Czech University of Life Sciences Prague

Faculty of Economics and Management

Department of Information Technologies

Diploma Thesis

Implement classification for traffic signs using convolutional neural network

Mohamad Abdulrahman

CZECH UNIVERSITY OF LIFE SCIENCES PRAGUE

Faculty of Economics and Management

DIPLOMA THESIS ASSIGNMENT

Mohamad Abdulrahman

Systems Engineering and Informatics
Informatics

Thesis title

implement classification for traffic signs using convolutional neural network

---

Objectives of thesis

I will give an overview of the areas in which convolutional networks have been successfully used, then i will choose a specific problem for which there is publicly available data and proposes a method for its solution. then I will implement this method and verify it on data using the Tensorflow and Keras libraries.

Methodology

I will describe the current state of research in the field of convolutional neural networks. I will give an overview of the areas in which convolutional networks have been used successfully and describe some typical applications. also, I will describe open-source software designed for their implementation, especially the frameworks Tensorflow and Keras. I will use (GTSRB) dataset and implement classification by convolutional neural network using Tensorflow and Keras and then verify the results using a laptop camera.

The proposed extent of the thesis

60 pages

Keywords

Deep learning, convolutional Neural Network, Classification, ILSVRC

Recommended information sources

GÉRON, A. *Hands-on machine learning with Scikit-Learn and TensorFlow : concepts, tools, and techniques to build intelligent systems.* Beijing ; Boston ; Farnham ; Sebastopol ; Tokyo: O'Reilly, 2019. ISBN 978-1-492-03264-9.

Expected date of thesis defence

2020/21 SS – FEM

The Diploma Thesis Supervisor

doc. Ing. Arnošt Veselý, CSc.

Supervising department

Department of Information Engineering

Electronic approval: 30. 3. 2021	Electronic approval: 30. 3. 2021
Ing. Martin Pelikán, Ph.D.	**Ing. Martin Pelikán, Ph.D.**
Head of department	Dean

Prague on 30. 03. 2021

Declaration

I declare that I have worked on my diploma thesis titled " implement classification for traffic signs using convolutional neural network" by myself and I have used only the sources mentioned at the end of the thesis. As the author of the diploma thesis, I declare that the thesis does not break copyrights of any other person.

In Prague on 31/03/2021 ______________________

Acknowledgment

I would like to thank **doc. Ing. Arnošt Veselý, CSc.**, **my family** and **my friends** for the support throughout this project.

Abstract

The ability to classify traffic signs quickly and effectively is great importance for autonomous cars or as a driver's assistance in some difficult circumstances. many techniques in machine learning were used like Support Vector Machine and Random Forest, on the other hand deep learning techniques reached state of art performance in many image classifications tasks.

Convolutional neural networks have proven to be highly capable in many areas of deep learning, particularly in the field of image classification. As this work provides an overview of the most important architectures and techniques, what are the areas other than image classification, and which open-source programs are most used with these architectures.

The ability of CNN to treat with real experiments is discussed. and real tests to check the performance of the proposed model is implemented by using an external camera as a test tool with modifying parameters of CNN model to achieve the highest possible accuracy value for prediction process with lowest possible training time.

Keywords: Convolutional Neural Network, Deep Learning, ILSVRC

Contents

1. Introduction

The concept of intelligent machines began to appear in the early twentieth century, and the first person to use the word robot to describe these new concepts was the Czech writer (Karel Čapek) in 1920.

And this concept began to crystallize in the fifties of the twentieth century, and the question was: Why cannot machines use the available information and look at the causes of things happening as humans do to solve their daily problems and make decisions, and this was what was included in the research paper of (Alan Turing) in 1950.

The field of artificial intelligence has gone through many successes and setbacks, as computers were not as efficient as we know them today and there was a need to prove the importance of this field to convince the financiers.

With the continued attempts, artificial intelligence aroused in the 90's of the twentieth century. Perhaps the most prominent successes were the defeat of the world champion in chess by the computer program Deep Blue designed by IBM. Successes continue to follow at various levels to this day and artificial intelligence is used in all fields of science [3].

Ongoing research and the demands of emerging life have resulted in the emergence of more in-depth concepts such as machine learning and deep learning. Computer vision was one of the important areas to delve into, until we got to the convolutional neural networks which achieved impressive results, where the development of computing capabilities and the availability of large amounts of data were a major role in its development.

2. Objectives and methodology

2.1 Objectives

I will give an overview of the areas in which convolutional networks have been successfully used, then I will choose a specific problem for which there is publicly available data and proposes a method for its solution. then I will implement this method and verify it on data using the Tensorflow and Keras libraries.

2.2 methodology

I will describe the current state of research in the field of convolutional neural networks. I will give an overview of the areas in which convolutional networks have been used successfully and describe some typical applications. also, I will describe open-source software designed for their implementation, especially the frameworks Tensorflow and Keras. I will use (GTSRB) dataset and implement classification by convolutional neural network using Tensorflow and Keras and then verify the results using a laptop camera.

3. Literature review

3.1 Artificial Intelligence

The applications of artificial intelligence are many, and they are present in many different sectors and industries. In health care, playing chess and self-driving cars. Each of these apps should take into account whatever action it takes, as this will affect the end result. It is also used in applications related to the financial industry and banking where it may reveal unusual use of payment cards.

Perhaps there are several definitions of artificial intelligence, there may be several definitions of artificial intelligence, as this relates to the field in which AI is perceived and some have argued that artificial intelligence should be defined in terms of its goals, as Russell and Norvig mentioned in their texts [5]

where the definition was divided into four quarters.

- Systems that think like humans
- Systems that behave like humans
- Systems that think rationally
- Systems that act rationally

There are three terms that overlap with each other, and artificial intelligence forms the general framework for them Figure 1. [6].

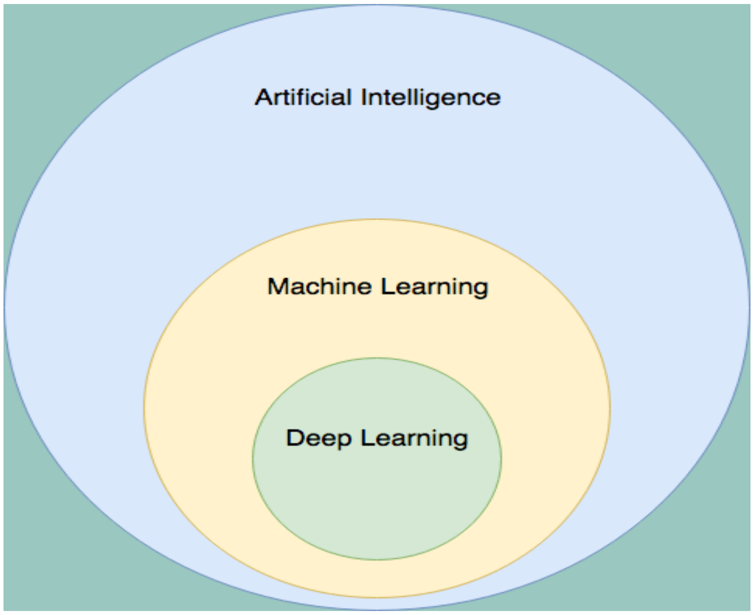


Figure 1. AI, ML and deep learning, source [6]

3.1.1 Biological side

The living things that surround us and their ways of life and their adaptation to their environmental surroundings have always been a source of inspiration for many scientists. Perhaps the most notable simulation of these living creatures was aircraft design. In the field of computing the nervous system had a great impact. As the attempts to produce systems and programs that possess the same features began more than 70 years ago.

Work began on developing these systems to simulate the analytical ability of living systems, and of course the quest to add the feature of learning from errors and benefiting from the mistakes of others, and this is what leads us to the term machine learning.

3.1.2 Source of inspiration for neural networks

To understand an artificial neuron, we will look at a biological neuron. In general, a biological neuron consists of a cell body, from which many dendrites emerge, in addition to a main axon, which divides near its end into branches that have synapses at the end.

Synapses, in turn, connect to dendrites coming from other cells or directly to the body of other neurons. These nerve cells communicate by means of electrical signals called neurotransmitters [1, 2], this brings us to one of the earliest and simplest models of artificial neural networks which is Perceptron

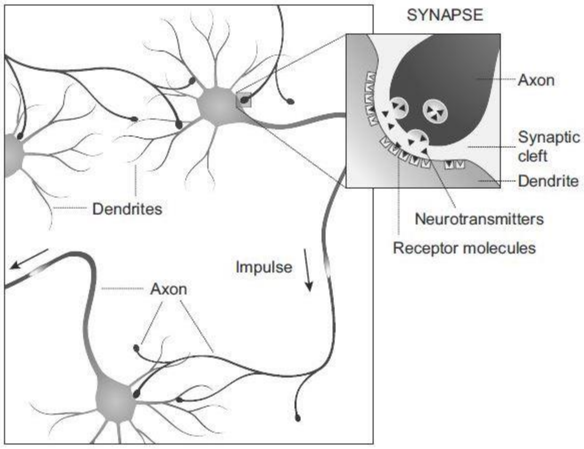


Figure 2. Biological neurons with detail of synapse, source [1]

3.1.3 The Perceptron

It was invented by Frank Rosenblatt in 1957, It is one of the simplest architectures of an artificial neural network and an essential entry point to understanding these networks. It mainly depends on the so-called (LTU) linear threshold unit, each input (x) has a weight (w) that increases or decreases the importance or impact of this input, and then the inputs and weights are calculated according to the following equation: $Z = W_1X_1 + W_2X_2 + \cdots + W_nX_n$

then the step function is applied, and we have bias neuron (x_0) which outputs always (1) value.

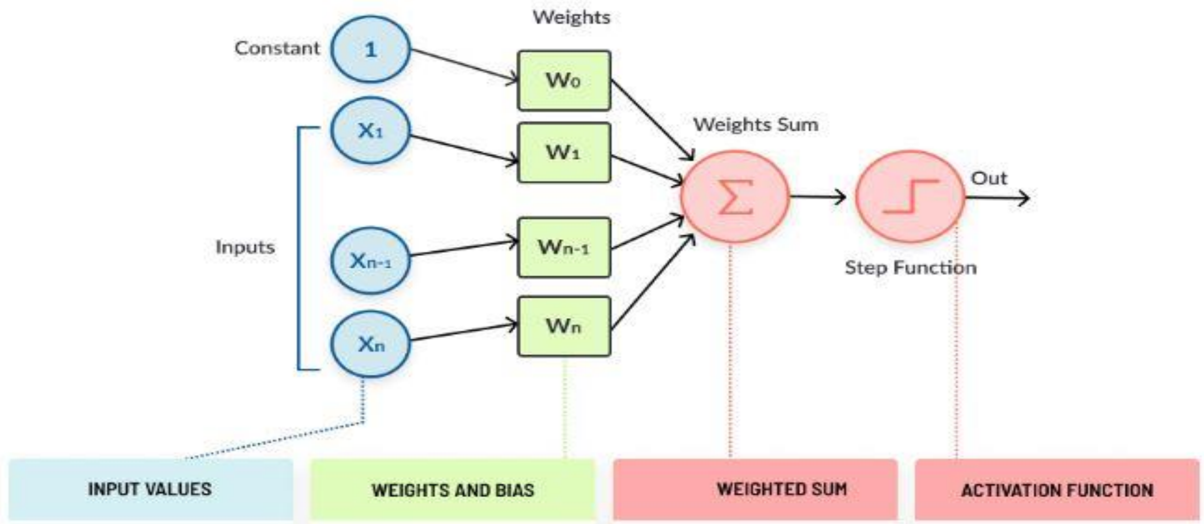


Figure 3. The Perceptron, source [9]

The most used step function is Heaviside step function (i) and sometimes the Sign function (ii)

heaviside (z) = $\begin{cases} 0\ if\ z < 0 \\ 1\ if\ z\ \geq 0 \end{cases}$ (i) sgn (z) = $\begin{cases} -1\ if\ z < 0 \\ 0\ \ if\ z = 0 \\ 1\ \ if\ z > 0 \end{cases}$ (ii)

This perceptron can be used for a very simple linear binary classification, i.e. it gives either positive or negative class like what logistic regression does.

But this perceptron stands incapable of solving simple problems like XOR, this problem can be solved by multi-layer perceptron where we have one or more non-linear layers called the hidden layers, where all the neurons in a layer are connected to every neuron in the previous layer, it is called denes layer. [4] , unlike single Perceptron, multi-layer Perceptron has Capability to learn non-linear models Figure 4.

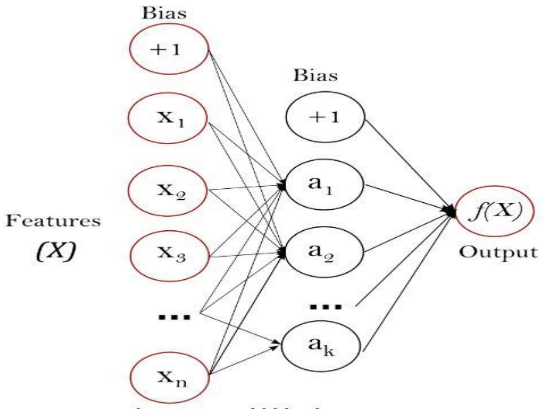


Figure 4. multi-layer Perceptron, source [4]

[x] represents input features.

[a_1, a_2,…, a_k] represents the values from the previous layer with a weighted linear summation

In the output layer we have non-linear activation function such as the hyperbolic tan function.[4]

3.2 Machine learning

Machine Learning is field of study that gives computers the ability to learn without being explicitly programmed, Arthur Samuel [1959]

With the rapid development in the field of computing we can find that machine learning is very useful in following cases:

- Problems requiring a lot of hand tuning as a simple algorithm can do the job perfectly.[2]
- Sometimes there are problems that cannot be solved by traditional methods
- When there is need to process large amounts of data in less time

where we can train the algorithm using specific data and find the relationship between input and output, and upon completion of this process, the algorithm is able to guess the results when using new data. There are several concepts to explain types of machine learnings, Perhaps the most common classifications are related to human supervision

There are four main categories

1. Supervised, which will be used in practical part.

In supervised learning we feed the training data to the algorithm includes the desired solution which called labels like in spam filter in e-mail systems. The main algorithms used in supervised learning are:

- K -nearest Neighbors.
- Linear regression.
- Logistic regression.
- Support vector machine.
- Decision trees and random forests.
- Neural networks (it could be unsupervised or semi supervised in some applications).

2. Unsupervised
3. Semi supervised
4. Reinforcement learning

3.3 Deep learning

Deep learning is a subfield of machine learning and both are within the field of AI, and deep learning theory was first developed in the 1980s

Deep learning is a method of machine learning that teaches computers to do what is normal for humans. for example, Deep learning is the fundamental technology behind self-driving cars, enabling them to recognize traffic signs and traffic lights, or distinguish pedestrians from roadside poles and other elements. And perform classification tasks directly from images, text, or audio. Deep learning models can achieve advanced accuracy, and sometimes exceed human performance. This would not have been possible without the great development in computing and storage technologies because it requires large amounts of labeled data and great computing power. In addition to high-performance GPUs, when all combined with cloud computing, this reduces the training time for a deep learning network from weeks to hours or less. [7]

The term "deep" refers to the number of hidden layers in a neural network where deep networks can contain more than a hundred layers, to be distinguished from traditional neural networks (shallow networks) which consist of only 1 or 2 hidden layers at most, Figure 5.[8]

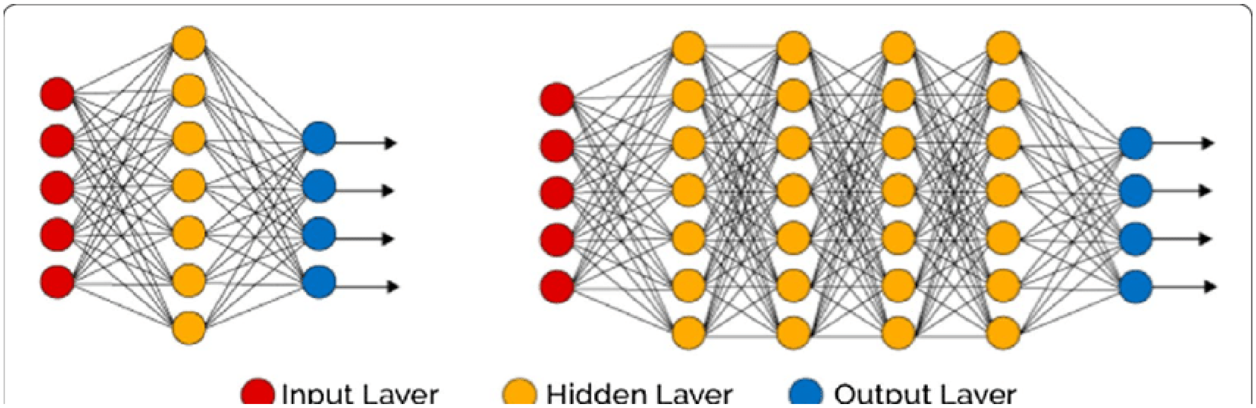


Figure 5. shallow and deep networks, source [8]

3.3.1 Reasons to use deep learning

For a while the ideas of deep learning (deep neural networks) have been around. But it began to develop rapidly because of:

1- Availability and large size of data: As a result of the massive proliferation of digital devices in all areas of life and the expansion of the Internet, this generates huge amounts of data that are supplied or fed to learning algorithms.

2- Computational scale : if we collect a lot of data, we can figure out that the performance of traditional learning algorithms, such as logistic regression will “flattens out,” and the algorithm stops improving even if you fed it with more data ,on the other hand If we feed the same data to a small, then medium, and then large(in term of number of hidden units/layers/parameters) neural network, we will notice that the performance improves as the size of the neural network increases, figure.6

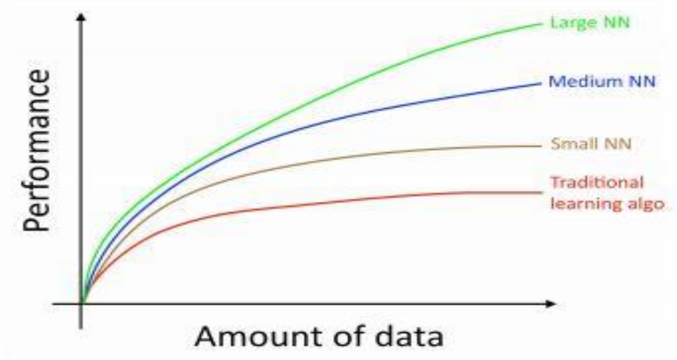


Figure 6. impact of size of neural network on performance, source [10]

Artificial neural networks are the backbone of deep learning and are ideal for dealing with complex problems such as classifying very large numbers of images and speech recognition.

In this work I will speak about famous architecture of DNN which is CNN, this architecture managed to achieve superhuman performance on some complex visual tasks

3.4 Convolutional neural network

3.4.1 CNN vs ANN

The major difference between a traditional Artificial Neural Network (ANN) and CNN is that only the last layer of a CNN is fully connected whereas in ANN, each neuron is connected to every other neuron [55].

ANNs are not suitable for handling images, although this works fine for small images, but it will not be suitable to deal with large images because of the large number of parameters generated. For example, if we have 200 × 200 image, which means we have 40000 pixel and if the first layer has only 500 neurons, this means a total of 10000000 connection and we are still in the first layer. On the other hand, CNN can deal with this using partially connected layers and weight sharing.

3.4.2 Main concepts used in CNN architectures

convolution layers: The most important building block of a CNN where the neurons in the convolution layers are not connected to every single pixel in the previous layer but only to pixels in their receptive fields which defined by filter size figure [2]

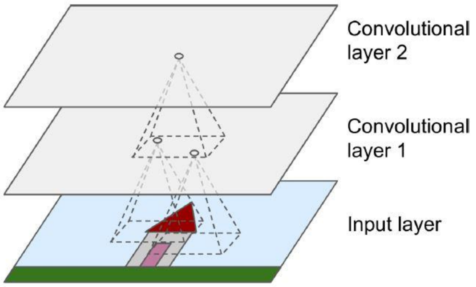


Figure 7. CNN layers with rectangular local receptive fields, source [2]

Pooling layer: The function of this layer is subsampling to prevent the risk of overfitting and the connection with previous layer like convolution layer, and we must define the size, stride, and the padding of receptive field in the previous layer, and all it does is aggregate the inputs using an aggregation function such as the max or mean. most common type of pooling layer is max pooling layer.[2]

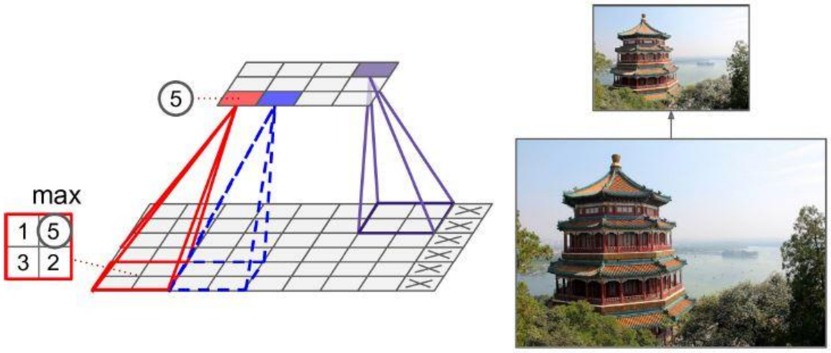


Figure 8. Max pooling layer (2 × 2 pooling kernel, stride 2, no padding), source [2]

Fully connected layer: all the neurons in a layer are connected to every neuron in the previous layer, also called denes layer.

Filters (convolution kernels): The filter is used to extract information from the previous layer, and it is often square dimensions like (3 × 3)

Padding: It is the process of adding a frame of pixels to an image, there are many types of padding:

- same padding or zero padding: we add new zero pixels and the output has the same shape of input
- constant padding: we add constant value (non-zero), output has the same shape of input

Stride: The number of pixels the filter shifts through in a single step

Activation functions:

- ReLU: Rectified Linear Unit function

- Sigmoid
- Tanh: hyperbolic tangent function

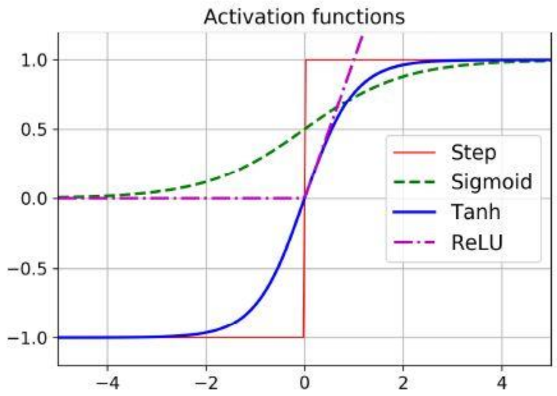


Figure 9. Nonlinear activation functions, source [2]

3.4.3 Stages of development of Convolutional Neural Networks

It began to be used since the eighties of the last century, but it began to develop rapidly after the tremendous technical development in the 2000s.

Between 2010 and 2017, a big competition appeared in all universities in the world. the competition used ImageNet data repository to improve the algorithms of CNN. Since then, literature has worked both in designing more accurate networks as well as in designing more efficient networks from a computational-cost point of view.

The competition called "The ImageNet Large Scale Visual Recognition Challenge (ILSVRC)". This had a huge impact on CNN's development

I will mention the stages of development of CNN in the chronological order in which they appeared and especially the ones that are most used and that have achieved state of art.

3.4.3.1 LeNet-5

LeNet-5 is the ancestor of convolutional neural networks and one of the most fundamental deep learning models that is primarily used to classify handwritten digits (it used MNIST dataset) and one of the earliest neural networks that use the convolution operation. Combining back-propagation algorithms with CNN It became at that time the most advanced in classifying images using deep learning.

The LeNet-5 network was shown to incur an error rate less than 1% on test data which are very close to state of art.

