ABSTRACT

The purpose of this work is to create and develop the approach on structural parametric synthesis of convolutional neural network to receive the unique CNN architecture with the good image recognition accuracy.

The paper deals with the methods of processing and classification of graphic images using convolutional neural networks and mathematical algorithms for their support. Using researches, there shown that for the proper use of such system it's requires compliance with special technical conditions.

Today, in modern convolutional neural networks for the independent processing of graphic data there is a problem of lack of accuracy in the selection of special criteria. The urgency of this problem over time is only increasing due to the proliferation of the problem of digital identification.

In order to increase the accuracy of the results of the work, there designed system includes the algorithm of input data preparation, generating the neural network architecture and configuration its global and local parameters with means of structural parametric synthesis algorithms. Also, there were done relative surveys and tests as well as implemented all the algorithms by means of programming.

List of abbreviations

CNN – Convolutional neural network

GA – genetic algorithm

SE – squeeze and excitation

AGI – artificial general intelligence

LT – logic theorist

RU – residual unit

ASIC – application specific integrated circuit

RGB – red-green-blue

VPAD – valid padding

MP – max pooling

AP – average pooling

SVM – support vector machines

PCA – principal component analysis

LDA – linear discriminant analysis

LN – landmark normalization

LFA – local feature analysis

EBGM – elastic bunch graph method

NMF – non-negative matrix factorization

RP – random projection

GPU – graphical processing unit

RNN – recurrent neural network

ReLU – rectified linear units

CL – convolution layer

Content

Introduction

Recognition of visual images is one of the most important components of control and information processing systems, automated systems and decision-making systems. Problems related to the classification and identification of objects, phenomena and signals characterized by a finite set of certain properties and features arise in such industries as robotics, information retrieval, monitoring and analysis of visual data, and artificial intelligence research. Algorithmic processing and classification of images are used in security systems, control and management of access, in video surveillance systems, virtual reality systems and information retrieval systems. At the moment, in production, systems for the recognition of handwritten text, license plates, fingerprints or human faces are widely used, which are used in software product interfaces, security and personal identification systems, as well as in other application purposes.

Intensive research in this area has a long history and is associated with the works of D. Hubel and T. Wiesel,T. Kohonen, M. Turk and A. Petland, D. Hinton, J. Lekun and others. Recently, significant progress in the recognition of visual images has been achieved with the advent of dimensionality reduction methods, convolutional neural networks and constellation models. However, despite the successes achieved, modern research confirms the fact that image recognition algorithms still do not have the full capabilities of biological visual systems, such as the ability to function on a wide, unbounded set of recognition classes, resistance to invariant transformations and variability of objects in within categories.

Thus, the actual problem recognized by the scientific community is the recognition of the depicted objects under the influence of affine transformations that can significantly change the shape of the image without affecting the belonging of the object to the recognition category. Attempts to solve this problem, which appears in the theory of pattern recognition as the inversion problem, have been undertaken in methods such as SIFT and ORB, as well as multilayer convolutional networks, but at the moment these methods offer partial solutions

that provide resistance to a limited subset of transformations. The urgency of this problem is especially high in industries where pattern recognition is used in a natural environment (video surveillance, data analysis from monitoring cameras, robotic visual systems), where the visual sensor can have an arbitrary limited viewing angle with respect to the desired object.

Currently, biometric systems for human identification are becoming more widespread. Their main advantages over traditional identification methods are as follows: they are based on unique biological characteristics, and, therefore, they are extremely difficult to counterfeit. Also, the convenience of their use is obvious - they do not require a person to possess any special cards, keys, etc.

There are several ways of biometric identification. At the moment, the leaders are fingerprint and retinal identification. Other types of identification (by face or voice) are less developed. They are not so reliable in nature (easier to falsify), and therefore their use is possible only in some areas.

The relevance of the problem of recognizing a person by the image of his face, as well as its preference over other means of identification of a person (for example, identification by fingerprints or by the retina) lies in the fact that there is no need for direct contact between the system and the person.

The main difficulties that need to be overcome when identifying a person by face is to ensure the independence of the system from factors such as illumination, angle, and age-related changes. Many methods involve a large and costly preprocessing step. However, without understanding some of the general semantics of the image, it is difficult to get it right.

That is why the direction of neural network methods looks promising. The principle of operation of such methods is based on the principle of the human brain. With the help of training, they allow you to find the relationship between individual features of the image and perform recognition with sufficient accuracy.

1. IMAGE PROCESSING OF DIAGNOSTICS SYSTEMS

1.1. Problem statement

In modern world there a lot of fields in which diagnostics systems takes important role in result sufficiency. One of them is the medical diagnostics, especially in procedures like UWI, CT, US, MRI scans etc. In the processing of such scans human factor takes a serious influence and there's no ready to go solution to make it automatically or partially automatically processed and analyzed. The goal of this work is to design such system and configure it to work with medical data by means of structural parametric synthesis algorithms and convolutional neural networks.

