionary algorithms.
- By applying k-NN slicing, the number of parameters was reduced to “only” 18,800.
- The approach employed in optimizing the 18,800 parameters is the *serial optimization*:
  - first, the kernels from the first convolutional layer are optimized.
  - Then, the optimized weights are kept fixed, whilst optimizing the weights for the second convolutional layer.

Experiments

- The term *inference engine* is used to refer to a trained neural network architecture file which holds the optimized weights.
- The table below details different setups of the inference engine by gradually reducing the number of neurons in the first fully connected layer.

Nr. Of neurons on 1st f.c. layer	No. of Params of trained network	Inference engine size
128	1.2 million	4.5 MB
64	122k	0.5 MB
32	70k	273 KB
16	44.6k	174 KB
8	31.7k	124 KB

Experiments

- The original network was trained on the graphics card, i.e. nVidia GTX 1050 Ti, with 768 CUDA cores. The training time for the network on 5 epochs is only 45 seconds.
- The evolutionary optimization was conducted on an Intel i5 9600k CPU with 6 cores running at 4.7 GHz.
- The optimization time is roughly 24 hours for ES with CMA and 12 hours for PSO optimization.
- By applying the proposed k-NN slicing procedure (k=9), the inference engine size is only 74 KB, which is
  - about 60 times smaller than the original network model with 128 neurons on the first fully connected layer and
  - about 7 times smaller than the network model with 64 neurons on the first fully connected layer.

Experiments

- The classification accuracy for different setups is summarized in the table below:

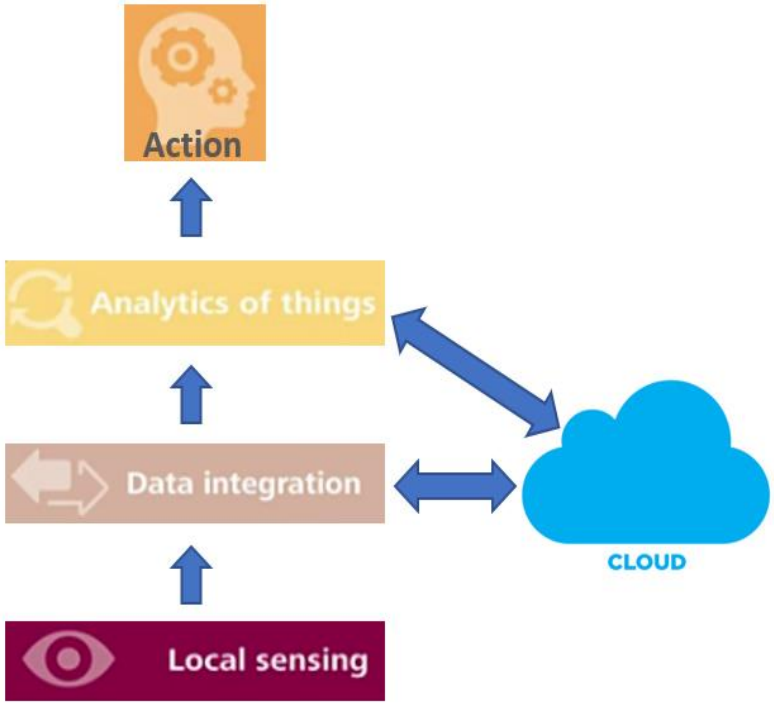
No. Params in network	Accuracy for Train/Test of 4000/1000 5 epochs	Accuracy for Train/Test of 60000/10000 5 epoch	Accuracy for Train/Test of 4000/1000 20 epoch
1.2 million	97.5	99.91	98
122k	97	98.96	97.5
70k	97	98.98	97.5
44.6k	96.1	98.65	97.4
31.7k	87.5	97.8	96.5
18.8k CMA-ES	97.5	98.57	97.5
18.8k PSO	97.3	98.52	97.3

Towards AI on IoT devices

- *Ubiquitous computing* is generally defined as a computing environment where computing systems weave themselves in various aspects of everyday life and vanish into the background.

- *Edge computing* implies the generation, collection and analysis of data at the site where data generation takes place

Towards AI on IoT devices



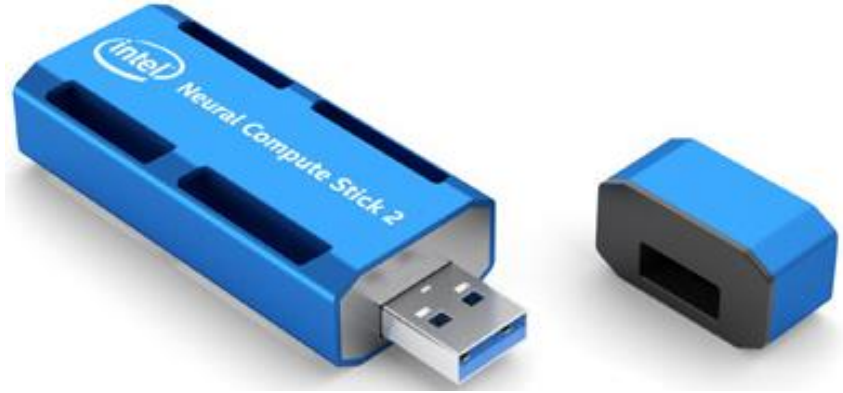
Edge computing information flow

Towards AI on IoT devices

nVidia Jetson Nano is equipped with a quad-core 64-bit ARM CPU and a 128-core integrated nVidia GPU. It also includes 4GB LPDDR4 memory in an efficient, low-power package with 5W/10W power modes and 5V DC input, as shown in figure below.

Towards AI on IoT devices

Intel introduced in 2018 an USB device called Movidius Neural Compute Stick, version 2 which can be attached to any computer or embedded board, such as Raspberry Pi to bring hardware AI capabilities to an existing device.



In progress: comparing the running speed of the original neural network architecture on an Intel Neural Compute Stick 2 vs k-NN slicing approach on generic laptop

Conclusions

- The present work was focused on enhancing the usability of CNNs by eliminating the fully connected layers and training a generic classifier, such as k-NN on the activation maps of convolutional layers
- ES with CMA and PSO algorithms were tested
- The experimental results show that the obtained accuracy of reduced-size network optimized with the evolutionary algorithms is comparable to the classification accuracy obtained by training the full-size CNN using the classical backpropagation approach.
- Future work: compare running speed of the full inference engine on Intel NCS2 vs “optimized” inference engine on generic laptop