The layers of the LeNet-5 architecture as shown in Figure 10. [11]:

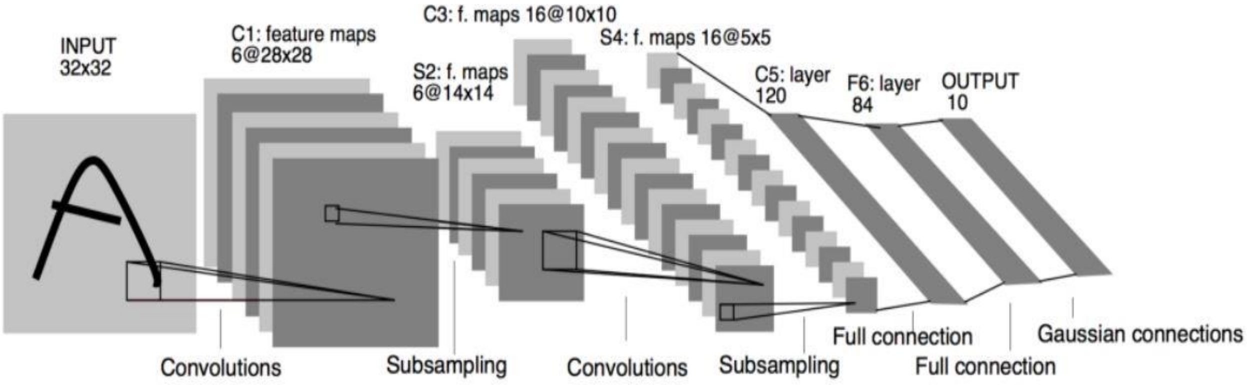


Figure 10. LeNet-5 architecture, source [11]

It consists of seven layers without input layer: 3 convolutional layers (C1,C3,C5) and two subsampling layers (C2,C4) , fully connected layer (F6) and the output layer [12]

C1: First convolution layer: (num_kernels=6, kernel_size=5×5, padding=0, stride=1)

S2: First subsampling Layer (kernel_size=2×2, padding=0, stride=2)

C3: Second Convolution Layer (num_kernels=16, kernel_size=5×5, padding=0, stride=1)

S4: Second subsampling layer (kernel_size=2×2, padding=0, stride=2)

F5: Fully Connected Layer (out_features=120)

F6: Fully Connected Layer (out_features=84)

F7: Fully Connected Layer (out_features=10)

To clarify more

1- The input is a single-channel image, (28*28*1) or (32*32*1) pixel .
2- The output of convolutional layers and fully connected layers pass through tanh activation function.
3- The output of subsampling (pooling) layers passes through Sigmoid activation function.
4- the SoftMax activation function is applied on output layer and it’s contains the predictions of the network
5- we have total of 60,850 trainable parameters and 340,918 connections overall.
6- Classify images in 1 of 10 classes, every class represents one value from 0 to 9

We can realize that the high and the width goes down while the number of channels (number of connections) goes up as we go deep in the network

Limitation of leNet-5

1- this network works only with one channel images (gray scale image) which limited its applications.
2- modern approach is to implement a max-pooling operation instead of average pooling operation.
3- this network has only 60 thousand parameters while nowadays we use from 10 to 100 million parameter.
4- Modern architectures use ReLu activation instead of tanh and sigmoid which used in LeNet-5, as ReLu usually leads to higher classification accuracies

3.4.3.2 AlexNet

In 2012 Alexnet architecture which was primarily designed by Alex Krizhevsky won the competition of ILSVRV, we can say it is a Classic type of Convolutional Neural Network, Figure 11.

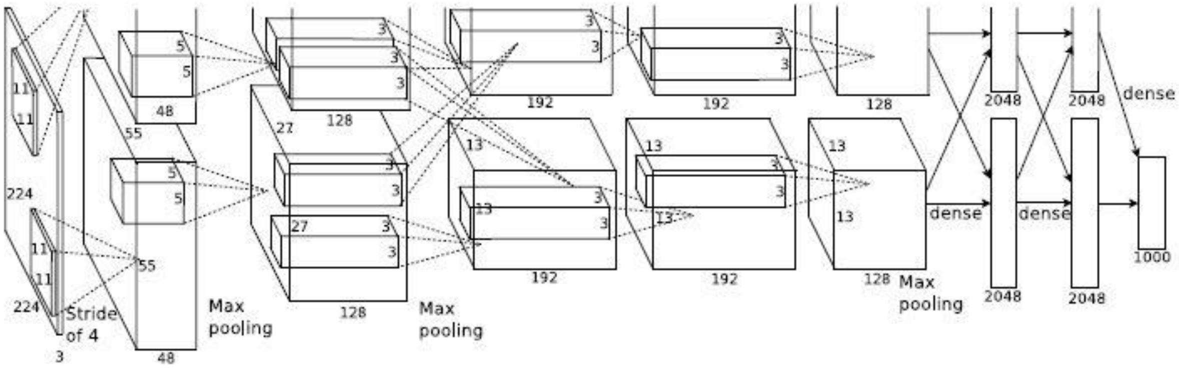


Figure 11. AlexNet architecture, source [14]

AlexNet architecture has a lot of similarity to LeNet-5, but much bigger.

This architecture has 8 layers without input layer , the first five layers are convolutional layers and the remaining 3 are fully connected layers.

Input layer fed with (227*227*3) image

First convolution layer: (num_filters=96, filter_size=11*11*3, stride = 4), Then applying overlapping Max-pooling and response normalization

Second convolution layer: (num_filters=256, filter_size=5*5*48), Then applying overlapping Max-pooling and response normalization

Third convolution layer: (num_filters=384, filter_size=3*3*256)

fourth convolution layer: (num_filters=384, filter_size=3*3*192)

fifth convolution layer: (num_filters=256, filter_size=3*3*192)

the last three fully connected layers have 4096 neurons each

note that the third, fourth and the fifth convolution layers are connected to each other without any Max-pooling or normalization

The ReLU is applied to the output of every convolutional and connected layer.

The output is fed to a 1000-way SoftMax which produce a distribution over the 1000 class label.

Overfitting problem:

To reduce overfitting which is caused by large number of inputs (60 M), so the (Data Augmentation) and (Dropout) was used.

- Data Augmentation: increases the size of the training set by generating many realistic variants of each training instance. The is done by randomly changing the lighting conditions or shifting or rotating the image to different degrees, noting that applying augmentation to some images completely change their meaning.
- Dropout: it consists of setting to zero the output of each hidden neuron with probability 0.5. The neurons which are “dropped out” in this way do not contribute to the forward pass and do not participate in backpropagation [14]
-

Advanced of AlexNet

1- It has 60 million parameters, as it used ImageNet dataset, so we can see the big size of data helped the algorithm to reach a remarkable performance.
2- This architecture made much better than LeNet-5 because of using ReLU activation function and the increase in depth of the network
3- In 2012 GPUs were still slower in comparing with nowadays, so the training done by using 2 GPU and some layers were split across these tow GPU, but it was the first GPU based CNN model

Limitations:

1- It’s not deep enough, so it struggles to learn features from image set.
2- It takes more time to reach higher accuracy comparing to later models.

3.4.3.3 VGG-16

In 2014 Karen Simonyan and Andrew Zisserman [15] published their paper (very deep convolutional networks for large-scale image recognition), the architecture was known as VGG-16, Figure 12. This model achieved 92.7% top-5 test accuracy in ImageNet, which is a dataset of over 14 million images belonging to 1000 classes.

This model is one of the most popular models in the research community due to its simple approach and because pre-trained weights are freely available, making it easier to fine-tune this powerful model to new tasks.

Instead of having so many hyper parameters, they used much simpler network and they focused on having conv layers with 3*3 filters and layers always use SAME padding, this means the output layer will have the same spatial dimension of input layer.

All Max-pooling layers is 2*2 with stride of 2 [15]

It uses ReLU activation function on all stages and SoftMax activation on output

VGG16 was trained for weeks and was using NVIDIA Titan Black GPU’s. [16]

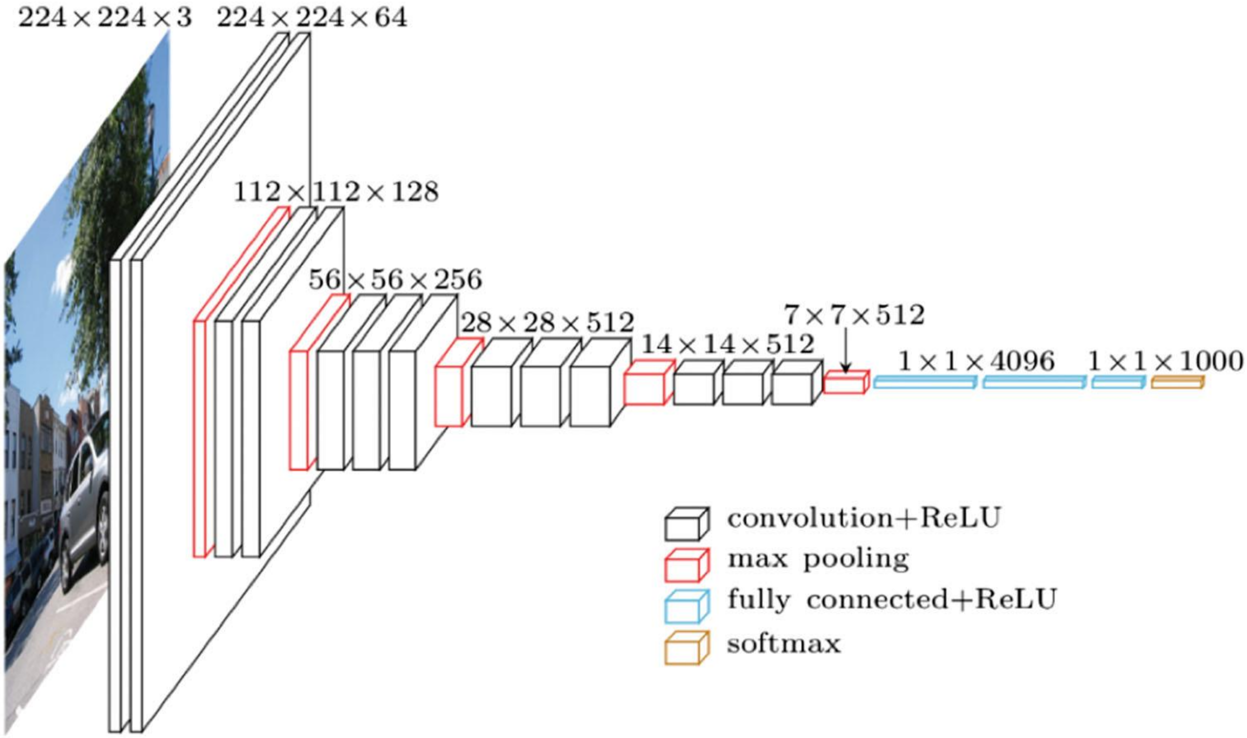


Figure 12. VGG-16 architecture, source [16]

Limitations

this model has so many weight parameters, about 138 million parameters (more than 500 MB of weight size), this means a long training time and expensive cost of computation and will face difficulties when it is deployed on low resource systems.

3.4.3.4 Inception-v1

2014 winner model of Image classification competition (ILSVRC), they chose GoogLeNet as their team-name [17]. It uses 12× fewer parameters than the architecture of AlexNet [14], but it is more accurate.

It is 27 layers deep. This model includes two techniques from [18]:

1- Fully connected layers are replaced with global average pooling.
2- It uses 1×1 convolution, which can help to reduce model size which in turn can also somehow help to reduce the overfitting problem.

Since pooling was essential to reach top results, the suggestion was to add a parallel pooling path at each stage, this gives additional beneficial effect, Figure 13. [17]

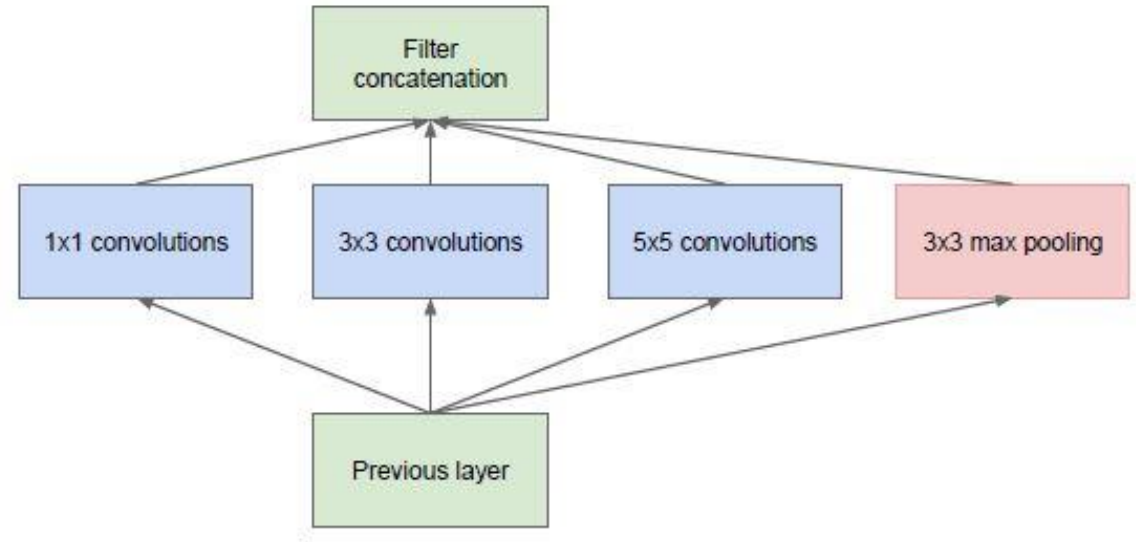


Figure 13. inception module, source [17]

Problem:

In this module (Figure 13), there was one big problem, that will be a significant and inevitable increase in transition from one stage to another due to the merging of the outputs of convolutional layers with the output of the pooling layer, Which leads to the computational problems within a few stages (high computational cost)

Solution:

[17] The solution was by applying dimension reductions and projections and, 1×1 convolution is used to compute reductions before the expensive 3×3 and 5×5 convolutions. Also, they included the use of rectified linear activation and the result as in Figure 13.

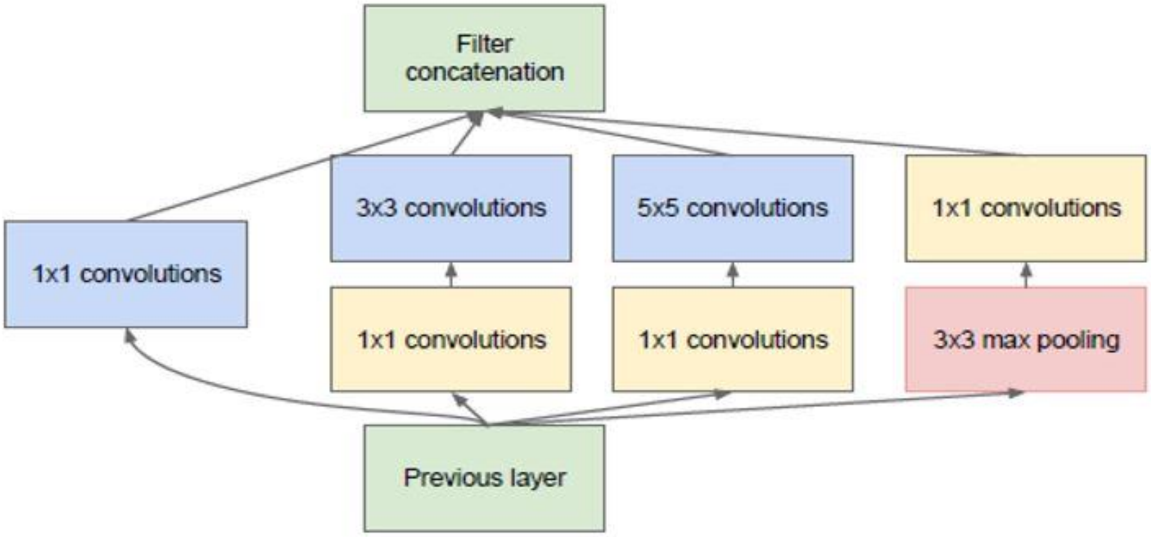


Figure 14. Inception module with dimensionality reduction, source [17]

Inception module is an image model block whose aim is to approximate an optimal local sparse structure. Simply put, it allows us to use several types of filter sizes in a single image block, rather than being limited to a single filter size.

This module repeated many times through network and the research continued to reach other architectures like in Inception-v2 and v3 and v4

3.4.3.5 Inception-v2 & v3

It was necessary to find solutions to reduce the complexity of the structure and increase the accuracy and the idea was that performance is better when the dimensions of the inputs are not significantly altered by the convolutional networks which in turn leads to loss of information known as a “representational bottleneck”

Improvements in v2

1- replace 5×5 convolution by two 3×3 convolution operations which leads to increase in architecture performance
2- replace convolutions of filter size (n × n) by a combination of 1×n and n×1 convolutions (3×3 became (1×3) followed by (3×1) which is 33% cheaper in terms of computational complexity in comparison with 3×3.

3- The filter banks in the module were expanded (made wider instead of deeper) to remove the representational bottleneck. [19][20]

Improvements in v3

The architecture almost the same in V2 with some changes

1- Use of RMSprop optimizer to decrease the step for large gradients to avoid exploding, and to increase the step for small gradients to avoid vanishing.[21]
2- Batch Normalization in the Auxiliary Classifiers.[22]
3- Use of 7×7 factorized Convolution. Figure 15.
4- Label Smoothing Regularization to prevent overfitting

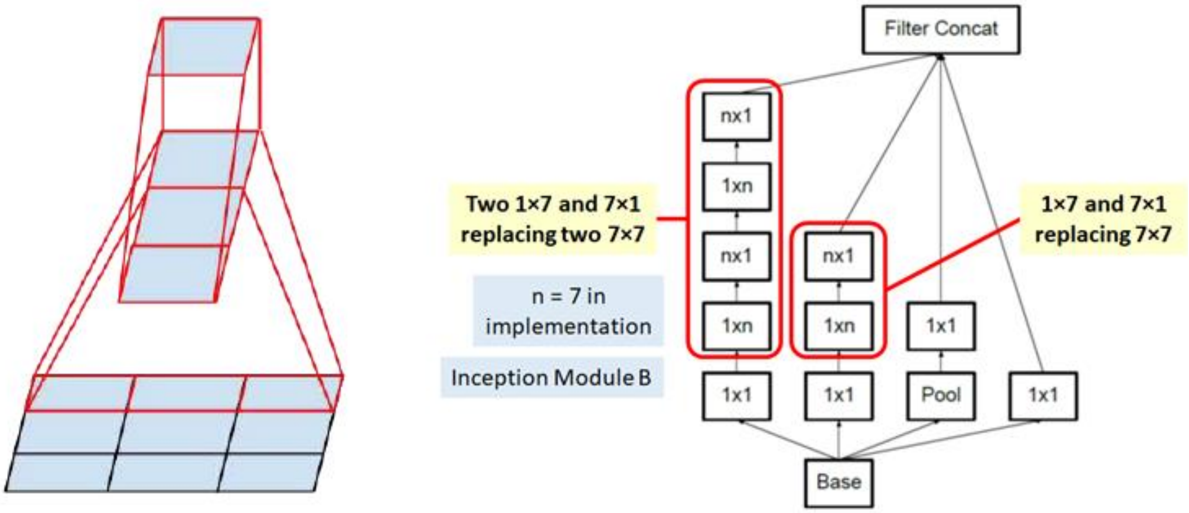


Figure 15. 7×7 factorized Convolution, source [31]

3.4.3.6 ResNet

In this architecture the authors introduced residual learning framework to train the networks in easy way that are deeper than those used previously [23]. It brought a massive improvement in accuracy and a major speed improvement.

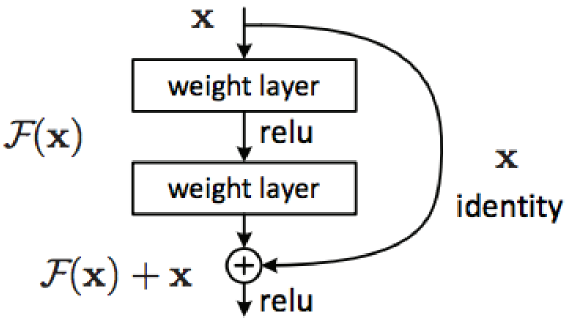

x
weight layer
relu
weight layer
$\mathcal{F}(\mathbf{x})$
$\mathbf{x}$
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu

Figure 16. Residual learning, source [23]

Residual learning was inspired by lateral inhibition in the human brain. It means that the neurons in the brain can control whether their neighboring neurons was triggered or not, that’s mean carry one activation from a layer and feed it to a layer much deeper (not directly next one as used to be).

3.4.3.7 Xception

Inception modules have been replaced with depthwise separable convolutions.

Xception modified the original inception block by making it wider and replacing the different spatial dimensions (1x1, 5x5, 3x3) with a single dimension (3x3) followed by a 1x1 convolution to regulate computational complexity.

Xception architecture shows a slight advantage over Inception V3 on the ImageNet dataset (which Inception V3 was designed for) and Show clear superiority over Inception V3 on a larger image classification dataset comprising 350 million images and 17,000 classes.

Keeping the same number of parameters as in Inception V3 shows that efficient use of model parameters was the main reason to increase performance [24]

Depthwise separable convolution:

the channel wise and spatial-wise computation is done within one step in standard convolution performs, on the other hand, the computation splits into two steps in Depthwise Separable Convolution: Depthwise convolution applies a single convolutional filter per each input channel and pointwise convolution is used to create a linear combination of the output of the depthwise convolution. comparison is shown in Figure 17.

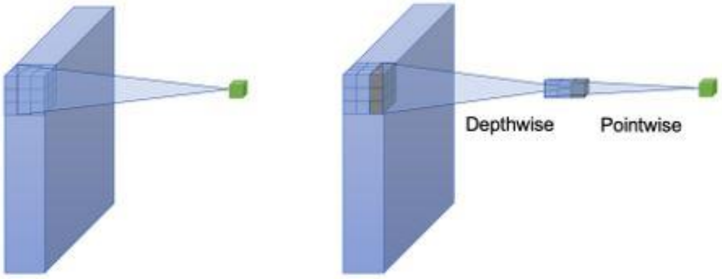


Figure 17. Standard convolution and Depthwise convolution, source [25]

Depthwise separable convolution have fewer parameters than standard convolution that makes it less prone to overfitting. Fewer parameters, means less operations to compute, and thus are cheaper and faster

3.4.3.8 Inception v4 and Inception-ResNet

These tow architectures were presented in the same paper in 2016 [26] , the author noticed that some of the modules were more complicated than necessary, despite of It was built on the same concept of previous versions, but it has more simplified architecture and more inception modules than Inception-v3. The overall schema of the inception-v4 network Figure 18.

We can see the new block in the architecture which is the Stem, and its contents are as in the figure 19.

On the other hand, the inception-Resnet-v2 has almost the same computational cost and it was being trained much faster but reached a bit worse final accuracy than Inception-v3.

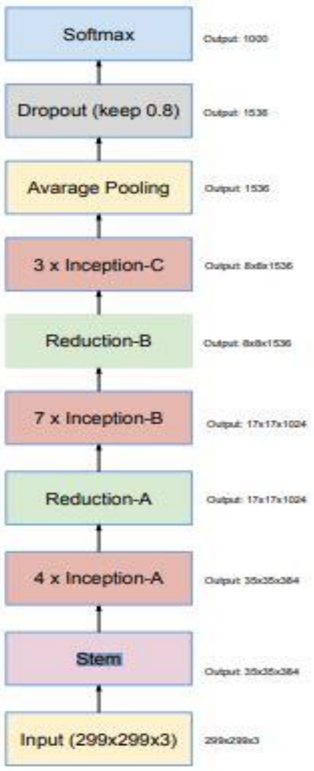


Figure 18. Inception v4, source [26]

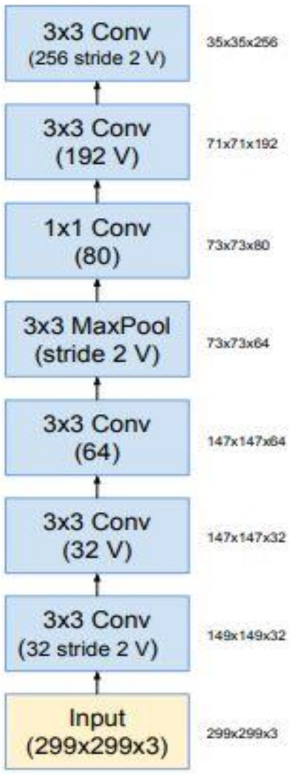


Figure 19, Stem, source [26]

3.4.3.9 ResNeXt

The ResNeXt architecture is an evolution of the deep residual network which replaces the standard residual block with one that benefit from the "split-transform-merge" strategy used in the Inception models. Simply, rather than performing convolutions over the full input feature map, the block's input is projected into a series of lower (channel) dimensional representations of which we separately apply a few convolutional filters before merging the results. [27]

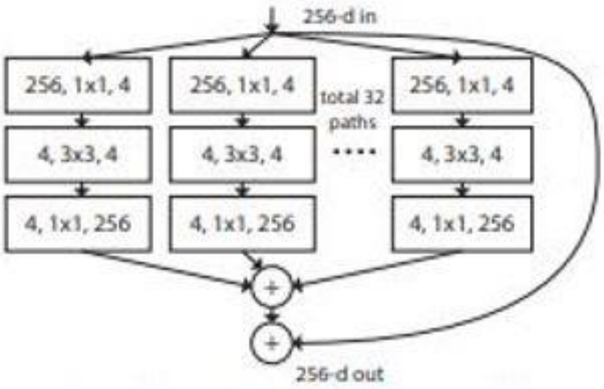


Figure 20. Block of ResNeXt, source [27]

3.4.3.10 SENet

It can be considered an extension of inception network or ResNet, It introduced a building block called Squeeze-and-Excitation block (SE Block) that adaptively recalibrates channel-wise feature responses by explicitly modelling interdependencies between channels, these blocks bring significant improvements in performance for existing state-of-the-art CNNs at slight additional computational cost [28], in another way, the new block analyses the output of the unit which it attached to , focusing especially on the depth dimension and it learns which features are usually most active together and then use this info to recalibrate the feature maps, as in Figure 21.