The aim of this work is to develop the convolutional neural network system, its own structure, configuration and parameters, choose and apply visual pattern recognition method capable of solving the inversion problem for various applications, recognizing patterns on the images, objects of the surrounding world, taking into account their invariant transformations.

To achieve this goal, it is necessary to solve the following tasks:

1. Development of a model of object representation using a hierarchy of features that are resistant to invariant transformations.

2. Development of an convolutional neural network architecture to work with following model, choose it's parameters and design the specific structure. To apply the feature extraction algorithm and an image recognition algorithm using the computer program means. Implement the algorithmic complex in the form of a computer program.

3. Evaluation of the performance of the developed method and criteria for

achieving the goal.

4. Evaluation of the effectiveness of the developed method in comparison with modern alternative recognition methods.

The research object of the thesis is the systems computer vision carrying out classification and identification of objects in the image.

The subject of the research is mathematical models and algorithms of image recognition.

To solve the set tasks, the methods of computer vision, optimization theory, mathematical statistics, the theory of artificial neural networks, probabilistic models, and the theory of experiment planning were used.

1.2. Standard structure of Convolutional Neural Network

The convolutional neural network systems (Conv Nets or CNNs) are the logical instrument receives an input parameters as image in the set of pixels view, finds some features on it and due to it sets the parameters (weighted coefficients) to wide data objects in the images and be able to highlight some things among all over objects. Conv Nets requires the less clean processing power relatively to other processing algorithms. Unlike to standard filter methods that working as hard-engineered unit, the convolutional neural network can achieve it through the training processes.

The structure of a Conv Nets is same as connectivity image of human brain biological neurons and was based on group of the “Visual Cortex”. Each one neuron responds to stimuli only in a specially-restricted area of the visual field known as the receptive area. A set of such areas overlap to cover the entire visual area. A Conv Nets are able to clearly capture the spatial and temporal dependencies in any input image data through the application based on relevant filters.

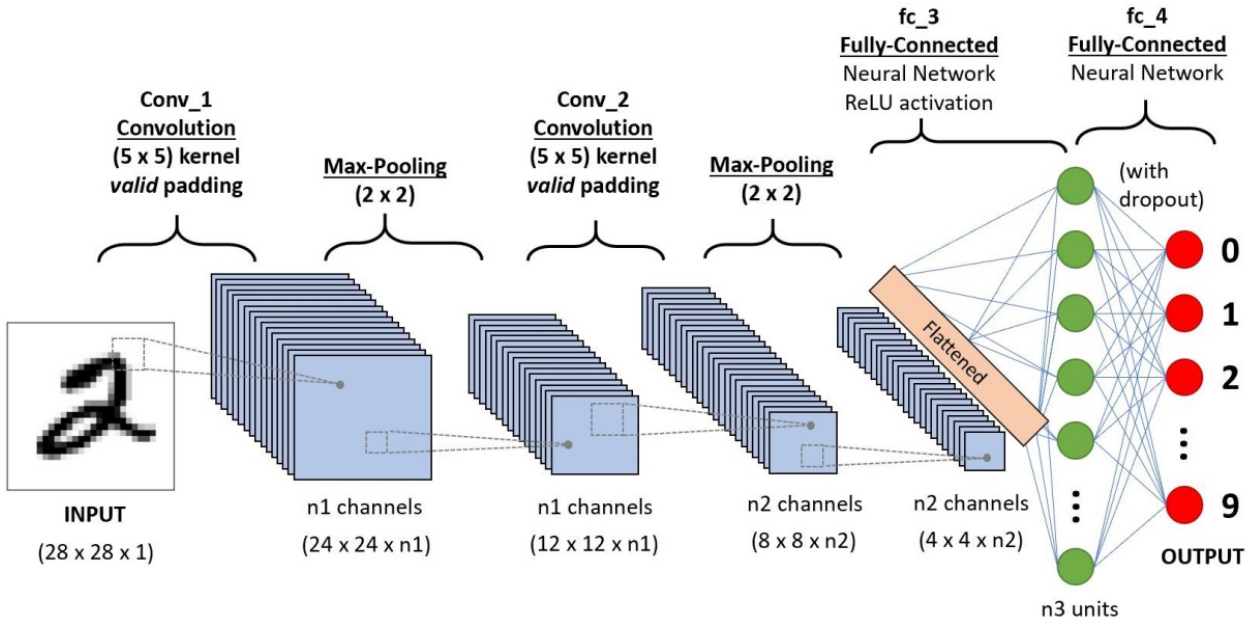


Fig. 1.1 Convolutional neural network structure

CNN structure performs a great setting to the image dataset with help of reduction in the number of parameters involved and reusability of weights. Therefore, the neural network can be trained to understand the sophistication of the image better. [4]

1.2.1. Convolutional Layer (Kernel)

The main and the first layer of Conv Net is the convolutional layer. This layer detects the logic into sets of pixels and extracts the features from them. There the logical operation which includes input set of pixels(image) and filter(Kernel).