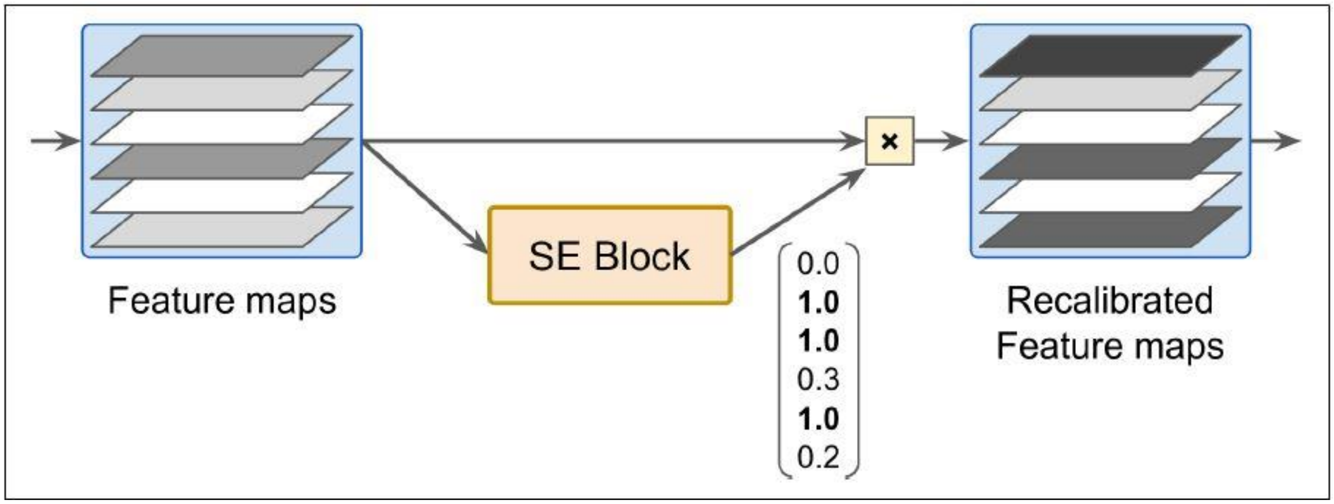


Figure 21, Squeeze-and-Excitation block, source [2]

The result on ImageNet dataset was improved by 25%, it won the competition with an amazing 2.25% top-5 error rate

When generating the output features map, the network usually weights each of its channels equally. SENet aim to change this by incorporating a content-aware mechanism that adjusts the weighting of each channel. This may be as simple as adding a single parameter to each channel and assigning a linear scalar to how significant each one is.

Squeezing the features maps to a single numeric value, on the other hand, gives them a global understanding of each channel. This yields an n-dimensional vector, where n is the number of convolutional channels. It is then fed into a two-layer neural network, which produces a vector of the same size. On the original function maps, these n values can now be used as weights.

3.4.3.11 NASNet

The authors studied a method to learn the model architectures directly on the dataset of interest. As this approach is expensive when the dataset is large, they suggest searching for an architectural building block on a small dataset and then transfer the block to a larger dataset [29]

The basic idea was to find the best combination of filter sizes, output channels, strides, number of layers, and other parameters in the specified search space. The accuracy of the searched architecture on the given dataset was the reward for each search operation in this Reinforcement Learning environment.

In the ImageNet competition, NASNet received a state-of-the-art result. The computing power needed for NASNet, on the other hand, was so high that only a few companies were able to use the same technique.

3.4.3.12 GhostNet

The author [53] proposes a novel Ghost module to generate more feature maps from cheap operations, so this module can be used on devices with limited memory and limited computational power like embedded devices. This module can be used as a plug-and-play component to upgrade existing convolutional neural networks.

3.4.4 Comparison between state of arts of CNN architectures

Most deep neural networks (DNNs) suggested in the state-of-the-art for image recognition are examined in detail by researchers [30]. Recognition accuracy, model complexity, computational complexity. In Figure 22. we can see that ball chart reporting the Top-1 accuracy vs. computational complexity. Top-1 using only the center crop versus floating-point operations (FLOPs) required for a single forward pass are reported. The size of each ball corresponds to the model complexity.

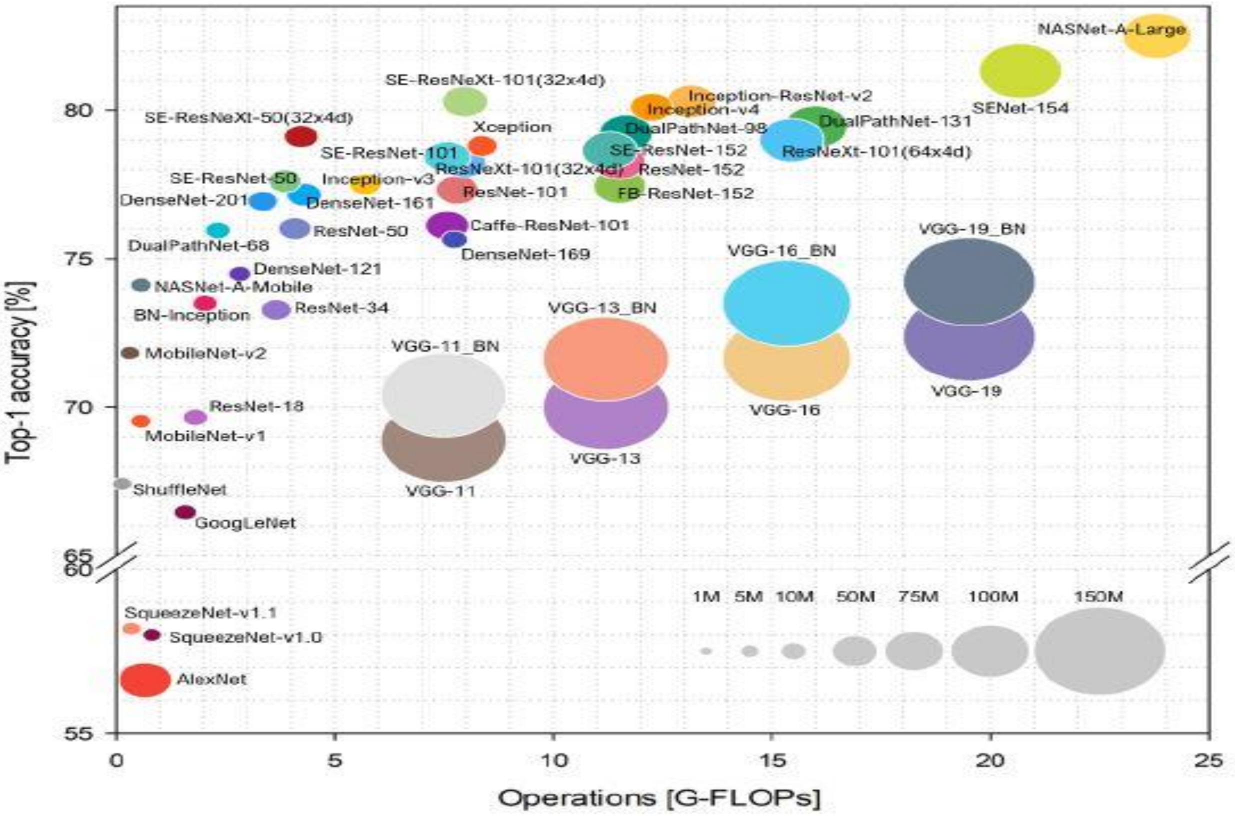


Figure 22. Top-1 accuracy vs. computational complexity, source [30]

3.4.5 An overview of the areas in which CNN is used

In addition to image classification CNN has succeeded in various tasks related to machine learning. Here are the most famous areas in which it has achieved great success.

3.4.5.1 NLP (natural language processing)

DNNs have revolutionized the field of NLP, and the two main types of DNN architectures are CNN and Recurrent Neural Network (RNN), which they are handle different NLP tasks. CNN is supposed to be good at extracting position in variant features and RNN at modeling units in sequence. There is great competition between the two algorithms to prove the advantage of one of them.

In paper [41] the authors found that RNNs perform well and robust in a broad range of tasks except when the task is essentially a key phrase recognition task as in some sentiment detection and question-answer matching settings. In addition, hidden size and batch size can make DNN performance vary dramatically. This suggests that optimization of these two parameters is crucial to good performance of both CNNs and RNNs.

There are many tasks NLP can perform, such Text classification, and Speech recognition. In [43] the CNN shows advantages over DNN like Noise robustness, Distant speech recognition, Low-footprint models, and Channel-mismatched training-test conditions

3.4.5.2 computer vision

As in all systems that seek to simulate and implement the tasks that a person performs, computer vision works to carry out the tasks of the human eye and perform image analysis in a manner like the brain, and one of the most important of these applications is face recognition. CNN ability to display images as data, make it a popular solution in different computer vision tasks.

Face recognition

The difficulty lies in recognizing faces because of changing facial features, whether it is due to an emotional state, or because of a reaction to the weather and temperature, or because of changes that occur due to lighting or change in the face pose.

The studies on LFW database showed that performance improves steadily from about 60% to above 90%, while using deep learning improves performance by up to 99.80% in just three years [32]

there are three main steps needed, Figure 23.

1- Face detector is used to localize faces in images or videos.

2- The faces are aligned.

3- The FR module is implemented

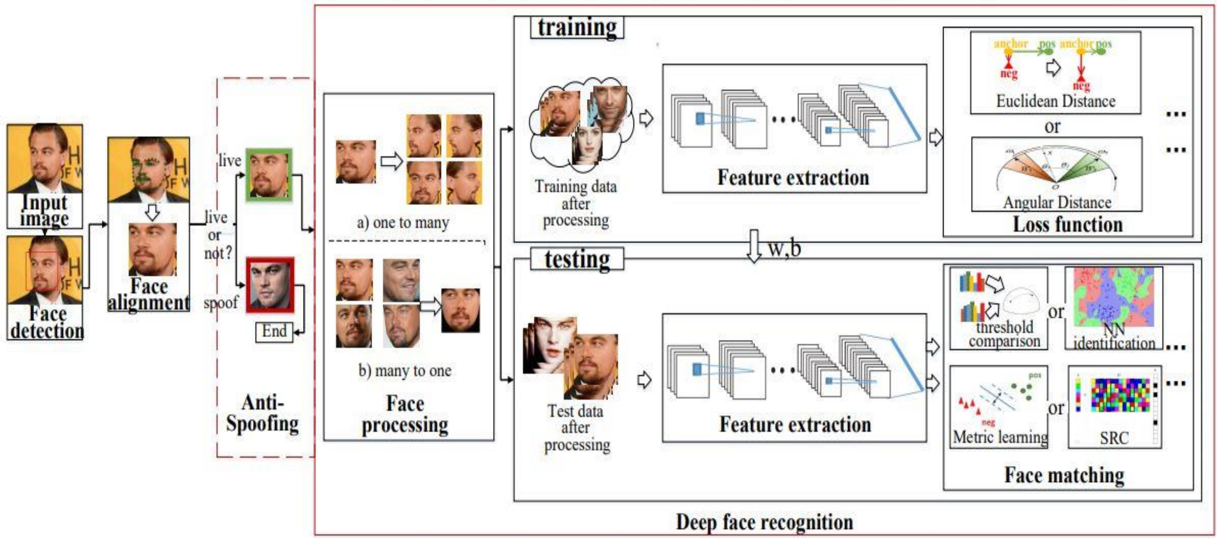


Figure 23. three steps of face recognition, source [32]

In [33] the paper shows that a deep CNN, without any embellishments but with appropriate training, can achieve results comparable to the state of the art

We can see that FR used in wide area like

- identify people on social media platforms
- find missing persons
- unlock phones

and many other applications

Vehicle detection

Vehicle detection and tracking applications play an important role in monitoring highway traffic and helping to regulate traffic. Road vehicle detection is used to monitor vehicle speed, traffic accidents, traffic jams, traffic analysis, and vehicle classification, and can be implemented in different environments.

Autonomous driving

We already have Advanced Driving Assistance System (ADAS), which provides the driver with the latest available surrounding information from sonar, radar and cameras deployed along the road and it is noticeable that these technologies are used for long-range detection, here comes the role of CNN in near and medium-range detection such as pedestrians, road signs and nearby cars.

For autonomous driving, there must be a comprehensive understanding of the surrounding environment and road conditions, an understanding of the signs on the roads, the location of obstacles, and possibly sudden accidents and others, and the ability to deal with them.

Autonomous driving consists of three main tasks [44]:

- Perception: understanding of the environment.
- Planning: the process of making decisions to achieve the vehicle's goals.
- Control: the vehicle's ability to execute the planned actions.

CNN-based object detection and semantic segmentation are useful for these tasks.

In [45] author proposed an end-to-end learning method for autonomous driving, which input dash-board camera images into a CNN and outputs steering angle directory. The advantage of the system configuration is simplified because CNN learns automatically and consistently without explicit understanding of the surrounding environment and motion planning [44].

Pose estimation

Understanding the human pose is one of the difficult challenges in computer vision. The reason for this is the high dimensions of the input data as well as the large number of poses the body can take.

DeepPose [38] is the first application of CNNs to human pose estimation problems. the task is formulated as a regression problem to body joint coordinates. DeepPose captures the full context of each body joint by taking the whole image as the input [37]

[42] State-of-the-art performance for human pose estimation has been achieved using Deep Convolutional Neural Networks. These networks use two approaches:

- Regressing heat maps of each body part
- Learning deep-structured output

Room classification

The ability to classify rooms in a home is one of the many desirable features of social robots. In the paper (40), the researchers address the problem of classifying internal chambers across several convolutional neural network (CNN) architectures, such as VGG16, VGG19, and Inception V3. The main goal is to learn about the five interior classes (bathroom, bedroom, dining room, kitchen, and living room).

Object recognition

Object detection problem is more complicated than classification, regarding classification can also recognize objects but has trouble locating it, also classification does not work when there are multiple objects in one image.

there are many concepts under this term:

- **Object detection:** classify multiple objects in an image and place bounding boxes around them.
- **Object localization**: detect the presence of objects in an image and indicate their location with a bounding box.
- **Image Classification**: Predict the class or type of one or multiple objects in an image.
- **Semantic segmentation**: classify each pixel according to the class of the object that belongs to.

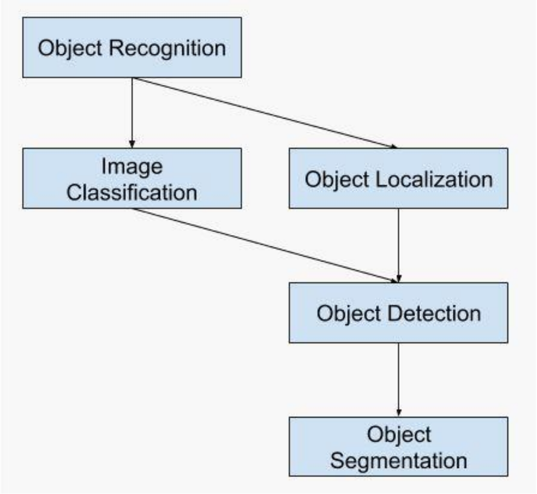


Figure 24, Object recognition concepts, source [2]

For many years the common approach was to take trained CNN network and then slide it across image, this technique was slightly simple, because it will detect the same object multiple times.

One of top-performing deep learning models is R-CNN which was introduced in in the 2014 paper by Ross Girshick [35], this model consists of 3 modules:

- Region Proposal: propose bounding boxes which are most likely to contain an object.
- Feature Extractor.
- Classifier.

But R-CNN had a problem that can't be implemented in real time and it takes some time, that leads us to the Fast R-CNN which was significantly faster.

Finally, we have YOLO (You Only Look Once), Unlike previous algorithms which is region-based algorithms, in YOLO the bounding boxes and class probabilities for these boxes are predicted by a single convolutional network. [36], it achieves high accuracy and it's running in real time and it considers now one of the most efficient object detection algorithms [51]

Robotic Grasp Detection

There is a time delay between the robot's response to grasp objects and the human response. Humans can know how to grasp new things even if they do not have previous knowledge about it. Therefore, the ability of robot to grasp object in real time is a difficult process, and as is the case for autonomous driving of vehicles, the task of capture is divided into three stages, detection, planning, and execution. In [46] authors used typical 3D vision system or RGB-D camera to detect objects and their approach was using ResNet-50 networks running in parallel to extract features from RGB-D images. First network task is analyzing the RGB component, and second network task is analyzing the depth channel. The outputs of these networks are then merged and fed into another convolutional network that predicts the grasp configuration, the complete architecture in Figure 25.

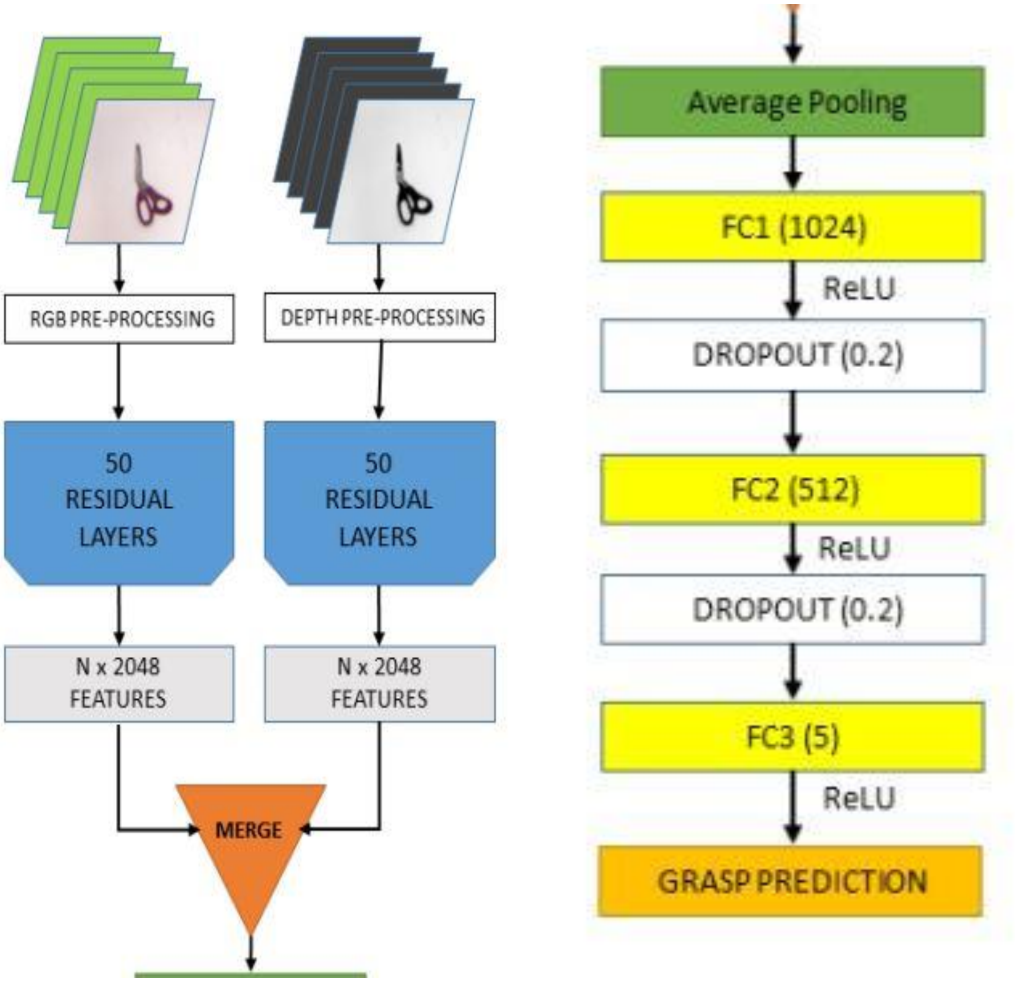


Figure 25, Robotic Grasp Detection network, source [46]

Learning robot dynamically

The environment around us is constantly changing and this is a problem for robots to understand this change and therefore it is necessary for the robot to learn the changes taking place in the surrounding environment dynamically. Authors in [47] used an network based on ResNet-50 and they found that using a pure CNN based architecture on both CNN and naïve classifier gives the best performance, although pure CNN based architecture achieved slightly better classification accuracy, but it takes long time for training which makes it useless on a dynamically learning robot

Radiology

The medical field has not been forgotten from the advancement in deep learning, particularly in CNN, as in diabetic retinopathy screening, skin lesion classification, and lymph node metastasis detection and radiology [48] due to its great ability in image classification, it has achieved expert-level performances in various fields. For example, it used for the classification of lung nodules whether it was benign or malignant figure 26. [48]

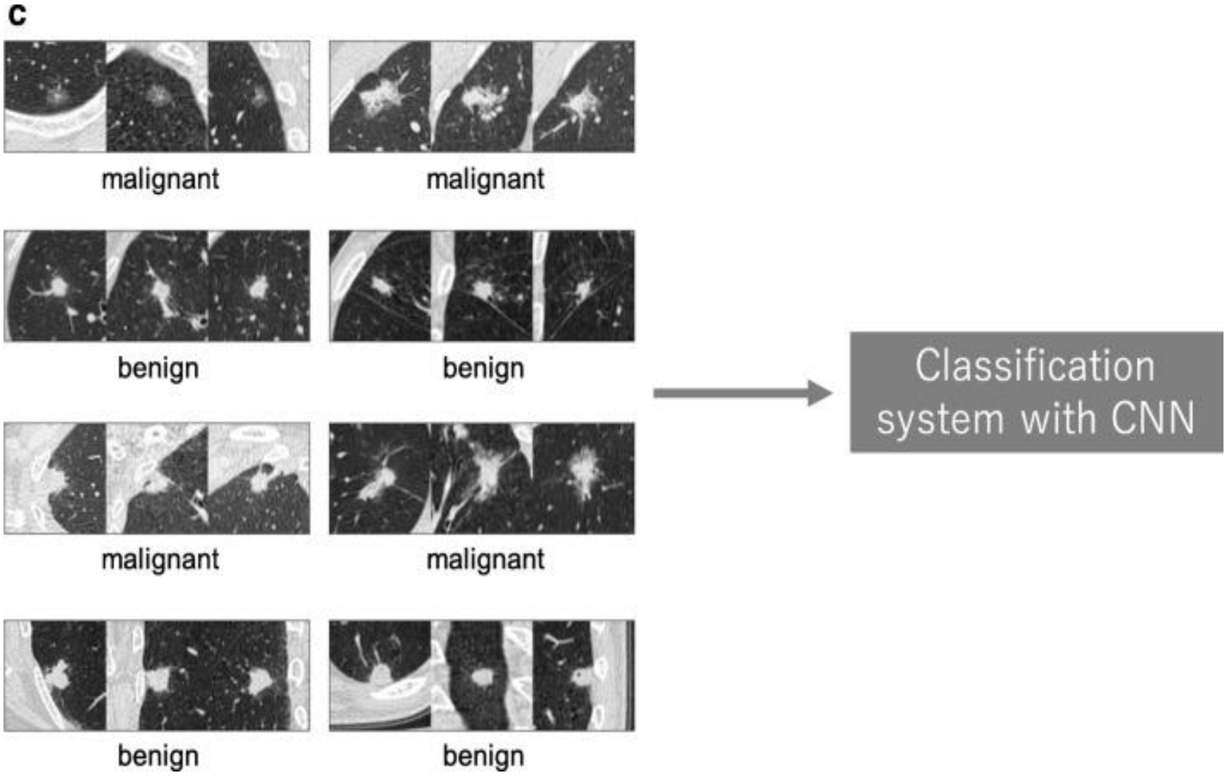


Figure 26. benign or malignant classification, source [48]

Classification depends mainly on segmentation to differentiate between benign and malignant parts. Segmentation can be done by the specialists manually, but it will consume a lot of time. Training data for the segmentation system consist of the target organ and the segmentation result.

Covid-19

Since covid is a respiratory disease, a chest radiograph images are one of the basic keys to diagnosing the disease and knowing its effects, by studying the occurring abnormalities. Authors in [50] introduced open-source network designs for COVID-19 detection from CXR images called COVID-Net and they used 13,975 CXR images.

The authors view the testing process as a complex manual process that takes a long time and has a very variable sensitivity, and the way this test works has not been explained clearly, in addition to changing the results according to the sampling method. They introduced alternative method, radiography examination be analyzing CXR images, it can be done quickly and is considered to be a good complement to PCR Test

COVID-Net was pretrained on ImageNet dataset and then trained on the CXR images dataset, we can see the architecture in Figure 27.

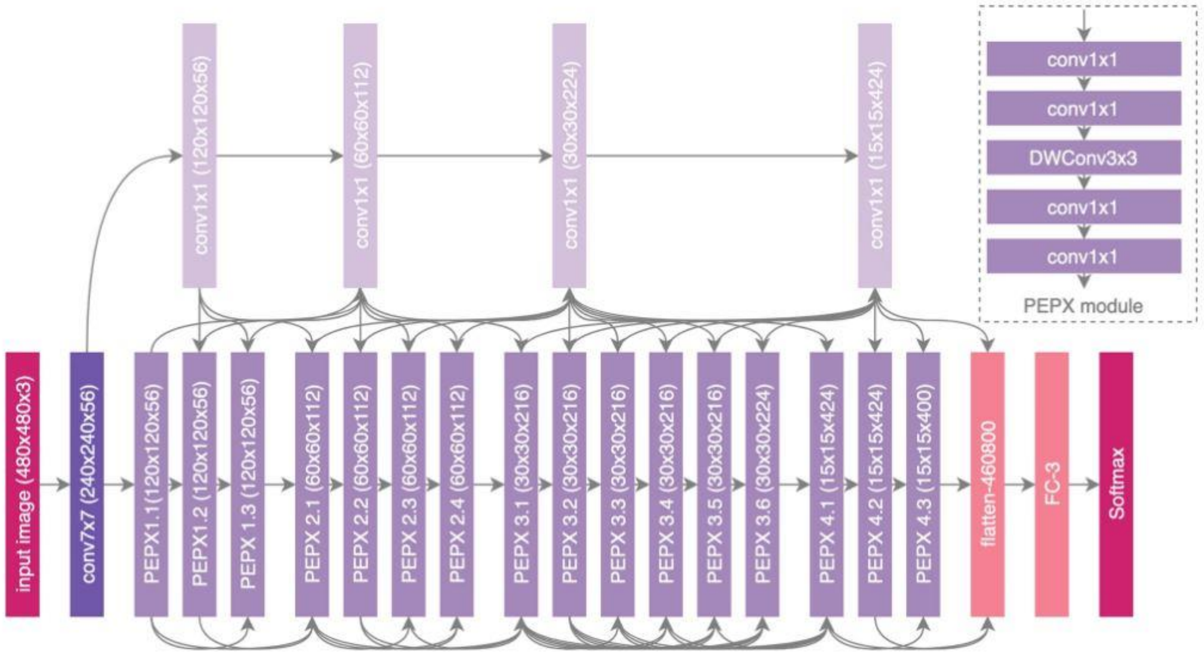


Figure 27. COVID-Net architecture, source [50]

Breast cancer detection in Mammography

Breast cancer is the second leading cause of cancer deaths among U.S. women and screening mammography has been found to reduce mortality. Despite the benefits, screening mammography is associated with a high risk of false positives as well as false negatives [52] the authors proposed to use convolutional layers as top layers, which preserve spatial information. Two blocks of convolutional layers (VGG or Resnet) can be added on top of the patch classifier layers, followed by a global average pooling layer and then the image's classification output as in Figure 28.

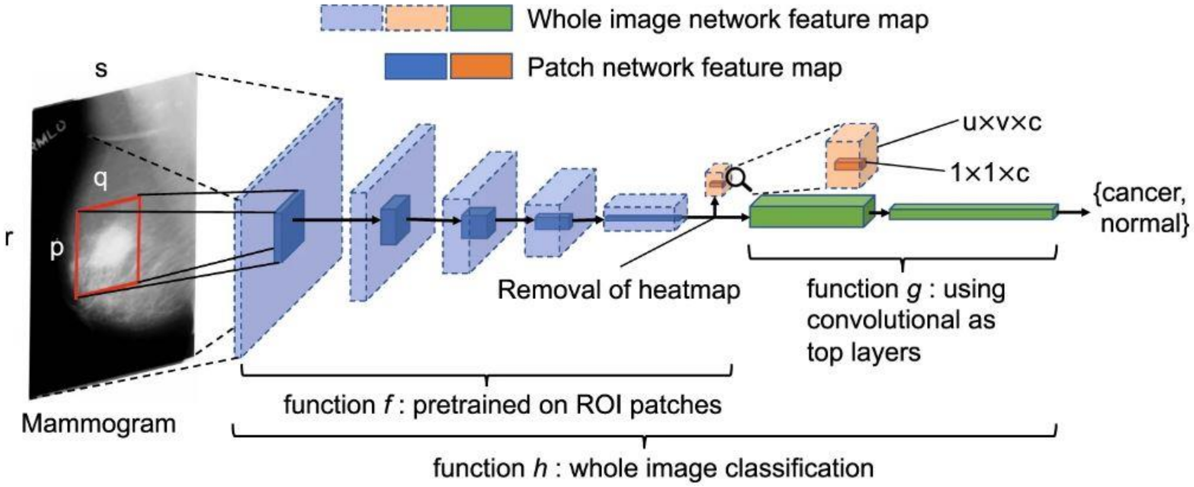

Whole image network feature map
Patch network feature map
s
q
r
p
Mammogram
u×v×c
1×1×c
{cancer, normal}
Removal of heatmap
function g : using convolutional as top layers
function f : pretrained on ROI patches
function h : whole image classification

Figure 28, Breast cancer detection in Mammography, source [52]

The results of this paper shows that CNN is outperforming human specialists in detecting and classifying tumors.

3.4.6 Open-source frameworks

The number of deep learning frameworks is as large as the algorithms used for that, and the goal of these frameworks is to create a suitable environment for working through standard programming languages, and often more than one framework is combined to reach the desired results. Reliability, scalability, and cost give big advantage for open-source software.

One of the most used frameworks are: Microsoft CNTK, Caffe, Caffe2, Torch, PyTorch, MXNet, Chainer and Theano, all these frameworks are open source, but the most popular according to the number of developers who use these frameworks are TensorFlow and the high-level API library Keras and they are growing very fast [54]

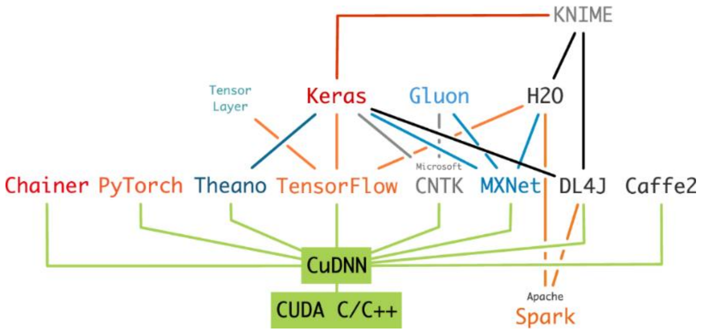


Figure 29. The most popular Deep Learning frameworks and libraries, source [54]

TensorFlow

It is an open-source numerical framework created and developed by Google Brain for large-scale distributed training and inference and for use both in research, development and production systems, it can run on single CPU systems, GPUs, mobile devices and large-scale distributed systems of hundreds of nodes, is also supported in Google and Amazon cloud environments [54].

Strong points

- By far the most popular DL tool, open-source, fast evolving, supported by a strong industrial company (Google).
- Numerical library for dataflow programming that provides the basis for DL research and development.
- Efficiently works with mathematical expressions involving multi-dimensional arrays.
- GPU/CPU computing, efficient in multi-GPU settings, mobile computing, high scalability of computation across machines and huge data sets. [54]

Weak points

- Defines computational flow (graph) statically before a model can run
- Still lower-level API difficult to use directly for creating DL models. [54]

Keras

Keras is Python wrapper library that provides bindings to other DL tools such as TensorFlow, It was developed with a focus on enabling fast experimentation and is released under the MIT license. Work with Python Models are described in Python code, which is compact, easier to debug, and allows for ease of extensibility [54]

Strong points

- Open-source, fast evolving, with backend tools from strong industrial companies like Google and Microsoft
- User friendliness
- Clean and convenient way to quickly define DL models on top of backends (e.g. TensorFlow, Theano, CNTK). Keras wraps backend libraries, abstracting their capabilities and hiding their complexity.

Weak points

- Sometimes it is slow on GPU and takes longer time in computation compared with its backends
- Multi-GPU not 100% working

Following Table 1. shows comparison between TensorFlow, keras and the third popular framework nowadays PyTorch and Caffe2

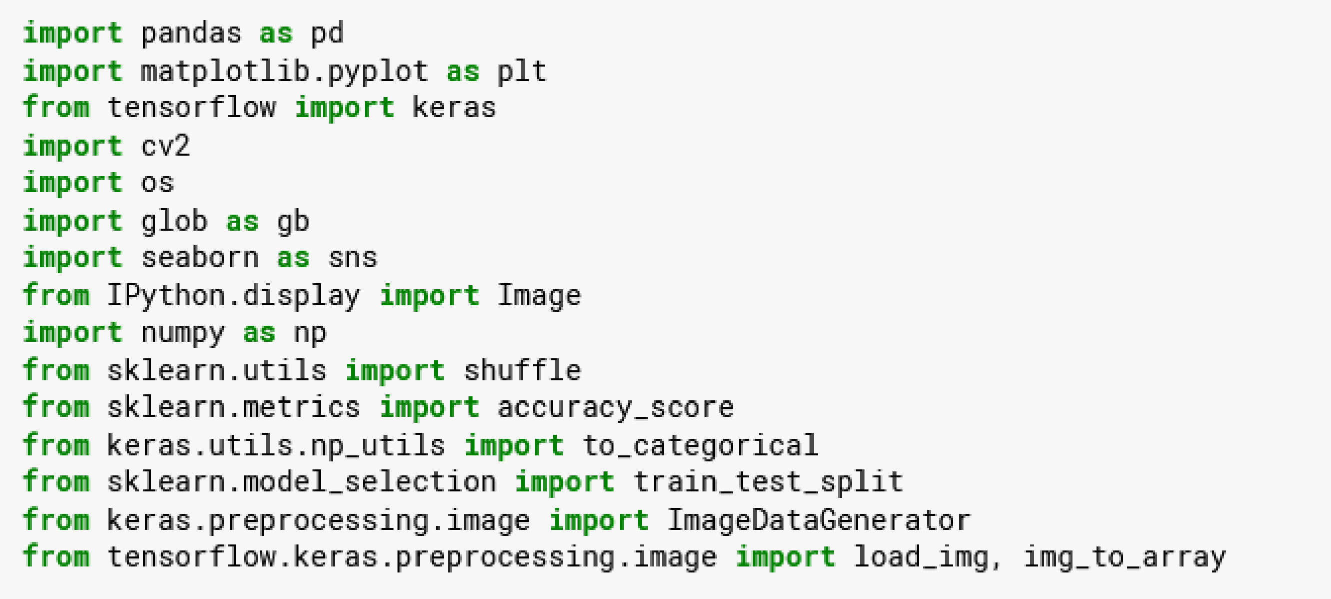
Framework, Library	**Computation graph**	**Usage**	**Datasets**	**Popularity**	**Trend**
TensorFlow	Static with small support for dynamic graph	Academic Industrial	Large datasets, high performance	Most popular	Using Keras for fast prototyping and TensorFlow for production. This trend is backed by Google.
Keras	Static	Academic Industrial	Smaller datasets	Second most popular	
PyTorch	Dynamic	Academic Industrial	Large datasets, high performance	Third most popular	Using PyTorch for prototyping and Caffe2 for production. This trend is backed by Facebook.
Caffe2	Static	Academic Industrial Mobile solution	Large datasets	Medium-low Growing fast	

Table 1. Most Popular Deep learning frameworks

4. Practical part

It is known that CNN help in running neural networks directly on images and are more efficient and accurate than many of the deep neural networks architectures. In this task, we will use tensorflow-keras package to build CNN model.

First, I will Import the libraries which I will use it in the task:



```
import pandas as pd
import matplotlib.pyplot as plt
from tensorflow import keras
import cv2
import os
import glob as gb
import seaborn as sns
from IPython.display import Image
import numpy as np
from sklearn.utils import shuffle
from sklearn.metrics import accuracy_score
from keras.utils.np_utils import to_categorical
from sklearn.model_selection import train_test_split
from keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.preprocessing.image import load_img, img_to_array
```

4.1 Explore the train images

The German traffic signs detection dataset will be used in this task to recognize the traffic signs by external camera connected with the trained model.

The dataset consists of 39209 images as a train data and 12630 images as test data with 43 different classes. Also, there is a file attached to the data (Meta)file contains 43 different images (100,100,3) are shown in Figure to make the last test before implementing and testing the model by the external camera.

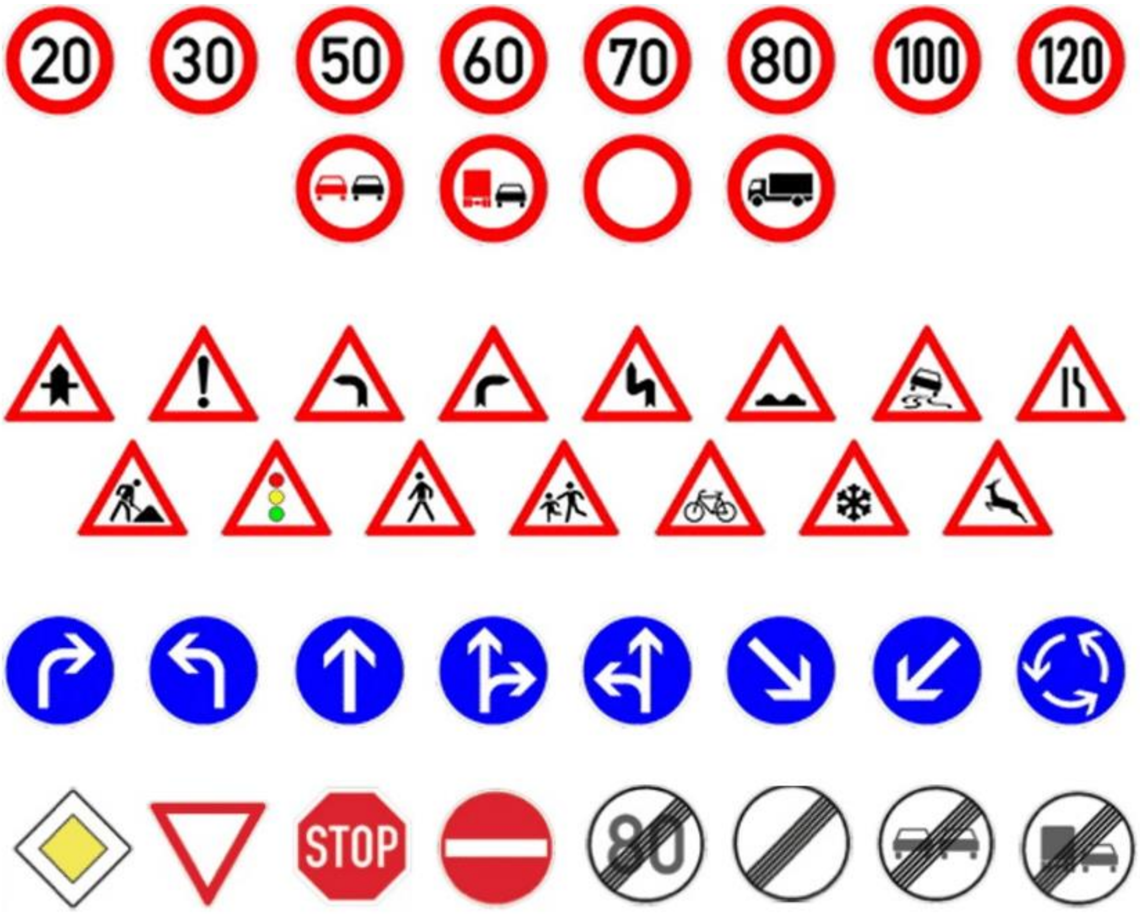
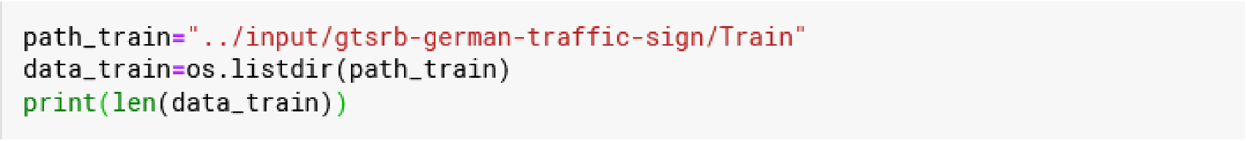


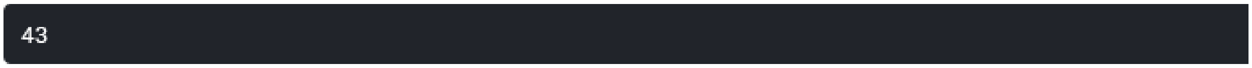
Figure 30. 43 different images represent 43 class

The images are distributed unevenly between those classes and hence the model may predict some classes more accurately than other classes. For that we used a data augmentation by populate the dataset with various image modifying techniques such as rotation, color distortion or blurring the image.to achieve the greatest equality between the data classes.

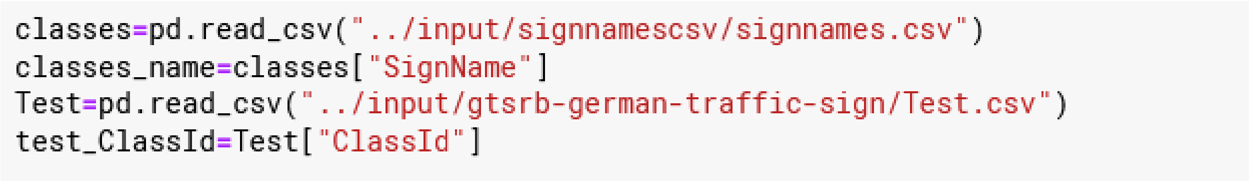
To open the files and join paths of dataset's folders, we will use **os** and glob packages:



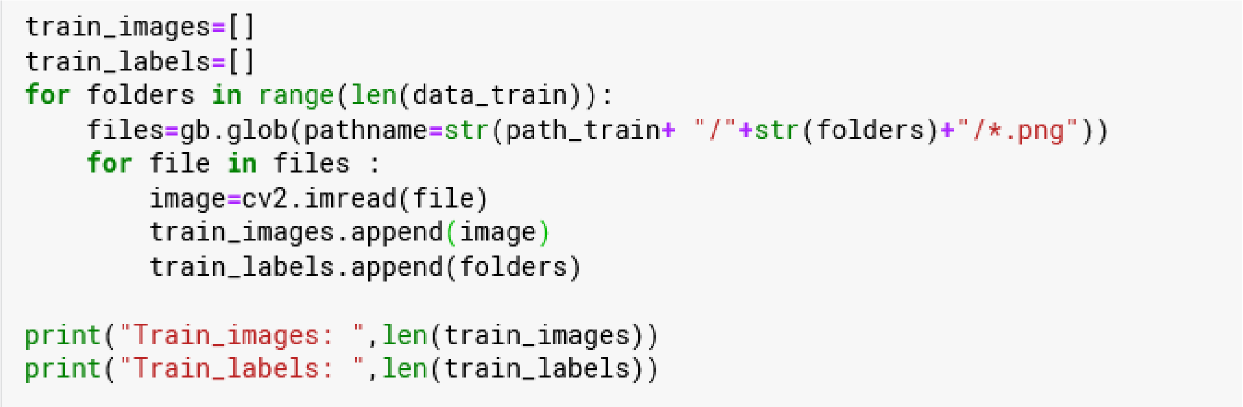
```
path_train="../input/gtsrb-german-traffic-sign/Train"
data_train=os.listdir(path_train)
print(len(data_train))
```



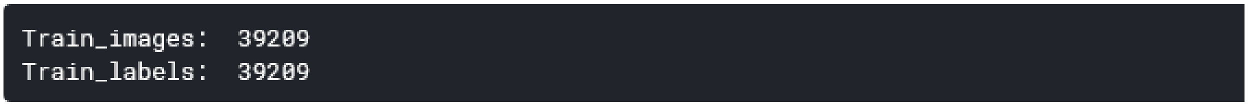
```
43
```



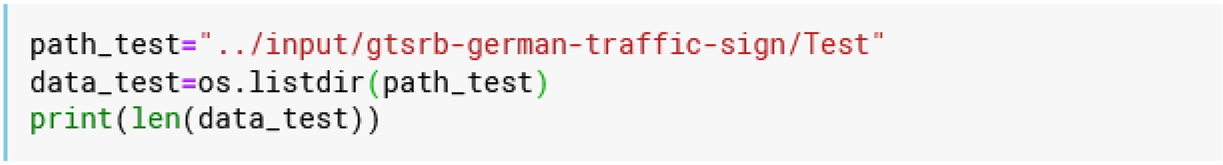
```
classes=pd.read_csv("../input/signnamescsv/signnames.csv")
classes_name=classes["SignName"]
Test=pd.read_csv("../input/gtsrb-german-traffic-sign/Test.csv")
test_ClassId=Test["ClassId"]
```



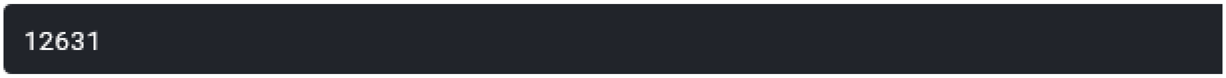
```
train_images=[]
train_labels=[]
for folders in range(len(data_train)):
    files=gb.glob(pathname=str(path_train+ "/"+str(folders)+"/*.png"))
    for file in files :
        image=cv2.imread(file)
        train_images.append(image)
        train_labels.append(folders)

print("Train_images: ",len(train_images))
print("Train_labels: ",len(train_labels))
```



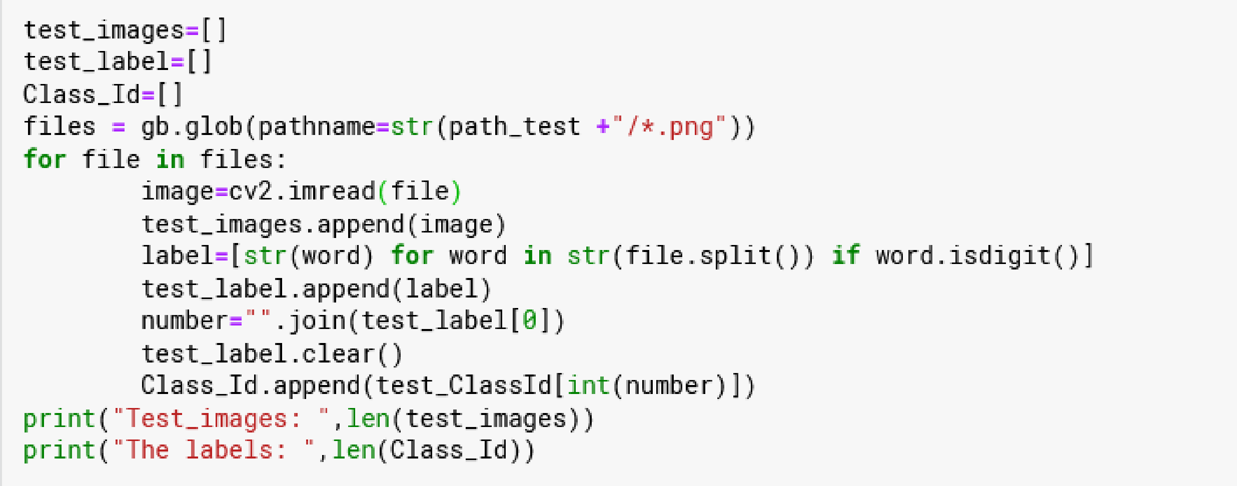
```
Train_images:  39209
Train_labels:  39209
```



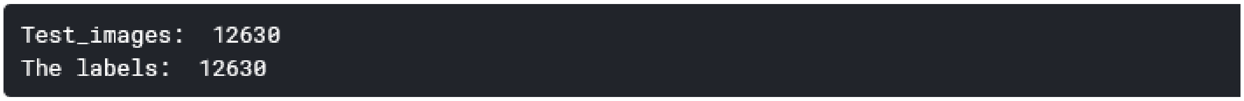
```
path_test="../input/gtsrb-german-traffic-sign/Test"
data_test=os.listdir(path_test)
print(len(data_test))
```



```
12631
```



```
test_images=[]
test_label=[]
Class_Id=[]
files = gb.glob(pathname=str(path_test +"/*.png"))
for file in files:
        image=cv2.imread(file)
        test_images.append(image)
        label=[str(word) for word in str(file.split()) if word.isdigit()]
        test_label.append(label)
        number="".join(test_label[0])
        test_label.clear()
        Class_Id.append(test_ClassId[int(number)])
print("Test_images: ",len(test_images))
print("The labels: ",len(Class_Id))
```

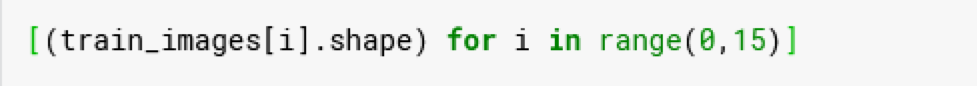


```
Test_images:  12630
The labels:  12630
```

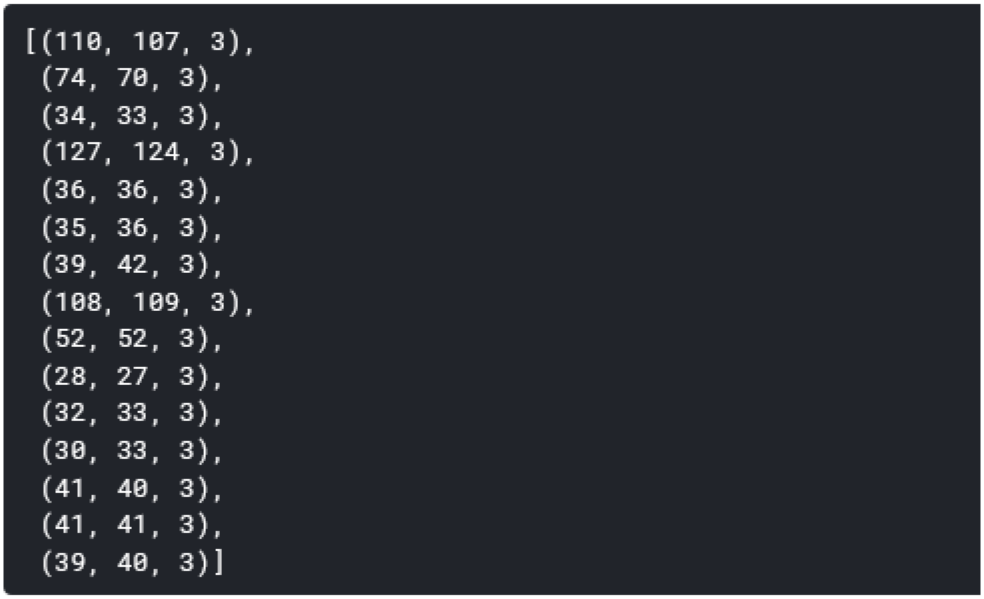
4.2 Data Pre-Processing

One of the limitations of the CNN model is that they cannot be trained on a different dimension of images. So, it is mandatory to have same dimension images in the dataset.