At the figure 2.3.1, green area resembles our 5x5x1 input image, I. The element which acts in convolutional operation of a convolution layer is called the “kernel” or in other words “filter”, K, which displayed with yellow color. K looks like 3 x 3 matrix.

“Kernel”/”Filter”, K =

1	0	1
0	1	0
1	0	1

Kernel just slides from left to the right with the previously set-upped stride value till it reach the end of current string. Then in jumped out on the next string and process will be repeated relatively. During some time filter end all the strings and we'll obtain the output matrix.

Applying the huge range of filters allows us to get different results such as edges, sharpen, features, relief map detection. Also, filters had the same color depth as the input image. Matrix multiplication is performed between K_n and I_n stack ([K1, I1]; [K2, I2]; [K3, I3]) and all the results are summed with the bias to give us a squashed one-depth channel "convoluted feature output". [5]

Let us introduce the convolutional layer l. In a similar architecture neural network l is taken as an odd number, that is $l = 1, 3, \ldots, 2\alpha + 1$.

Then, for the feature map n, the following will take place:

- $\mathbb{w}_{m,n}^{l} = \{w_{m,n}^{l}(i,j)\}$ - convolution applied to feature map m layer (L - 1), on layer L with feature map n;
- b_n^l - threshold values attached to the feature map n on layer l;
- V_n^l a list of all layer levels (l - 1) that are connected to feature map n layer l.

So the feature map n of the convolutional layer l will be computed in the following way:

$$y_n^l = f_l\left(\sum_{m \epsilon V_n^l} y_m^{l-1} \otimes w_{m,n}^l + b_n^l\right) \qquad (1.1)$$

where by the $\otimes$ operator we mean a mathematical operation of a two-dimensional convolutions.

Suppose that the size of the input feature maps y_m^{l-1} is equal $H^{l-1}\ X\ W^{l-1}$, and the size of the convolution applied to them is $\mathbb{w}_{m,n}^{l}$ equals $r^l\ X\ c^l$ then size of the output feature map y_n^l is calculated as (Fig. 1.4.1).

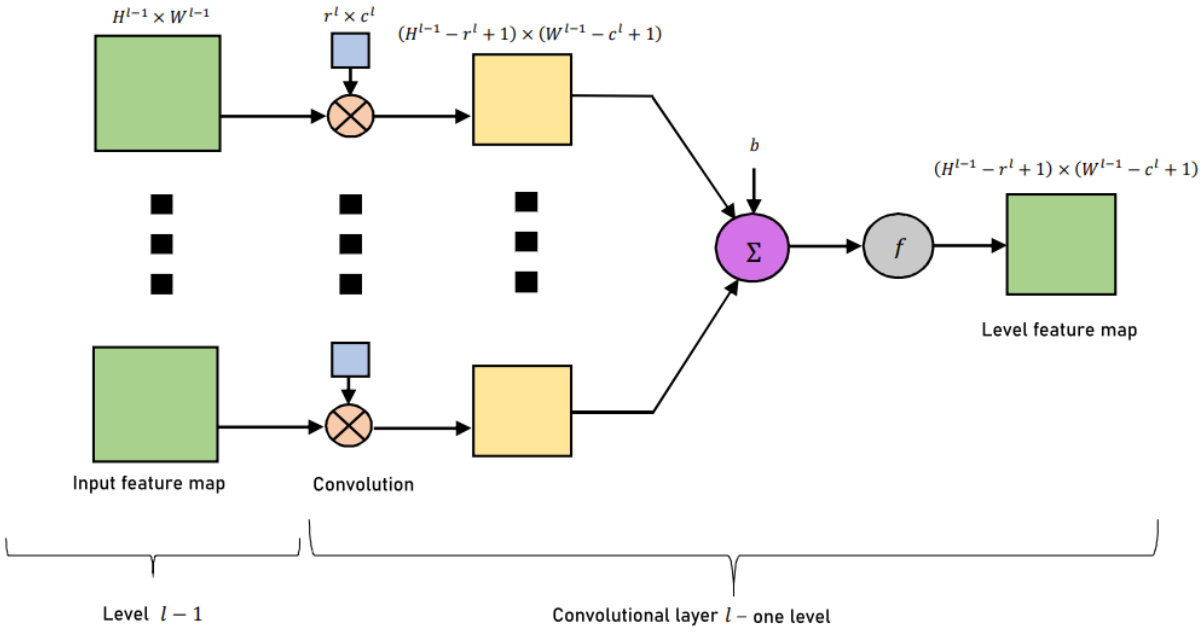


Fig. 1.2 Convolutional layer schema

The main aim of this is to get the features as edges and polygons from the input image. Conv Nets can hold any number of convolutional layers. Therefore, the first convolutional layer is reasonable in extracting the “low level” features such as edges, color, gradient orientation, etc.