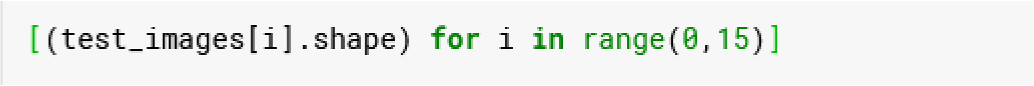
Let's at the beginning check the dimension of all the images (Train, Test) of the dataset so that we can process the images into having similar dimensions:



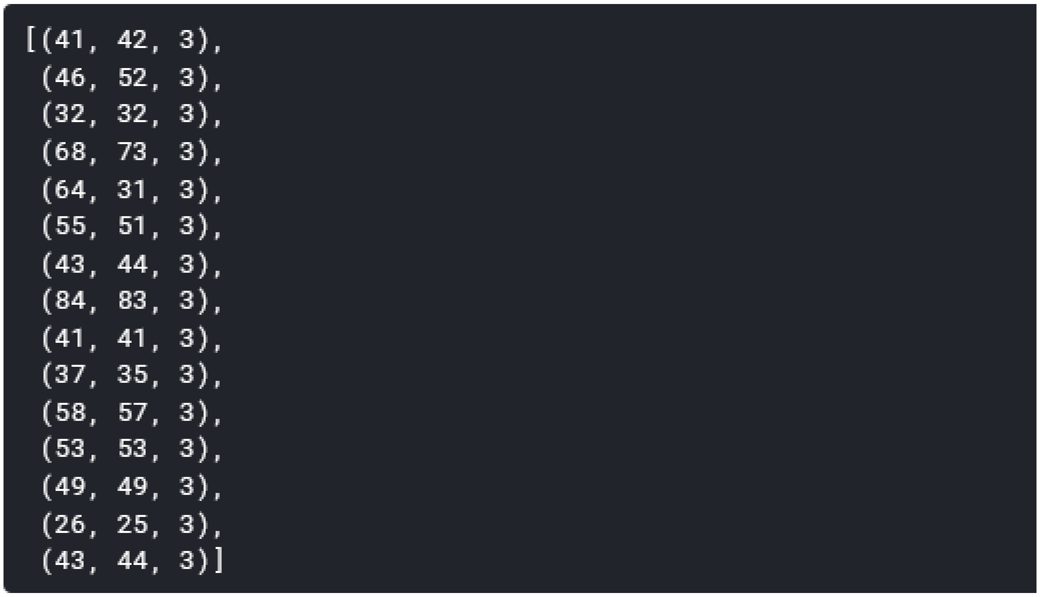
```
[(train_images[i].shape) for i in range(0,15)]
```



```
[(110, 107, 3),
 (74, 70, 3),
 (34, 33, 3),
 (127, 124, 3),
 (36, 36, 3),
 (35, 36, 3),
 (39, 42, 3),
 (108, 109, 3),
 (52, 52, 3),
 (28, 27, 3),
 (32, 33, 3),
 (30, 33, 3),
 (41, 40, 3),
 (41, 41, 3),
 (39, 40, 3)]
```



```
[(test_images[i].shape) for i in range(0,15)]
```



```
[(41, 42, 3),
 (46, 52, 3),
 (32, 32, 3),
 (68, 73, 3),
 (64, 31, 3),
 (55, 51, 3),
 (43, 44, 3),
 (84, 83, 3),
 (41, 41, 3),
 (37, 35, 3),
 (58, 57, 3),
 (53, 53, 3),
 (49, 49, 3),
 (26, 25, 3),
 (43, 44, 3)]
```

It is clear, that we have a different dimension in our both categories the train and the test images so they cannot be passed directly to the ConvNet model. Instead, we should resize all the images to (32,32,3).

Choosing the dimensions of the images should consider two important things:

- The number of parameters that will be processed in the network.
- the have chosen dimension should keep the image data mostly accurate.

I decided to start with (32,32,3) and check the accuracy of the model if it will be sufficient to predict new images.

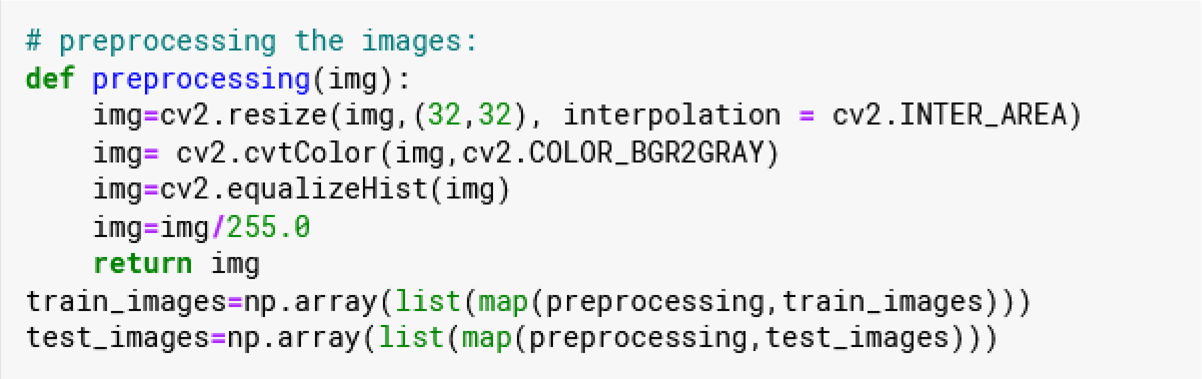
Note: W combined many steps in one function (**preprocessing Fun**).

Preprocessing Function do many steps together:

1. resize the images into (32,32,3) by using OpenCV package.

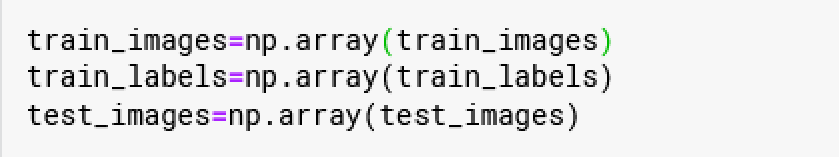
cv2 is a package of OpenCV. resize method transforms the image into the given dimension. OpenCV provides 5 types of interpolation techniques based on the method they use to evaluate the pixel values of the resulting image. The techniques are **INTER_AREA, INTER_NEAREST, INTER_LINEAR, INTER_CUBIC, INTER_LANCZOS4**. We will be using INTER_AREA interpolation technique it's more preferred for image decimation but for extrapolation technique it's similar as INTER_NEAREST.

2. Convert the images to grayscale.
3. Standardize the lighting in an image.
4. normalize the images to new range between 0-1 (This helps the model converge faster).



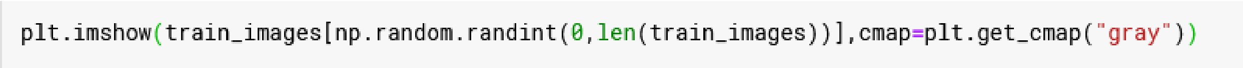
```
# preprocessing the images:
def preprocessing(img):
    img=cv2.resize(img,(32,32), interpolation = cv2.INTER_AREA)
    img= cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
    img=cv2.equalizeHist(img)
    img=img/255.0
    return img
train_images=np.array(list(map(preprocessing,train_images)))
test_images=np.array(list(map(preprocessing,test_images)))
```

The lists which contained the images, and the labels must be converted to arrays to be able to insert into keras.



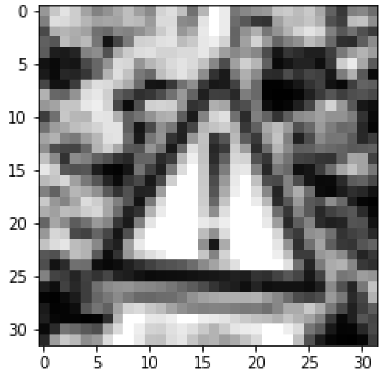
```
train_images=np.array(train_images)
train_labels=np.array(train_labels)
test_images=np.array(test_images)
```

The shape of the images after going out of Preprocessing Funnction:



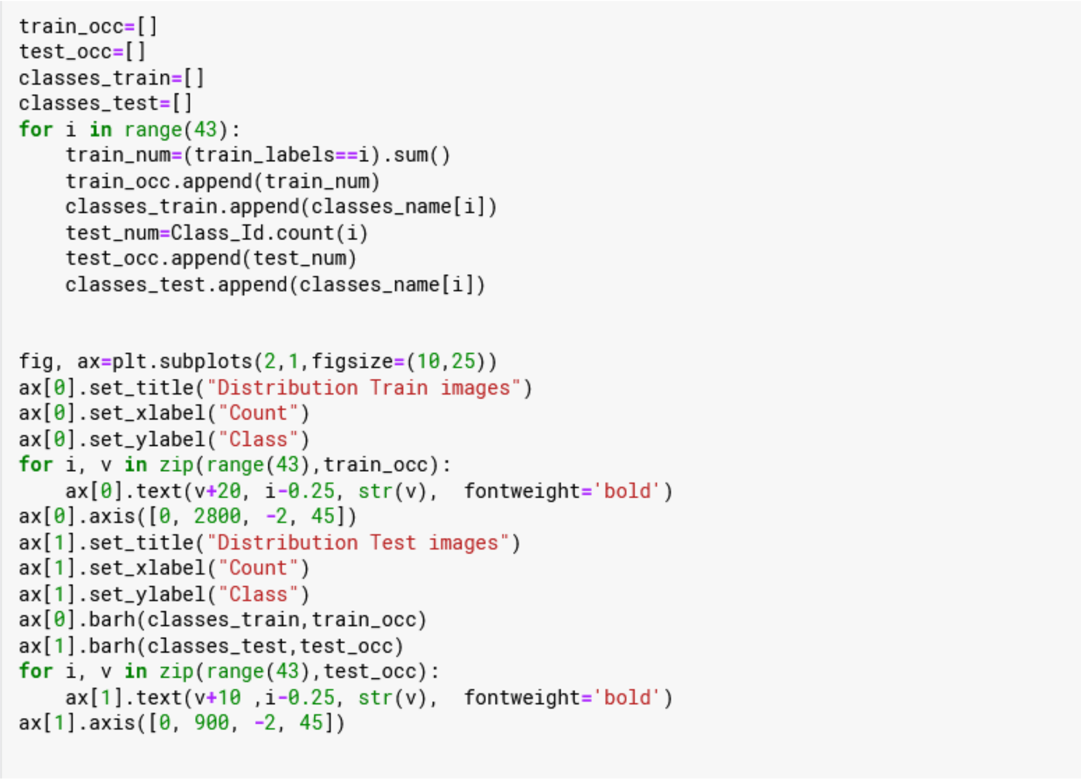
```
plt.imshow(train_images[np.random.randint(0,len(train_images))],cmap=plt.get_cmap("gray"))
```

```
<matplotlib.image.AxesImage at 0x7f81bc87db90>
```



4.3 Visualize the dataset

It is better to see the distribution of the data in the classes in both sets. frequencies of the classes are shown in Figure 30.



```
train_occ=[]
test_occ=[]
classes_train=[]
classes_test=[]
for i in range(43):
    train_num=(train_labels==i).sum()
    train_occ.append(train_num)
    classes_train.append(classes_name[i])
    test_num=Class_Id.count(i)
    test_occ.append(test_num)
    classes_test.append(classes_name[i])


fig, ax=plt.subplots(2,1,figsize=(10,25))
ax[0].set_title("Distribution Train images")
ax[0].set_xlabel("Count")
ax[0].set_ylabel("Class")
for i, v in zip(range(43),train_occ):
    ax[0].text(v+20, i-0.25, str(v),  fontweight='bold')
ax[0].axis([0, 2800, -2, 45])
ax[1].set_title("Distribution Test images")
ax[1].set_xlabel("Count")
ax[1].set_ylabel("Class")
ax[0].barh(classes_train,train_occ)
ax[1].barh(classes_test,test_occ)
for i, v in zip(range(43),test_occ):
    ax[1].text(v+10 ,i-0.25, str(v),  fontweight='bold')
ax[1].axis([0, 900, -2, 45])
```

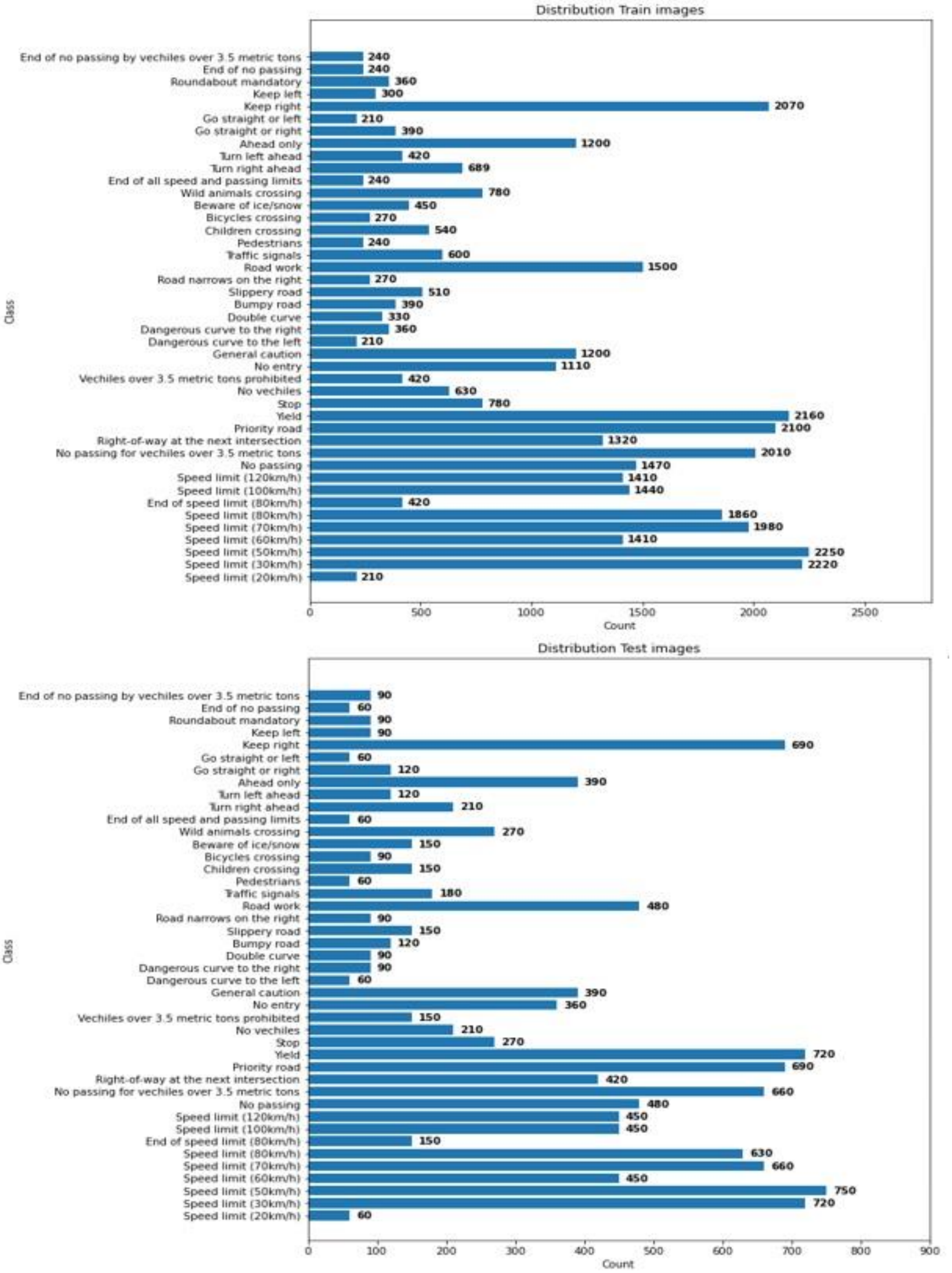
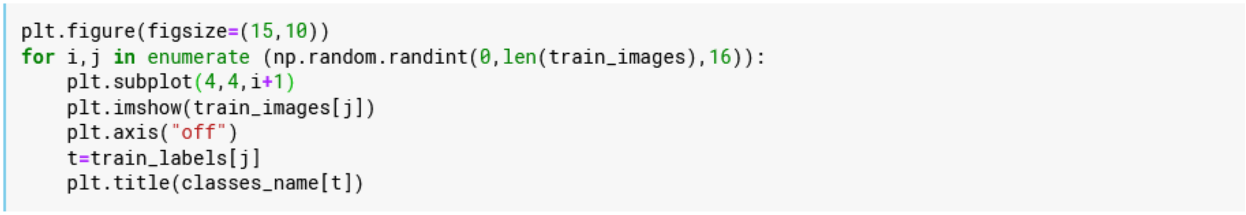
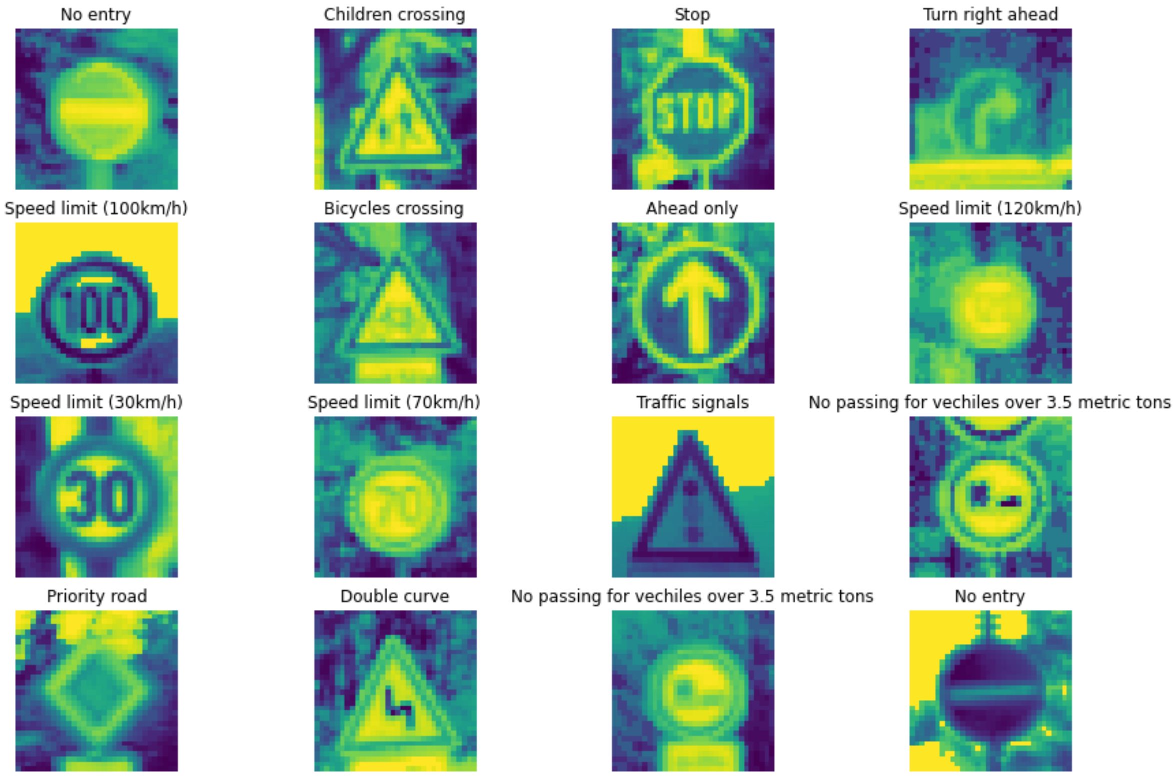


Figure 31, frequencies of classes in both sets

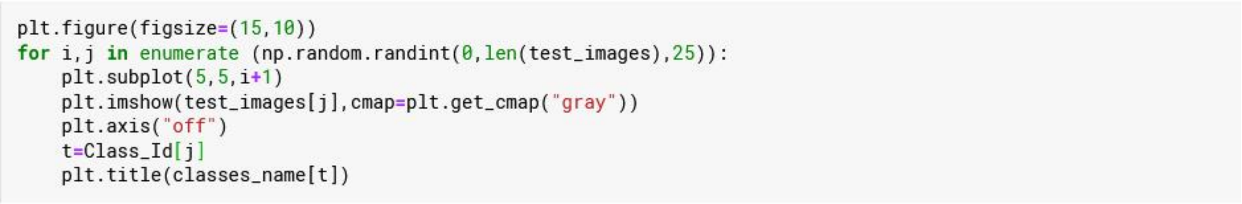
Samples of trained images



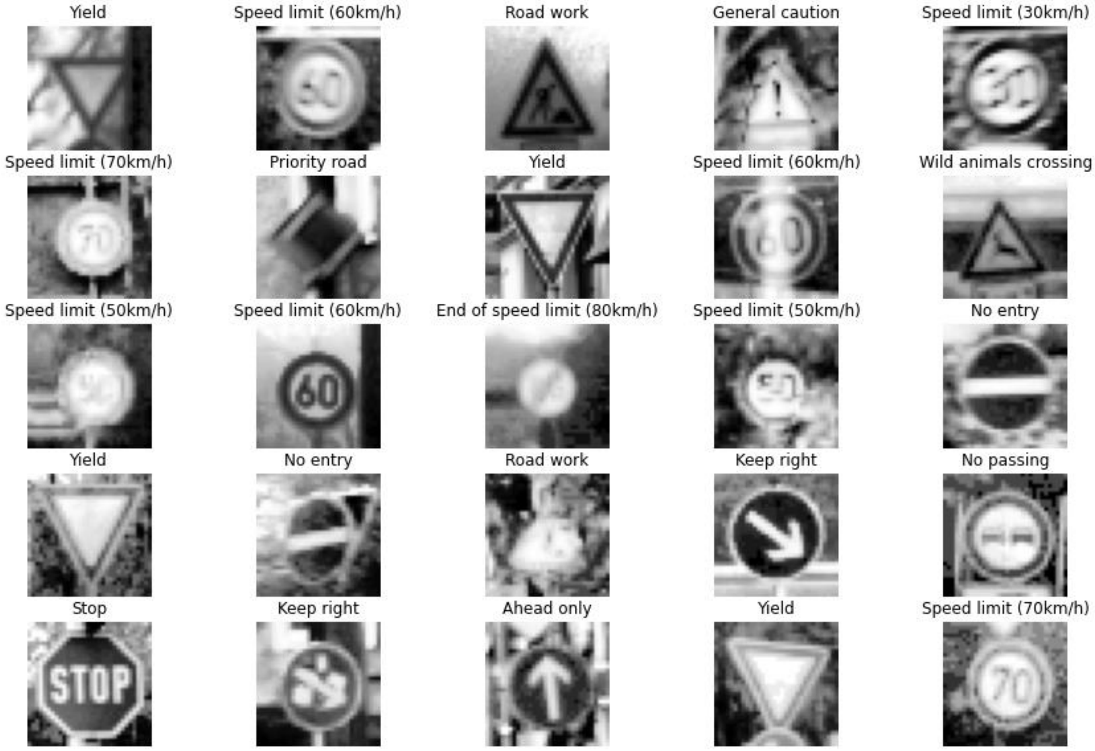
```
plt.figure(figsize=(15,10))
for i,j in enumerate (np.random.randint(0,len(train_images),16)):
    plt.subplot(4,4,i+1)
    plt.imshow(train_images[j])
    plt.axis("off")
    t=train_labels[j]
    plt.title(classes_name[t])
```

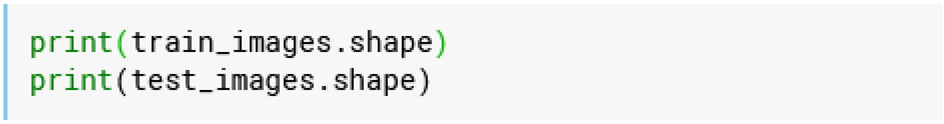


Samples of test images with their labels

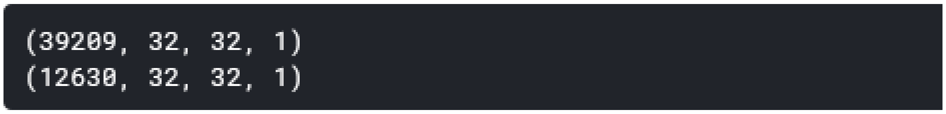


```
plt.figure(figsize=(15,10))
for i,j in enumerate (np.random.randint(0,len(test_images),25)):
    plt.subplot(5,5,i+1)
    plt.imshow(test_images[j],cmap=plt.get_cmap("gray"))
    plt.axis("off")
    t=Class_Id[j]
    plt.title(classes_name[t])
```





```
print(train_images.shape)
print(test_images.shape)
```

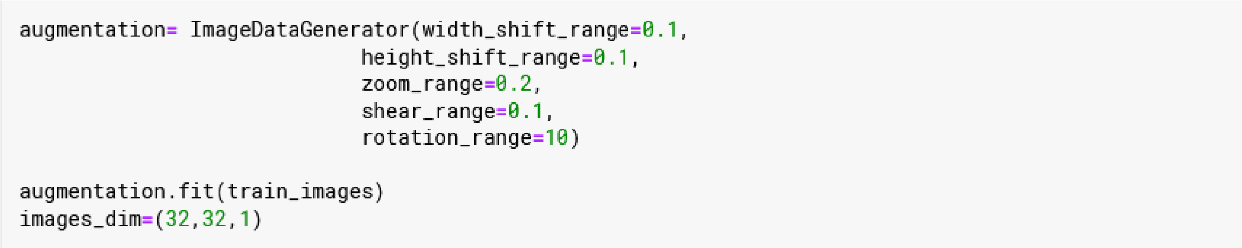


```
(39209, 32, 32, 1)
(12630, 32, 32, 1)
```

In the plot above we can see that the dataset does not contain equal number of images for each class and hence, the model may be biased in detecting some traffic signs more accurately than other. I decided to use the data augmentation with parameters

- height_shift_range =0.1 (means 10%).
- zoom_range=0.2 (0.2 MEANS CAN GO FROM 0.8 TO 1.2).
- shear_range (MAGNITUDE OF SHEAR ANGLE).

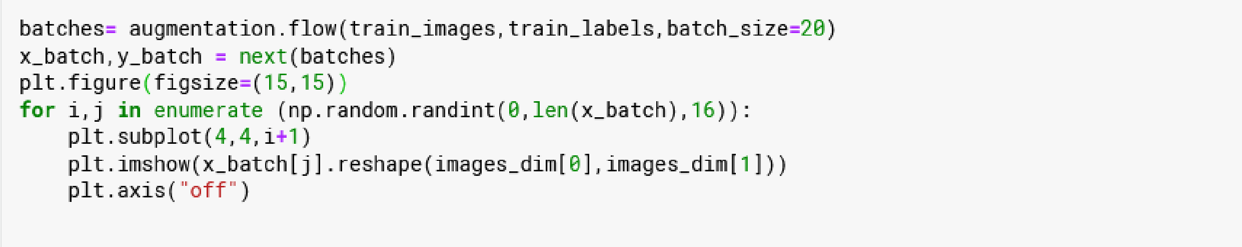
- rotation_range=10 (DEGREES).



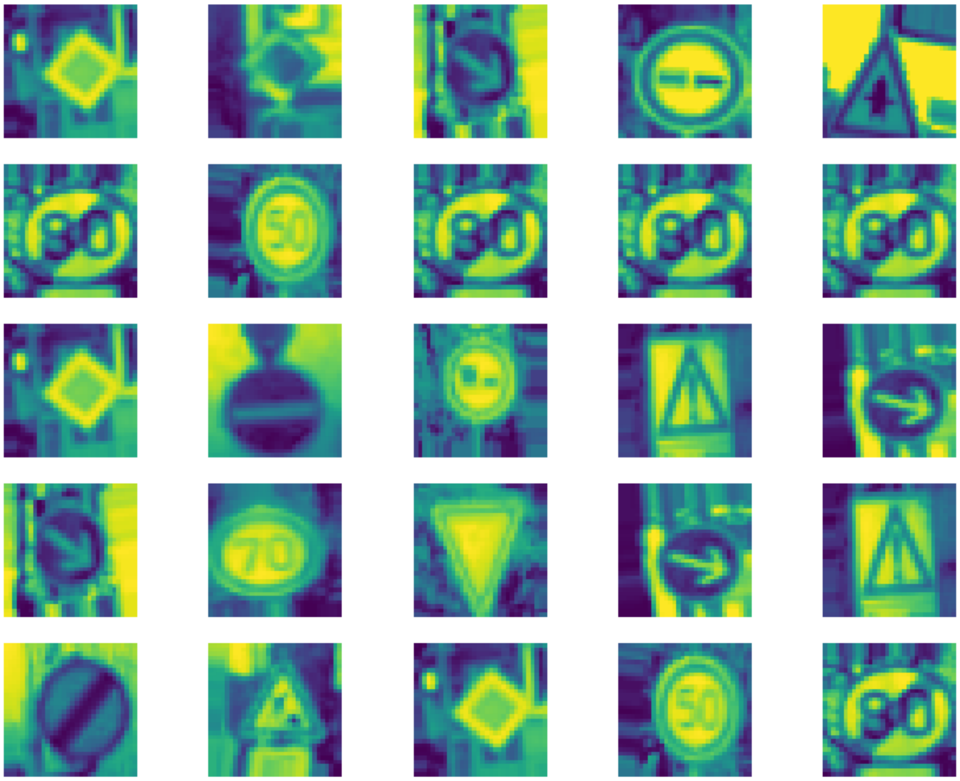
```
augmentation= ImageDataGenerator(width_shift_range=0.1,
                                 height_shift_range=0.1,
                                 zoom_range=0.2,
                                 shear_range=0.1,
                                 rotation_range=10)

augmentation.fit(train_images)
images_dim=(32,32,1)
```

Sample of augmented images:

REQUESTING DATA GENRATOR TO GENERATE IMAGES BATCH SIZE = NO. OF IMAGES CREAED EACH TIME ITS CALLED

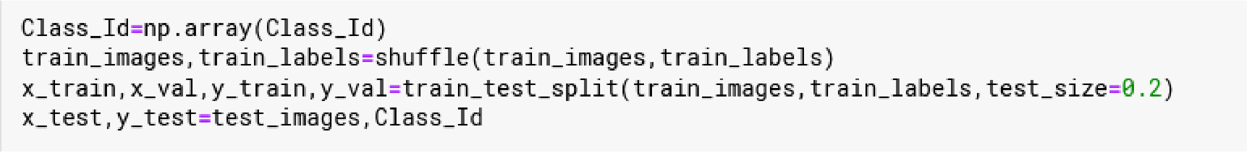


```
batches= augmentation.flow(train_images,train_labels,batch_size=20)
x_batch,y_batch = next(batches)
plt.figure(figsize=(15,15))
for i,j in enumerate (np.random.randint(0,len(x_batch),16)):
    plt.subplot(4,4,i+1)
    plt.imshow(x_batch[j].reshape(images_dim[0],images_dim[1]))
    plt.axis("off")
```



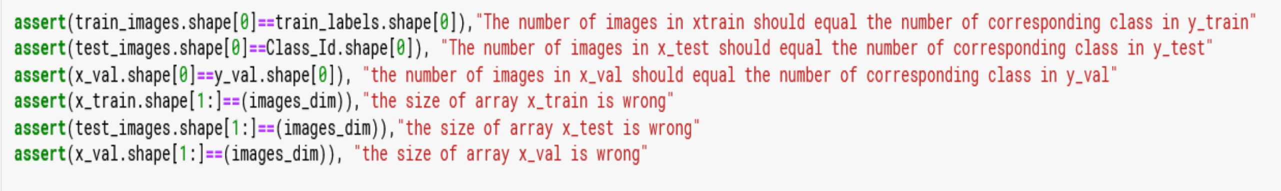
4.4 Split the dataset

Now we need to divide train images into training and validation set. Test set is already existing in separate file (12630 images). We shuffled the images with corresponding labels before splitting.



```
Class_Id=np.array(Class_Id)
train_images,train_labels=shuffle(train_images,train_labels)
x_train,x_val,y_train,y_val=train_test_split(train_images,train_labels,test_size=0.2)
x_test,y_test=test_images,Class_Id
```

Before insert the images inside keras, we should be sure that we have the same shape for images and their labels.



```
assert(train_images.shape[0]==train_labels.shape[0]),"The number of images in xtrain should equal the number of corresponding class in y_train"
assert(test_images.shape[0]==Class_Id.shape[0]), "The number of images in x_test should equal the number of corresponding class in y_test"
assert(x_val.shape[0]==y_val.shape[0]), "the number of images in x_val should equal the number of corresponding class in y_val"
assert(x_train.shape[1:]==(images_dim)),"the size of array x_train is wrong"
assert(test_images.shape[1:]==(images_dim)),"the size of array x_test is wrong"
assert(x_val.shape[1:]==(images_dim)), "the size of array x_val is wrong"
```

4.5 Build a CNN model to recognize the traffic signs

We create a CNN model by ([Conv2D, Conv2D, MaxPooling2D, Normalization], [Conv2D,Conv2D,MaxPooling2D,Normalization,Dropout], [Flatten ,Dense, Normalization ,Dropout]). As shown in the Figure 31.

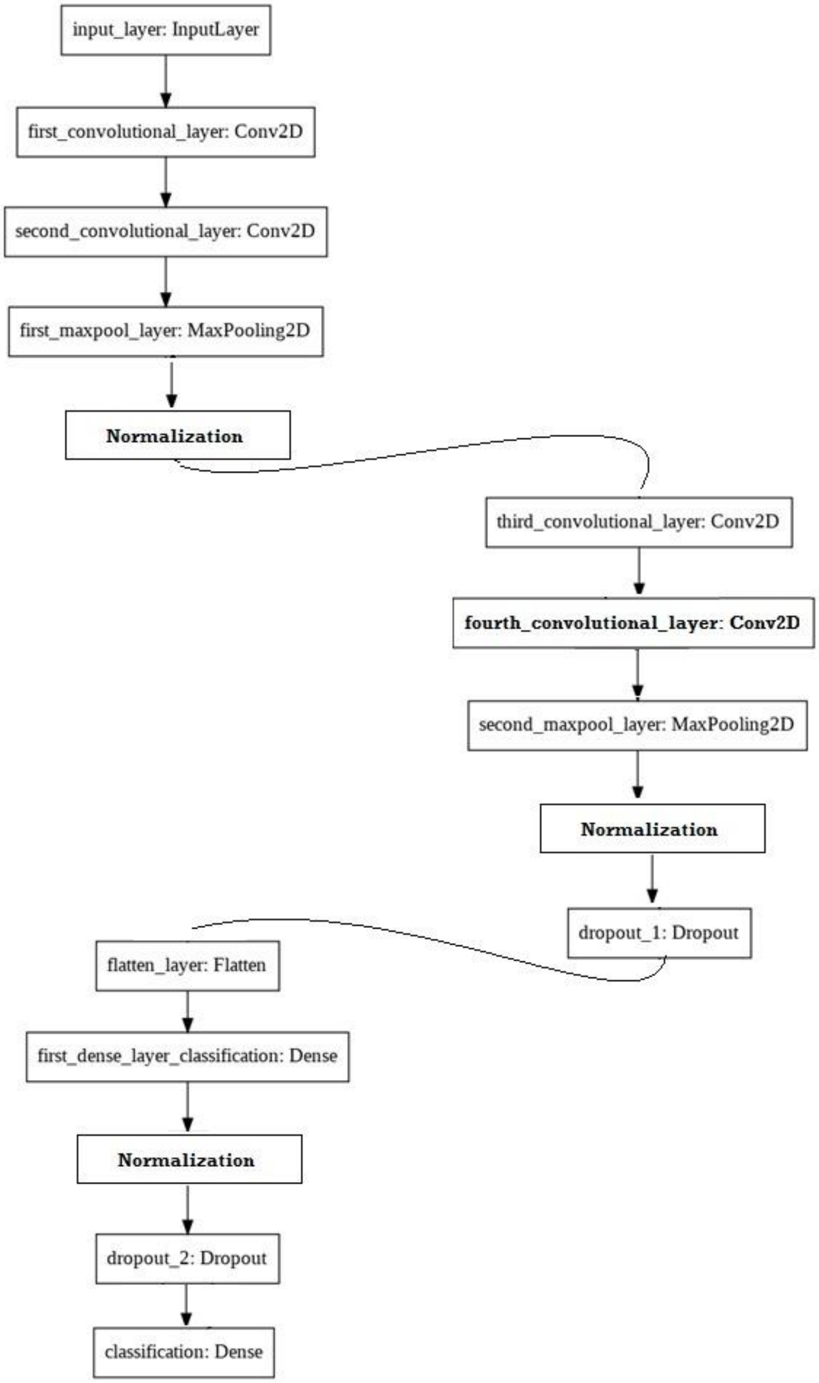
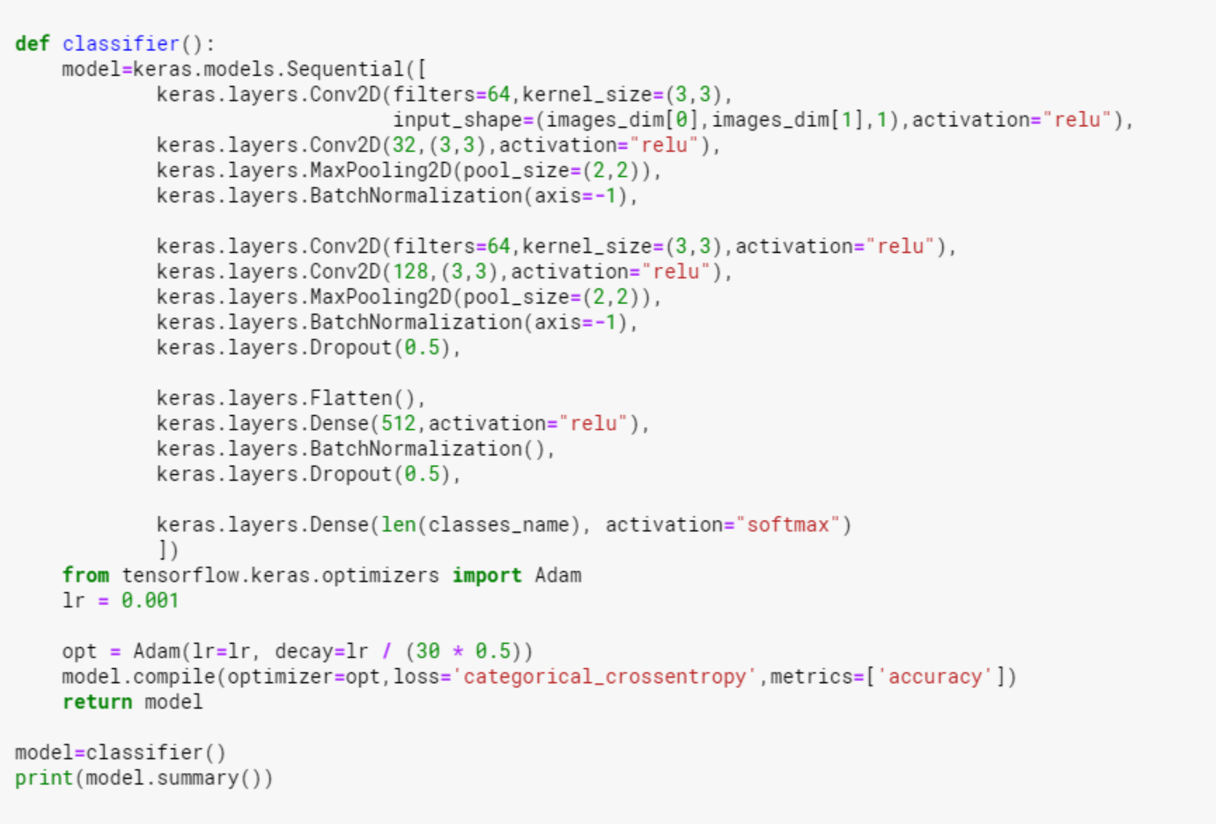
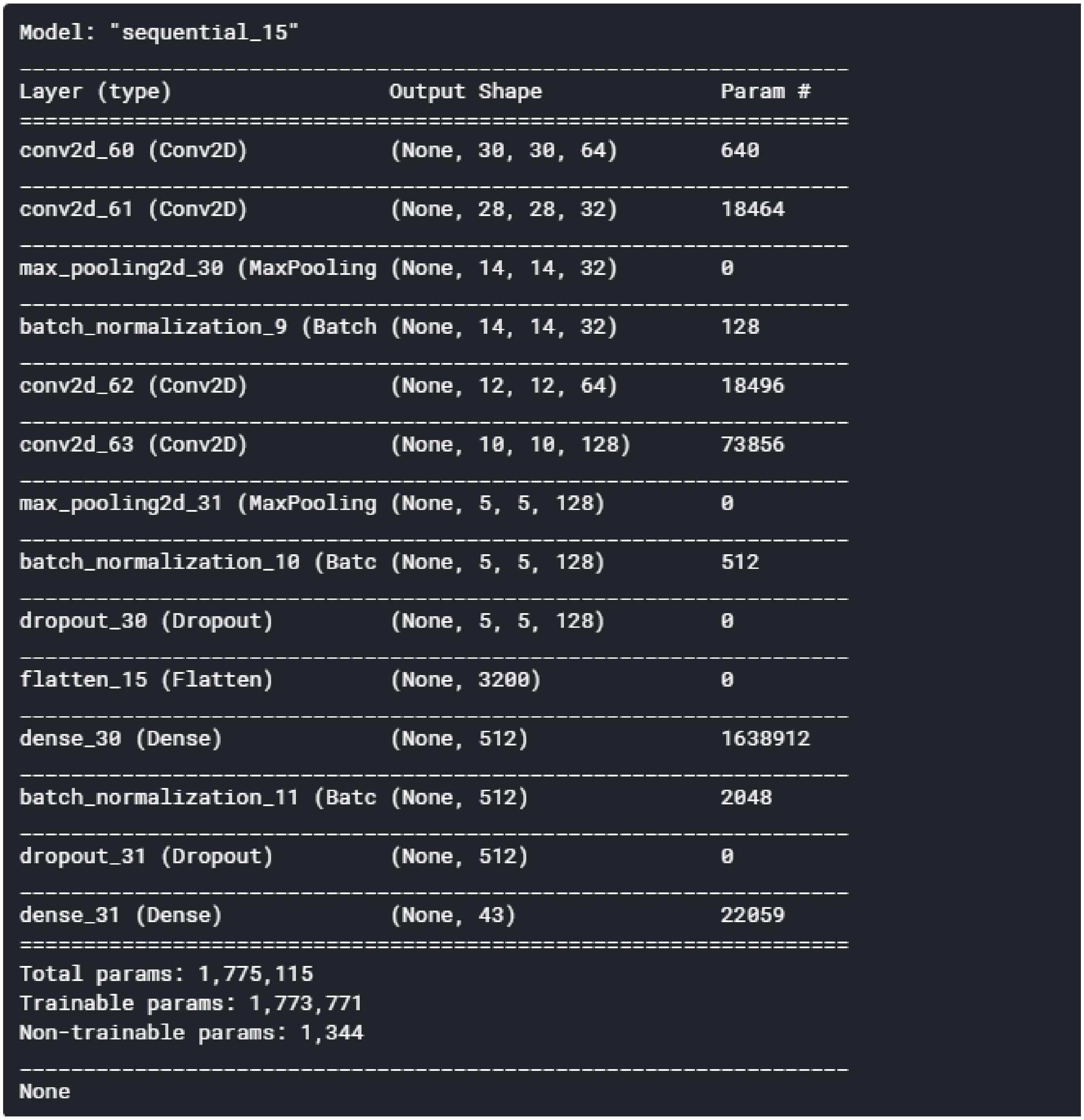

input_layer: InputLayer
first_convolutional_layer: Conv2D
second_convolutional_layer: Conv2D
first_maxpool_layer: MaxPooling2D
Normalization
third_convolutional_layer: Conv2D
fourth_convolutional_layer: Conv2D
second_maxpool_layer: MaxPooling2D
Normalization
dropout_1: Dropout
flatten_layer: Flatten
first_dense_layer_classification: Dense
Normalization
dropout_2: Dropout
classification: Dense

Figure 32. My CNN model

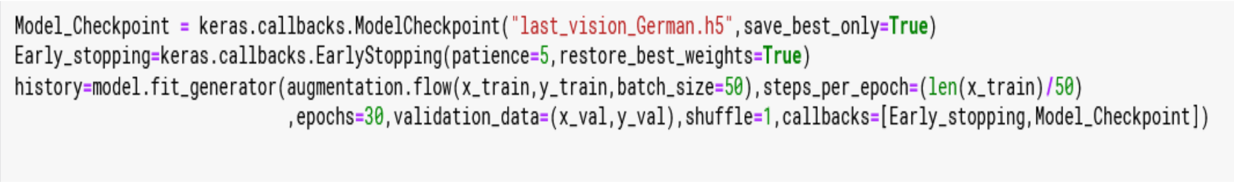


```
def classifier():
    model=keras.models.Sequential([
            keras.layers.Conv2D(filters=64,kernel_size=(3,3),
                                input_shape=(images_dim[0],images_dim[1],1),activation="relu"),
            keras.layers.Conv2D(32,(3,3),activation="relu"),
            keras.layers.MaxPooling2D(pool_size=(2,2)),
            keras.layers.BatchNormalization(axis=-1),

            keras.layers.Conv2D(filters=64,kernel_size=(3,3),activation="relu"),
            keras.layers.Conv2D(128,(3,3),activation="relu"),
            keras.layers.MaxPooling2D(pool_size=(2,2)),
            keras.layers.BatchNormalization(axis=-1),
            keras.layers.Dropout(0.5),

            keras.layers.Flatten(),
            keras.layers.Dense(512,activation="relu"),
            keras.layers.BatchNormalization(),
            keras.layers.Dropout(0.5),

            keras.layers.Dense(len(classes_name), activation="softmax")
            ])
    from tensorflow.keras.optimizers import Adam
    lr = 0.001

    opt = Adam(lr=lr, decay=lr / (30 * 0.5))
    model.compile(optimizer=opt,loss='categorical_crossentropy',metrics=['accuracy'])
    return model

model=classifier()
print(model.summary())
```



```
Model: "sequential_15"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_60 (Conv2D)           (None, 30, 30, 64)        640       
_________________________________________________________________
conv2d_61 (Conv2D)           (None, 28, 28, 32)        18464     
_________________________________________________________________
max_pooling2d_30 (MaxPooling (None, 14, 14, 32)        0         
_________________________________________________________________
batch_normalization_9 (Batch (None, 14, 14, 32)        128       
_________________________________________________________________
conv2d_62 (Conv2D)           (None, 12, 12, 64)        18496     
_________________________________________________________________
conv2d_63 (Conv2D)           (None, 10, 10, 128)       73856     
_________________________________________________________________
max_pooling2d_31 (MaxPooling (None, 5, 5, 128)         0         
_________________________________________________________________
batch_normalization_10 (Batc (None, 5, 5, 128)         512       
_________________________________________________________________
dropout_30 (Dropout)         (None, 5, 5, 128)         0         
_________________________________________________________________
flatten_15 (Flatten)         (None, 3200)              0         
_________________________________________________________________
dense_30 (Dense)             (None, 512)               1638912   
_________________________________________________________________
batch_normalization_11 (Batc (None, 512)               2048      
_________________________________________________________________
dropout_31 (Dropout)         (None, 512)               0         
_________________________________________________________________
dense_31 (Dense)             (None, 43)                22059     
=================================================================
Total params: 1,775,115
Trainable params: 1,773,771
Non-trainable params: 1,344
_________________________________________________________________
None
```

4.6 Model Training



```
Model_Checkpoint = keras.callbacks.ModelCheckpoint("last_vision_German.h5",save_best_only=True)
Early_stopping=keras.callbacks.EarlyStopping(patience=5,restore_best_weights=True)
history=model.fit_generator(augmentation.flow(x_train,y_train,batch_size=50),steps_per_epoch=(len(x_train)/50)
                            ,epochs=30,validation_data=(x_val,y_val),shuffle=1,callbacks=[Early_stopping,Model_Checkpoint])
```

Note: .fit_generator method supports data augmentation. However, if you are using tensorflow==2.2.0 or tensorflow-gpu==2.2.0 (or higher), then you must use the .fit method (which now supports data augmentation)

- Batch size = it can take any integer value or NULL and by default, it will be set to 32. It specifies no. of samples per gradient.
- Epochs: an integer and number of epochs we want to train our model for.
- Steps_per_epoch: it specifies the total number of steps taken from the generator as soon as one epoch is finished, and next epoch has started.

We can calculate the value of steps_per_epoch as the total number of samples in your dataset divided by the batch size.

- callbacks : a list of callback functions applied during the training of our model.

The **fit()** method accepts a callback argument, which let us specify a list of objects that Keras call during training at the start and end of training, at the start and end of each epoch and even before and after processing each batch.

For example, the **Model Checkpoint** callback saves checkpoints of your model at regular intervals during training by default at the end of each epoch. Moreover, if you use a validation set during training, you can set **save_best_only=True** when creating the Model Checkpoint. In this case, it will only save your model when its performance on the validation set is the best so far. This way, you do not need to worry about training for too long and overfitting the training set, simply restore the last model saved after training, and this will be the best model on the validation set.

Another way to implement **early stopping** is to simply use the '**Early Stopping' callback.**It will interrupt training when it measures no progress on the validation set for several epochs (defined by the patience argument), and it will optionally rollback to the best model. You can combine both callbacks to both save checkpoints of your model (in case your computer crashes), and interrupt training early when there is no more progress (to avoid wasting time and resources). In this case, the number of epochs can be set to a large value since training will stop automatically when there is no more progress. Moreover, there is no need to restore the best model saved in this case since the **Early Stopping callback** will keep track of the best weights and restore them for us at the end of training.

- The difference between using callbacks and save method is to save the model after fitting it.

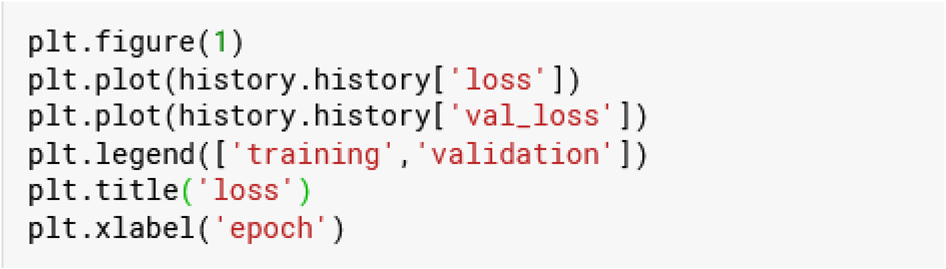
Save the model by **model.save()**, Keras will save both the model’s architecture (including every layer’s hyperparameters) and the value of all the model parameters for every layer (e.g., connection weights and biases), using the HDF5 format. It also saves the optimizer (including its hyperparameters and any state it may have). You will typically have a script that trains a model and saves it, and one or more scripts (or web services) that load the model and use it to make predictions. Loading the model is just as easy: model = keras.models.**load_model()** This will work when using the Sequential API or the Functional API, but unfortunately not when using Model subclassing. However, you can use save_weights() and load_weights() to at least save and restore the model parameters (but you will need to save and restore everything else yourself).

But what if training lasts several hours? This is quite common, especially when training on large datasets. In this case, you should not only save your model at the end of training, but also save checkpoints at regular intervals during training.

```
Epoch 1/30
274/274 [==============================] - 95s 344ms/step - loss: 2.7381 - a
ccuracy: 0.3579 - val_loss: 6.8018 - val_accuracy: 0.0493
Epoch 2/30
274/274 [==============================] - 95s 347ms/step - loss: 0.6267 - a
ccuracy: 0.8033 - val_loss: 1.8920 - val_accuracy: 0.4866
Epoch 3/30
274/274 [==============================] - 95s 346ms/step - loss: 0.3211 - a
ccuracy: 0.9043 - val_loss: 0.1220 - val_accuracy: 0.9593
Epoch 4/30
274/274 [==============================] - 95s 346ms/step - loss: 0.1943 - a
ccuracy: 0.9394 - val_loss: 0.0703 - val_accuracy: 0.9799
Epoch 5/30
274/274 [==============================] - 98s 358ms/step - loss: 0.1472 - a
ccuracy: 0.9556 - val_loss: 0.0407 - val_accuracy: 0.9862
Epoch 6/30
274/274 [==============================] - 95s 345ms/step - loss: 0.1161 - a
ccuracy: 0.9641 - val_loss: 0.0169 - val_accuracy: 0.9952
Epoch 7/30
274/274 [==============================] - 94s 342ms/step - loss: 0.1004 - a
ccuracy: 0.9696 - val_loss: 0.0400 - val_accuracy: 0.9875
Epoch 8/30
274/274 [==============================] - 94s 342ms/step - loss: 0.0938 - a
ccuracy: 0.9694 - val_loss: 0.0290 - val_accuracy: 0.9901
Epoch 9/30
274/274 [==============================] - 94s 343ms/step - loss: 0.0792 - a
ccuracy: 0.9758 - val_loss: 0.0152 - val_accuracy: 0.9949
Epoch 10/30
```

```
274/274 [==============================] - 95s 347ms/step - loss: 0.0737 - a
ccuracy: 0.9782 - val_loss: 0.0133 - val_accuracy: 0.9966
Epoch 11/30
274/274 [==============================] - 95s 346ms/step - loss: 0.0663 - a
ccuracy: 0.9777 - val_loss: 0.0128 - val_accuracy: 0.9963
Epoch 12/30
274/274 [==============================] - 97s 354ms/step - loss: 0.0635 - a
ccuracy: 0.9792 - val_loss: 0.0256 - val_accuracy: 0.9924
Epoch 13/30
274/274 [==============================] - 94s 343ms/step - loss: 0.0621 - a
ccuracy: 0.9817 - val_loss: 0.0170 - val_accuracy: 0.9936
Epoch 14/30
274/274 [==============================] - 94s 342ms/step - loss: 0.0553 - a
ccuracy: 0.9817 - val_loss: 0.0075 - val_accuracy: 0.9977
Epoch 15/30
274/274 [==============================] - 94s 343ms/step - loss: 0.0478 - a
ccuracy: 0.9841 - val_loss: 0.0098 - val_accuracy: 0.9969
Epoch 16/30
274/274 [==============================] - 94s 343ms/step - loss: 0.0466 - a
ccuracy: 0.9845 - val_loss: 0.0094 - val_accuracy: 0.9974
Epoch 17/30
274/274 [==============================] - 94s 344ms/step - loss: 0.0425 - a
ccuracy: 0.9858 - val_loss: 0.0127 - val_accuracy: 0.9961
Epoch 18/30
274/274 [==============================] - 95s 344ms/step - loss: 0.0437 - a
ccuracy: 0.9865 - val_loss: 0.0205 - val_accuracy: 0.9936
Epoch 19/30
274/274 [==============================] - 95s 345ms/step - loss: 0.0459 - a
ccuracy: 0.9851 - val_loss: 0.0075 - val_accuracy: 0.9977
```

Plot Loss function.



```
plt.figure(1)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.legend(['training','validation'])
plt.title('loss')
plt.xlabel('epoch')
```

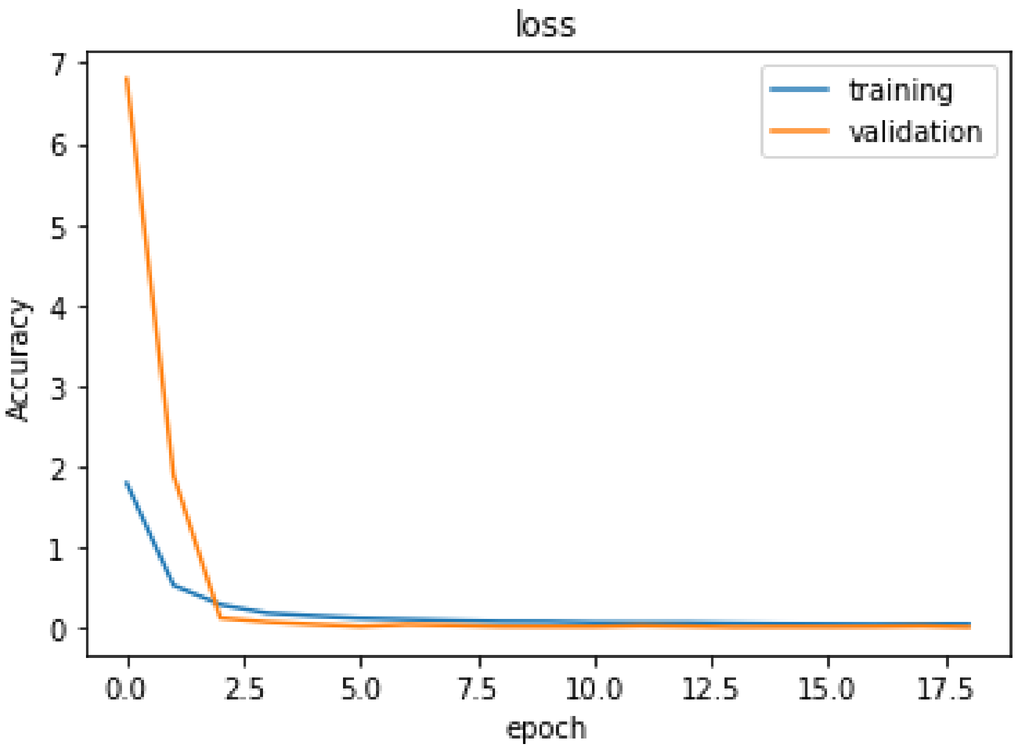
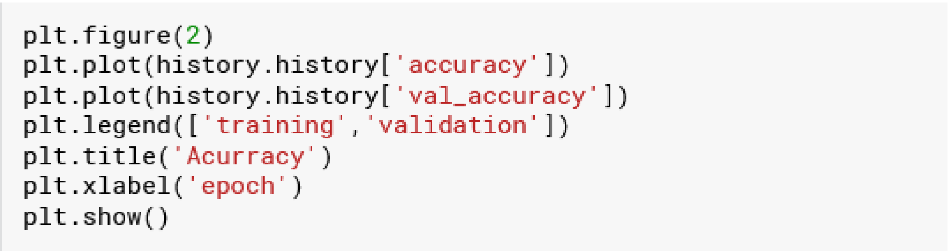


Figure 33. Loss function

Plot Accuracy



```
plt.figure(2)
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.legend(['training','validation'])
plt.title('Acurracy')
plt.xlabel('epoch')
plt.show()
```

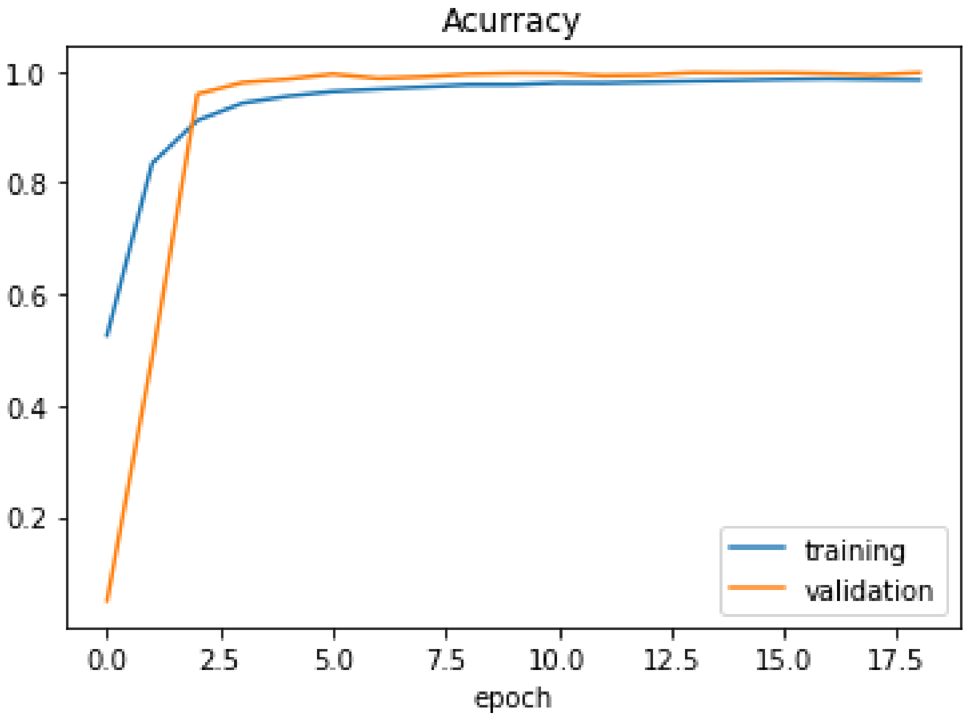


Figure 34. Accuracy

We can combine the (loss_val, loss, val_accuracy, accuracy) in one plot:

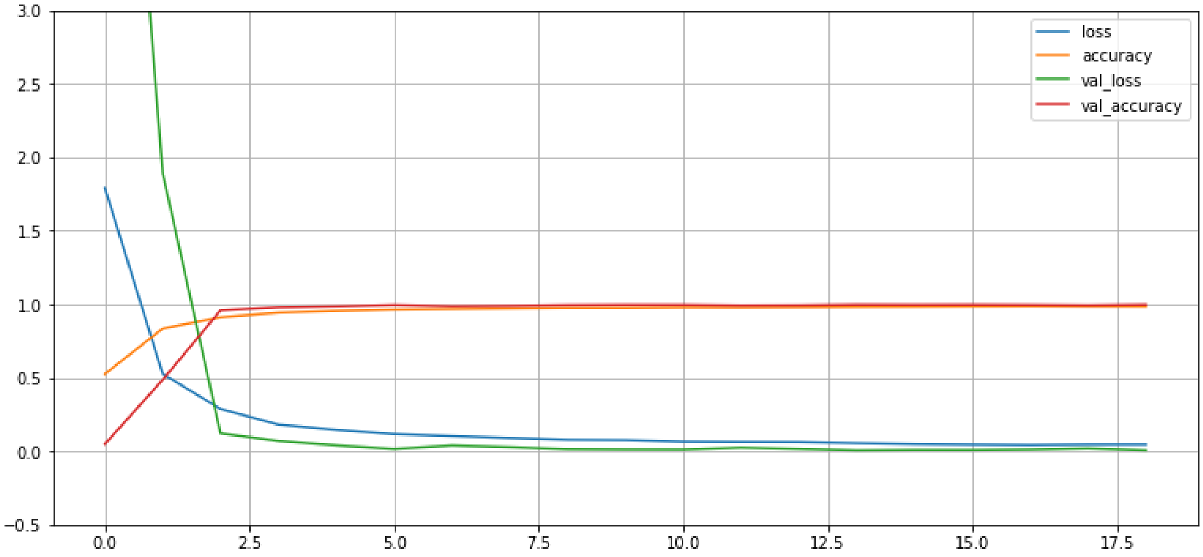


Figure 35. loss_val, loss, val_accuracy, accuracy

I consider (Accuracy and val_accuracy) as a fake accuracy, So I must check the test accuracy to know if we have an overfitting or not.

I have 97 % as a test accuracy, which means, I built a good model as CNN.

I checked the accuracy by Keras method and let's now see the accuracy of all predicted test images with the true labels by scikit-learn:

Both the accuracy measures are different:

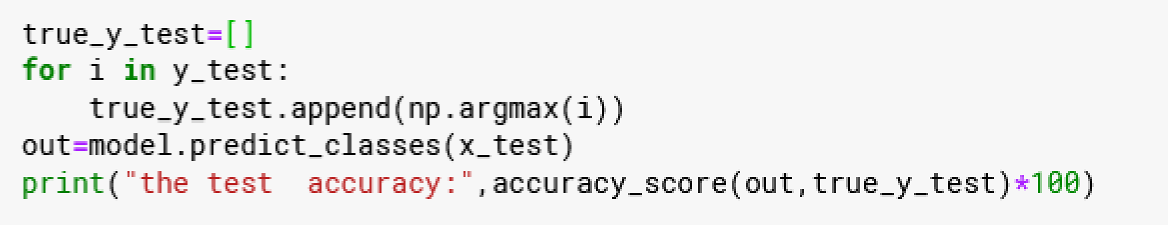
1 - sklearn accuracy is pretty straightforward.

- Predicted_values = [0, 10, 100, 1000]
- Actual_values = [0, 100, 10, 1000]
- Y_equals = [1,0,0,1]

sklearn accuracy = 0.5 which is the confidence.

2 - On the other hand, keras models' accuracy calculates the mean of y_equals for binary classes. Slightly more different for categorical.

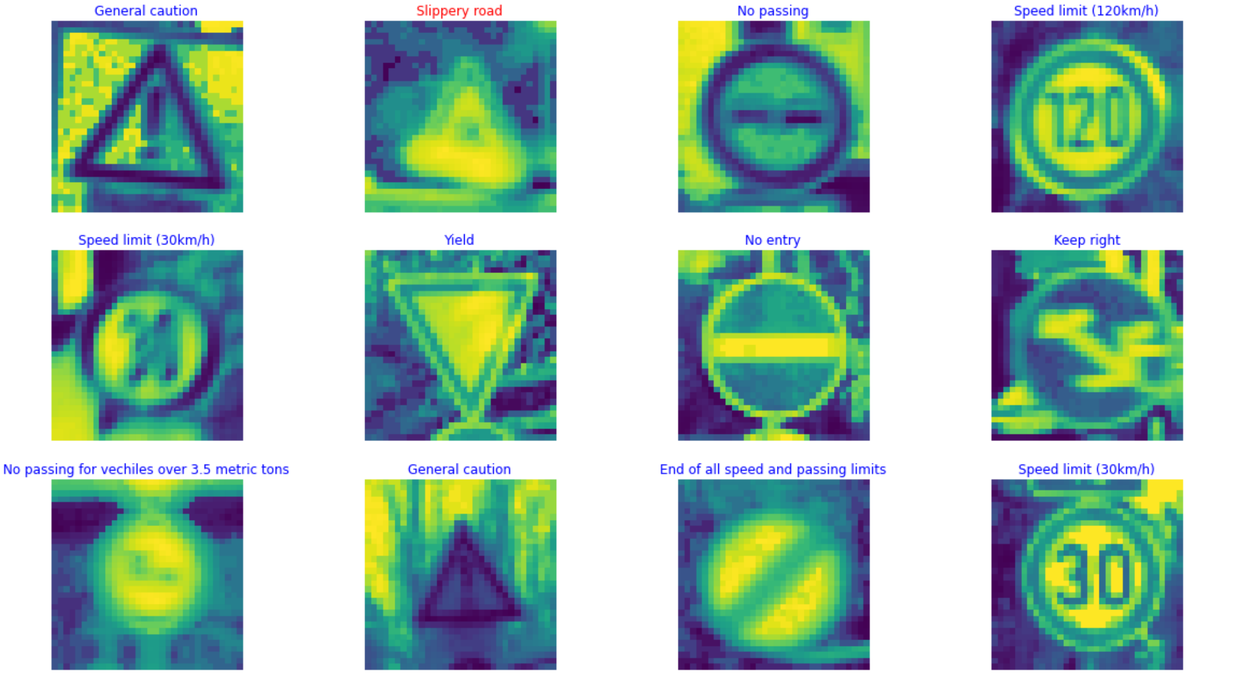
all the needed information about model evaluation in [56,57].



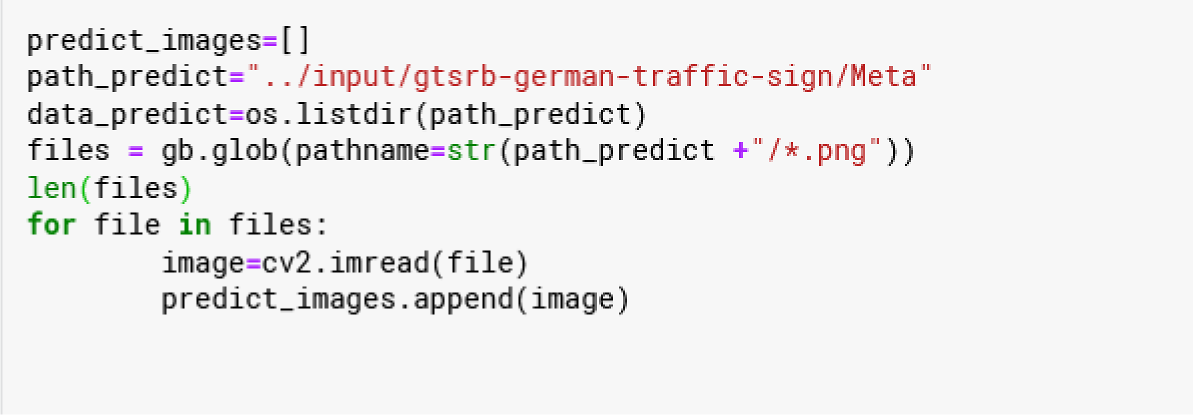
```
true_y_test=[]
for i in y_test:
    true_y_test.append(np.argmax(i))
out=model.predict_classes(x_test)
print("the test  accuracy:",accuracy_score(out,true_y_test)*100)
```

Test Accuracy: 0.9755344390869141

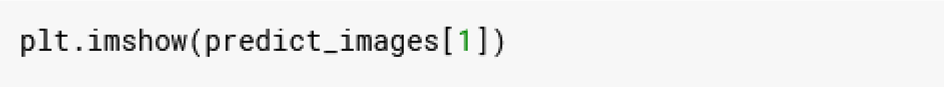
4.7 Predict a sample of test images



The datasets contain a "Meta" file with 42 different images to make the last test after using the model with external camera.

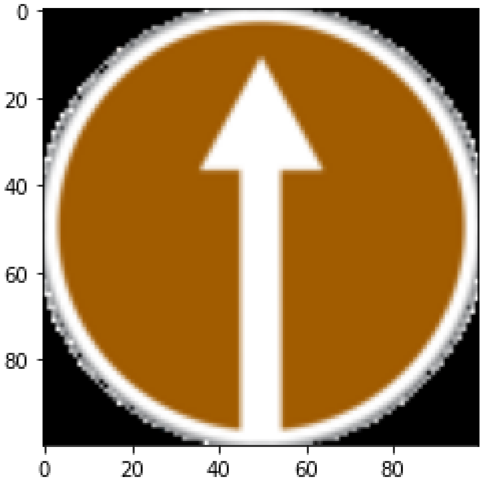


```
predict_images=[]
path_predict="../input/gtsrb-german-traffic-sign/Meta"
data_predict=os.listdir(path_predict)
files = gb.glob(pathname=str(path_predict +"/*.png"))
len(files)
for file in files:
        image=cv2.imread(file)
        predict_images.append(image)
```

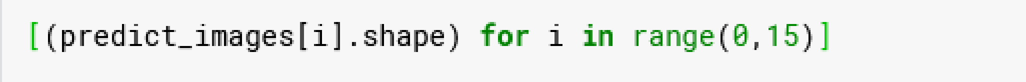


```
plt.imshow(predict_images[1])
```

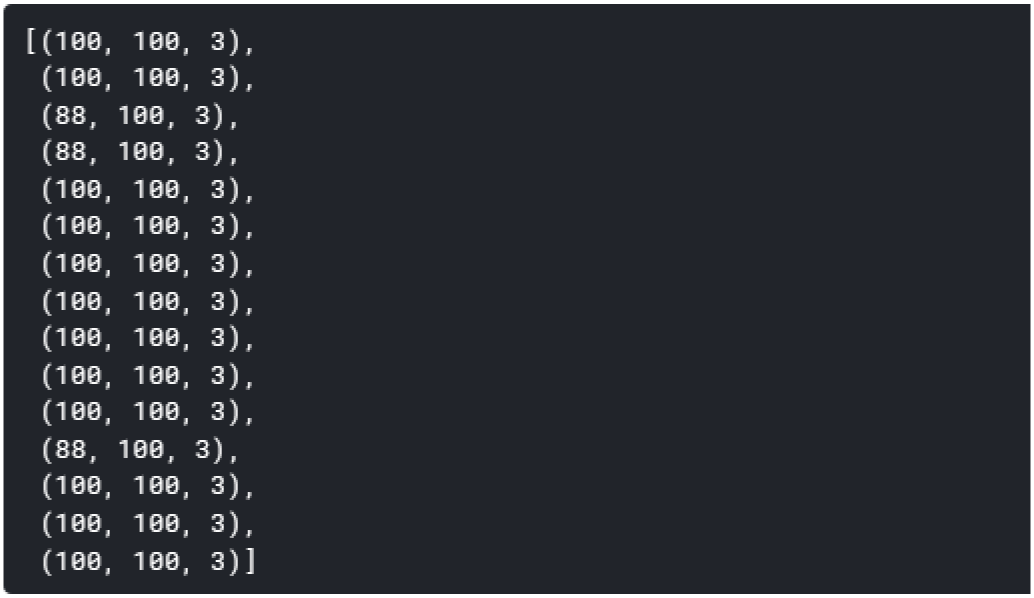
```
<matplotlib.image.AxesImage at 0x7f81b12fc1d0>
```



Almost the Meta images have a (100,100,3) dimension.

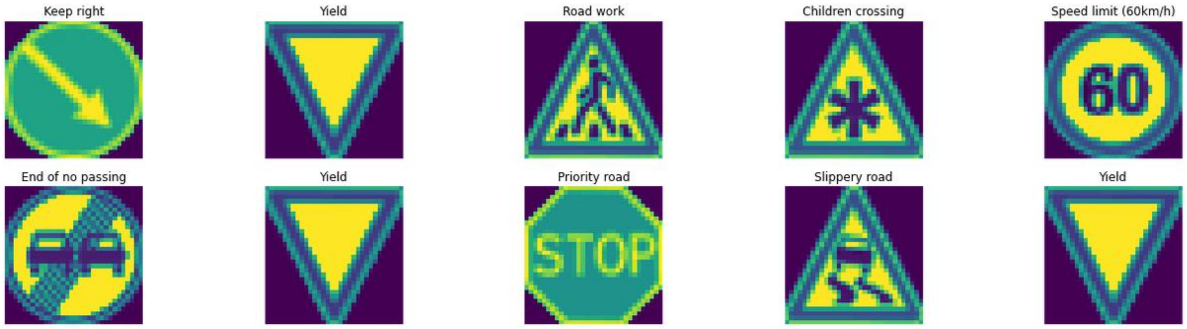


```
[(predict_images[i].shape) for i in range(0,15)]
```



```
[(100, 100, 3),
 (100, 100, 3),
 (88, 100, 3),
 (88, 100, 3),
 (100, 100, 3),
 (100, 100, 3),
 (100, 100, 3),
 (100, 100, 3),
 (100, 100, 3),
 (100, 100, 3),
 (100, 100, 3),
 (88, 100, 3),
 (100, 100, 3),
 (100, 100, 3),
 (100, 100, 3)]
```

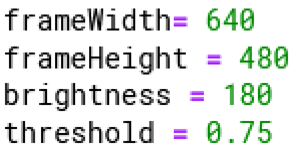
Predict a sample of Meta images



4.8 Camera Test

setting the camera:

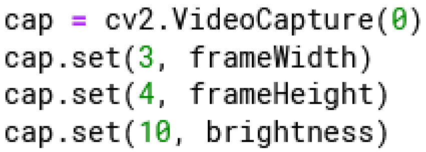
- Camera resolution:



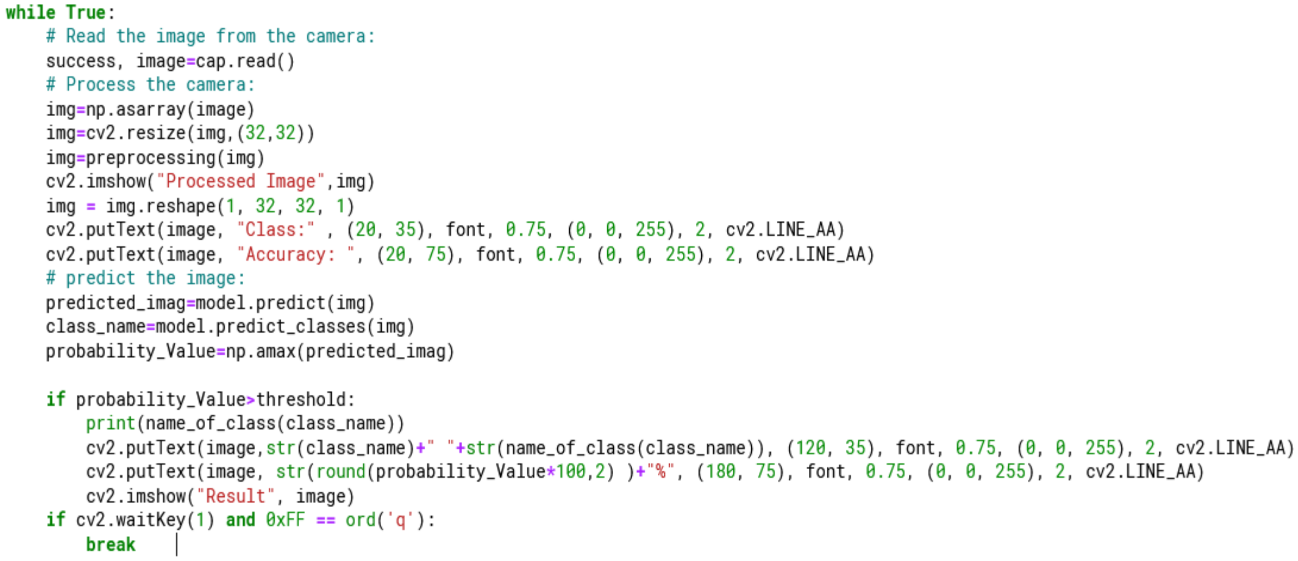
```
frameWidth= 640
frameHeight = 480
brightness = 180
threshold = 0.75
```

So, we put the PROBABLITY THRESHOLD = 0.75

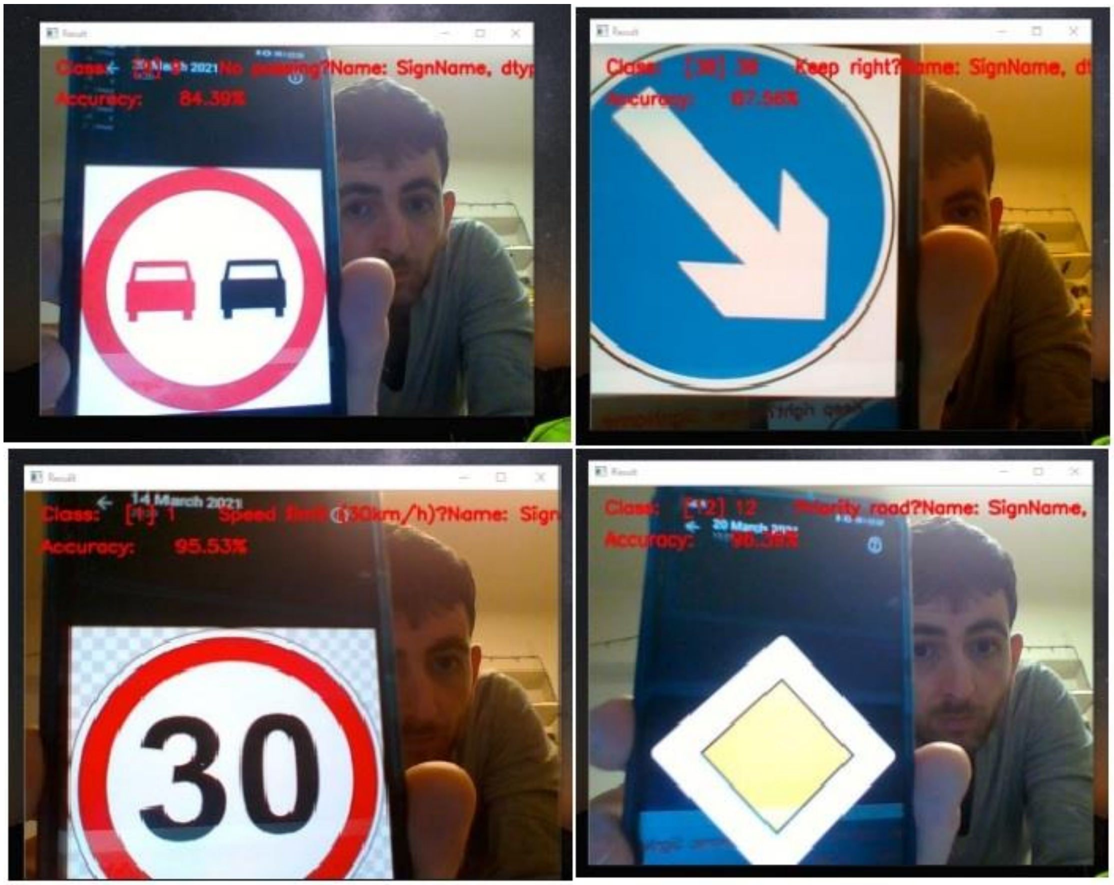
- Video camera settings:



```
cap = cv2.VideoCapture(0)
cap.set(3, frameWidth)
cap.set(4, frameHeight)
cap.set(10, brightness)
```



```
while True:
    # Read the image from the camera:
    success, image=cap.read()
    # Process the camera:
    img=np.asarray(image)
    img=cv2.resize(img,(32,32))
    img=preprocessing(img)
    cv2.imshow("Processed Image",img)
    img = img.reshape(1, 32, 32, 1)
    cv2.putText(image, "Class:" , (20, 35), font, 0.75, (0, 0, 255), 2, cv2.LINE_AA)
    cv2.putText(image, "Accuracy: ", (20, 75), font, 0.75, (0, 0, 255), 2, cv2.LINE_AA)
    # predict the image:
    predicted_imag=model.predict(img)
    class_name=model.predict_classes(img)
    probability_Value=np.amax(predicted_imag)

    if probability_Value>threshold:
        print(name_of_class(class_name))
        cv2.putText(image,str(class_name)+" "+str(name_of_class(class_name)), (120, 35), font, 0.75, (0, 0, 255), 2, cv2.LINE_AA)
        cv2.putText(image, str(round(probability_Value*100,2) )+"%", (180, 75), font, 0.75, (0, 0, 255), 2, cv2.LINE_AA)
        cv2.imshow("Result", image)
    if cv2.waitKey(1) and 0xFF == ord('q'):
        break
```



```
Class: No passing?Name: SignName, dtyp
Accuracy: 84.39%
```

```
Class: [38] 38 Keep right?Name: SignName, dt
Accuracy: 87.56%
```

```
Class: [1] 1 Speed limit (30km/h)?Name: Sign
Accuracy: 95.53%
```

```
Class: [12] 12 Priority road?Name: SignName,
Accuracy: 90.38%
```

Figure 36. camera test samples

5. Results and discussion

A bout 60000 images are resized to (32,32,1) by reduce their dimension in keras to (32,32) and convert them to grayscale type (preprocessing function).

40000 images divided into (training set and validation set) with 0.3 test size. This ratio is not a big ratio if we take in consideration the data augmentation technique which we use it to fill the lack of images in some classes. The difference of distribution of images in the number of classes may cause misleading in the prediction process. Where the model will be biased to some classes which has a big number of images. So, using data augmentation will remove this problem. However, on another hand, it is known that data augmentation is usually done only on training set and not on validation set, so increasing the value of test size ratio will make sense.

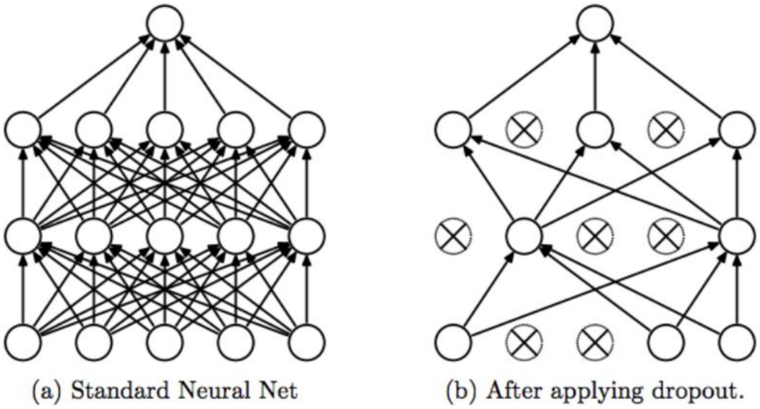
I used the Keras Sequential API, I designed a Convolutional Neural Network with two layers + Flatten layer (Figure 2) ([Conv2D, Conv2D, MaxPooling2D, normalizer], [Conv2D, Conv2D, MaxPooling2D, normalizer, Dropout], [Flatten, Dense, normalizer, Dropout]). The first convolutional (Conv2D) layer, after many experiments, I have chosen to set 2^6 = 64 and 2^5 = 32 filters respectively for the two firsts conv2D layers and 2^6 =64 and 2^7 = 128 filters respectively for the two last ones. Each filter transforms a part of the image (defined by the kernel size) using the kernel filter. The kernel filter matrix is applied on the whole image. Filters can be seen as a transformation of the image.

The second important layer in the first part of my network is "MaxPool2D" layer. This layer simply acts as a down sampling filter. It looks at the 2 neighboring pixels and picks the maximal values. These are used to reduce computational cost, and to some extent also reduce overfitting in somehow. We have to choose the pooling size (i.e., the area size pooled each time) more the pooling dimension is high, more the down sampling is important.

When combining convolutional and pooling layers, CNN are able to combine local features and learn more global features of the image.

Batch Norm is a normalization technique done between the layers of a Neural Network instead of in the raw data. It is done along mini batches instead of the full data set. It serves to speed up training and use higher learning rates, making learning easier.

The Dropout technique is a regularization method, Dropout is a technique used to prevent a model from overfitting. Dropout works by randomly setting the outgoing edges of hidden units (neurons that make up hidden layers) to zero at each update of the training phase.


(a) Standard Neural Net

(b) After applying dropout.

I used a “ReLU” as a rectifier (activation function max (0, x)). It is the most used activation function especially in the convolutional neural networks. The rectifier activation function is used to add nonlinearity to the network.

Flattening and fully connected layers: Flattening is converting the data into a 1-dimensional array for inputting it to the next layer. We flatten the output of the convolutional layers to create a single long feature vector. And it is connected to the final classification model, which is called a fully connected layer (we put all the pixel data in one line and make connections with the final layer).

At the end I used the features in two fully-connected (Dense) layers which is just an artificial neural networks (ANN) classifier. In the last layer (Dense(len(classes_name), activation="softmax") the network outputs distribution of probability of each class where we have 43 different classes.

Once the layers are added to the model, we need to set up a score function, a loss function and optimization algorithm.

We apply a loss function to measure the performance of model (i.e., how poorly our model performs on images with known labels.) It is the error rate between the predicted images and the true images.

The most important part is choosing the optimizer, this function will iteratively improve the parameters (filters kernel values, weights and bias of neurons and so on….).

Adam optimization algorithm is used in this project with default value for l_r= 0.001 and 30 epochs, Adam is an optimization algorithm that can be used instead of the classical stochastic gradient descent procedure to update network weights iterative based in training data.

Adam realizes the benefits of both AdaGrad and RMSProp. Instead of adapting the parameter learning rates based on the average first moment (the mean) as in RMSProp, Adam also makes use of the average of the second moments of the gradients (the uncentered variance).

Finally, the metric function "accuracy" is used to evaluate the performance. It plays the same role of loss function, except that the results from the metric evaluation are not used when training the model (However, Only for evaluation.)

6. Conclusion

During this work, we talked about convolutional neural networks and what improvements and additions to their architecture have been made. An overview of the areas of their use and their most widespread open-source programs were presented.

In practical part, a classification Convolutional neural network is designed to classify German traffic signs dataset. High training and test accuracy by simple CNN is obtained with about 60,000 images as training, testing and validation sets. Two layers + Flatten layer is used only to achieve more than 97 % as a test accuracy with 0.0075 as val_loss. Callbacks features in Keras is used also to help in reducing the training process time with "Earlystopping" feature which stopped the process after only 19 epochs. The algorithm showed high performance when I applied it for unknown images in Meta file which contains a different label images with different dimensions.

The large ability for the algorithm to deal with real experiment with great predictability was clear when an external camera is used to prove the efficiency the suggested network in real life. The idea is scalable in the future by adding more complicated tasks and goals for example to detect objects and recognize faces.

7. References

[1] Bio-inspired artificial intelligence: theories, methods, and technologies /Dario Floreano and Claudio Mattiussi , ISBN 978-0-262-06271-8 (hardcover : alk. paper)

[2] Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, 2nd Edition by Aurélien Géron Released September 2019 Publisher(s): O'Reilly Media, Inc.ISBN: 9781492032649

[3]Rockwell Anyoha, The History of Artificial Intelligence [online] https://sitn.hms.harvard.edu/flash/2017/history-artificial-intelligence/ , Accessed [20.02.2021]

[4] Open source https://scikit-learn.org/stable/modules/neural_networks_supervised.html [online]

[5] Russell, S. & Norvig, P., 2009, Artificial Intelligence: A Modern Approach 3rd edition, Saddle River, NJ: Prentice Hall.

[6] Tyler Elliot Bettilyon, Introduction To Deep Learning, [online] https://medium.com/tebs-lab/introduction-to-deep-learning-a46e92cb0022 , Accessed [20.02.2021]

[7] https://www.mathworks.com/discovery/deep-learning.html [online]

[8] https://www.researchgate.net/figure/Shallow-MLP-vs-deep-MLP-57_fig1_332165523

[9] SAGAR SHARMA , [online] https://towardsdatascience.com/what-the-hell-is-perceptron-626217814f53 , Accessed [22.02.2021]

[10] Machine learning yearning: Technical strategy for ai engineers in the era of deep learning

A Ng - Retrieved online at https://www. mlyearning. org, 2019

[11] Y. Lecun, L. Bottou, Y. Bengio and P. Haffner, "Gradient-based learning applied to document recognition," in Proceedings of the IEEE, vol. 86, no. 11, pp. 2278-2324, Nov. 1998, doi: 10.1109/5.726791.

[12] Azel Daniel, Understanding LeNet: A Detailed Walkthrough, [online], https://towardsdatascience.com/understanding-lenet-a-detailed-walkthrough-17833d4bd155

[13]Paul-Louis Pröve, Types of Convolutions in Deep Learning, [online]
https://towardsdatascience.com/types-of-convolutions-in-deep-learning-717013397f4d

[14] Krizhevsky, Alex & Sutskever, Ilya & Hinton, Geoffrey. (2012). ImageNet Classification with Deep Convolutional Neural Networks. Neural Information Processing Systems. 25. 10.1145/3065386.

[15] Simonyan, Karen & Zisserman, Andrew. (2014). Very Deep Convolutional Networks for Large-Scale Image Recognition. arXiv 1409.1556.

[16] Nash, Will & Drummond, Tom & Birbilis, Nick. (2018). A review of deep learning in the study of materials degradation. npj Materials Degradation. 2. 10.1038/s41529-018-0058-x.

[17] C. Szegedy et al., "Going deeper with convolutions," 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Boston, MA, USA, 2015, pp. 1-9, doi: 10.1109/CVPR.2015.7298594.

[18] Min Lin, Qiang Chen, and Shuicheng Yan. Network in network. CoRR, abs/1312.4400, 2013.

[19] Szegedy, Christian & Vanhoucke, Vincent & Ioffe, Sergey & Shlens, Jon & Wojna, ZB. (2016). Rethinking the Inception Architecture for Computer Vision. 10.1109/CVPR.2016.308.

[20] Bharath Raj, A Simple Guide to the Versions of the Inception Network , [online]https://towardsdatascience.com/a-simple-guide-to-the-versions-of-the-inception-network-7fc52b863202 , Accessed [25.02.2021]

[21] Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude

T Tieleman, G Hinton - COURSERA: Neural networks for machine learning, 2012

[22] Ioffe, Sergey, and Christian Szegedy. "Batch normalization: accelerating deep network training by reducing internal covariate shift. arXiv e-prints." (2015).

[23] He, Kaiming, et al. "Deep residual learning for image recognition." Proceedings of the IEEE conference on computer vision and pattern recognition. 2016.

[24] Chollet, Francois. (2017). Xception: Deep Learning with Depthwise Separable Convolutions. 1800-1807. 10.1109/CVPR.2017.195.

[25] Guo, Yunhui & Li, Yandong & Wang, Liqiang & Rosing, Tajana. (2019). Depthwise Convolution Is All You Need for Learning Multiple Visual Domains. Proceedings of the AAAI Conference on Artificial Intelligence. 33. 8368-8375. 10.1609/aaai.v33i01.33018368.

[26] Szegedy, Christian & Ioffe, Sergey & Vanhoucke, Vincent & Alemi, Alexander. (2016). Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning. AAAI Conference on Artificial Intelligence.

[27] S. Xie, R. Girshick, P. Dollár, Z. Tu and K. He, "Aggregated Residual Transformations for Deep Neural Networks," 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Honolulu, HI, USA, 2017, pp. 5987-5995, doi: 10.1109/CVPR.2017.634

[28] J. Hu, L. Shen and G. Sun, "Squeeze-and-Excitation Networks," 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, USA, 2018, pp. 7132-7141, doi: 10.1109/CVPR.2018.00745.

[29] Zoph, Barret & Vasudevan, Vijay & Shlens, Jonathon & Le, Quoc. (2018). Learning Transferable Architectures for Scalable Image Recognition. 8697-8710. 10.1109/CVPR.2018.00907.

[30] S. Bianco, R. Cadene, L. Celona and P. Napoletano, "Benchmark Analysis of Representative Deep Neural Network Architectures," in IEEE Access, vol. 6, pp. 64270-64277, 2018, doi: 10.1109/ACCESS.2018.2877890.

[31] Sik-Ho Tsang, From GoogLeNet, Merged with ResNet Idea (Image Classification) https://towardsdatascience.com/review-inception-v4-evolved-from-googlenet-merged-with-resnet-idea-image-classification-5e8c339d18bc [online] , accessed [01.03.2021]

[32] Mei, Wang & Deng, Weihong. (2018). Deep Face Recognition: A Survey. Neurocomputing. 429. 10.1016/j.neucom.2020.10.081.

[33] Parkhi, Omkar M., Andrea Vedaldi, and Andrew Zisserman. "Deep face recognition." (2015). Parkhi, Omkar M., Andrea Vedaldi, and Andrew Zisserman. "Deep face recognition." (2015).

[34] Sermanet, Pierre, et al. "Overfeat: Integrated recognition, localization and detection using convolutional networks." arXiv preprint arXiv:1312.6229 (2013).

[35] Girshick, Ross, et al. "Rich feature hierarchies for accurate object detection and semantic segmentation." Proceedings of the IEEE conference on computer vision and pattern recognition. 2014.

[36] Rohith Gandhi, R-CNN, Fast R-CNN, Faster R-CNN, YOLO — Object Detection Algorithms,[online], https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e ,accessed [10.03.2021]

[37] Gu, Jiuxiang, et al. "Recent advances in convolutional neural networks." Pattern Recognition 77 (2018): 354-377.

[38] A. Toshev and C. Szegedy ,Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2014), pp. 1653-1660

[39] Chen, Linkai, et al. "An algorithm for highway vehicle detection based on convolutional neural network." Eurasip Journal on Image and Video Processing 2018.1 (2018): 1-7.

[40] Othman, K.M.; Rad, A.B. An Indoor Room Classification System for Social Robots via Integration of CNN and ECOC. Appl. Sci. 2019, 9, 470. https://doi.org/10.3390/app9030470

[41] Yin, Wenpeng, et al. "Comparative study of CNN and RNN for natural language processing." arXiv preprint arXiv:1702.01923 (2017).

[42] Ashwin Bhandare#1, Maithili Bhide, Pranav Gokhale, Rohan Chandavarkar, Applications of Convolutional Neural Networks , [online], https://ijcsit.com/docs/Volume%207/vol7issue5/ijcsit20160705014.pdf

[43] J. Huang, J. Li and Y. Gong, "An analysis of convolutional neural networks for speech recognition," 2015 IEEE International Conference on Acoustics, Speech and Signal Processing

(ICASSP), South Brisbane, QLD, Australia, 2015, pp. 4989-4993, doi: 10.1109/ICASSP.2015.7178920.

[44] Hironobu Fujiyoshi, Tsubasa Hirakawa, Takayoshi Yamashita,Deep learning-based image recognition for autonomous driving,IATSS Research,Volume 43, Issue 4,2019,Pages 244-252,ISSN 0386-1112,https://doi.org/10.1016/j.iatssr.2019.11.008

[45] Bojarski, M., Del Testa, D., Dworakowski, D., Firner, B., Flepp, B., Goyal, P., Jackel, L.D., Monfort, M., Muller, U., Zhang, J. and Zhang, X., 2016. End to end learning for self-driving cars. arXiv preprint arXiv:1604.07316.

[46] Kumra, Sulabh, and Christopher Kanan. "Robotic grasp detection using deep convolutional neural networks." 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). IEEE, 2017.

[47] Cruz, Edmanuel & Rangel Ortiz, Jose & Gomez-Donoso, Francisco & Bauer, Zuria & Cazorla, Miguel & Rodríguez, José. (2018). Finding the Place: How to Train and Use Convolutional Neural Networks for a Dynamically Learning Robot. 1-8. 10.1109/IJCNN.2018.8489469.

[48] Yamashita R, Nishio M, Do RKG, Togashi K. Convolutional neural networks: an overview and application in radiology. Insights Imaging. 2018;9(4):611-629. doi:10.1007/s13244-018-0639-9

[50] Wang, L., Lin, Z.Q. & Wong, A. COVID-Net: a tailored deep convolutional neural network design for detection of COVID-19 cases from chest X-ray images. Sci Rep 10, 19549 (2020). https://doi.org/10.1038/s41598-020-76550-z

[51] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. Yolov4: Optimal speed and accuracy of object detection, 2020.

[52] Li Shen, L. R. Margolies, J. H. Rothstein, E. Fluder, R. McBride, and W. Sieh. Deep learning to improve breast cancer detection on screening mammography. Nature Research, 2019.

[53] Han, Kai, et al. "Ghostnet: More features from cheap operations." Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2020.

[54] Nguyen, G., Dlugolinsky, S., Bobák, M. et al. Machine Learning and Deep Learning frameworks and libraries for large-scale data mining: a survey. Artif Intell Rev 52, 77–124 (2019). https://doi.org/10.1007/s10462-018-09679-z

[55] Gogul, I. & Kumar, Sathiesh. (2017). Flower species recognition system using convolution neural networks and transfer learning. 1-6. 10.1109/ICSCN.2017.8085675.

[56] scikit-learn.org/stable/modules/generated/sklearn.metrics.accuracy_score.html,[online]

[57] https://datascience.stackexchange.com/a/14742 [online]

List of Figures

List of Tables

List of abbreviations

CNN: Convolutional Neural Network

ANN: Artificial Neural Network

DNN: Deep Neural Network

RNN: Recurrent Neural Network

NLP: Natural Language Processing

FR: Face Recognition

LFW: Labeled Face in the Wild

OS: Operating System

GTSRB: German Traffic Sign Recognition Benchmark

DL: Deep Learning