By expanding it with another layers the architecture adapts to the “high level” features as well, giving us a network which has the wholesome understanding of images in the dataset, similar to how we would.

On the output we’ll get two types of data where the convoluted feature loses its dimensionality relatively to the origin and the other with the increasing or holding actual dimensionality. To achieve it is necessary to use “valid padding” in first case and the “same padding” with another one. [3]

In situation when we expand the 5x5x1 picture into a 6x6x1 picture and after we’ll filter it through 3x3x1 Kernel one, we’ll get out image matrix (convolved matrix) with 5x5x1 sizes. In other way in the case when we apply such operation without using the padding method it will cause that the output dimensionality value will be the same as the Kernel filters dimensionality (3x3x1).

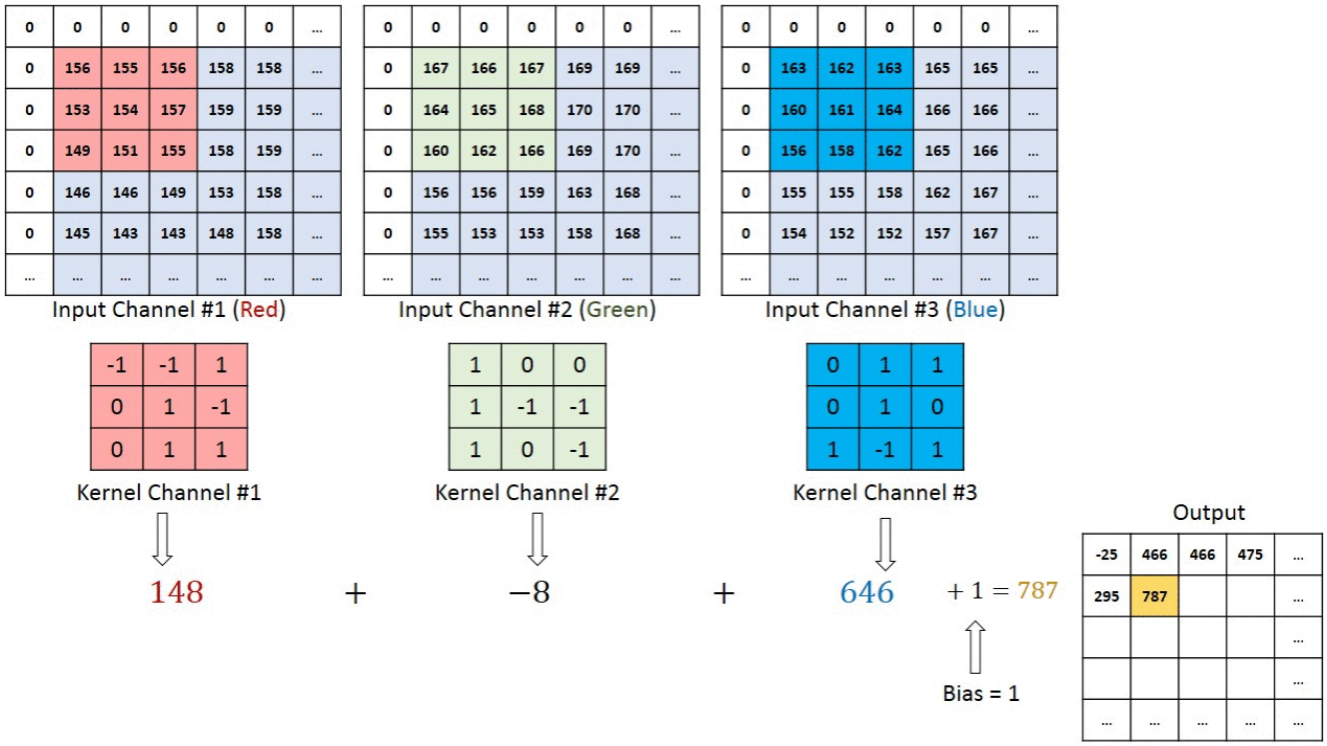


Fig. 1.3 Representation of Kernel filters

1.2.2. Pooling Layer

As the convolution layer the pooling one based on reducing the spatial size of the convoluted feature. By reducing the dimensionality it will slows the load on the computational elements. Also, it is allowable to highlight the dominant features in any angles and positions to burst the learning process.

Here exist only two types of pooling layers: “max pooling” and “average pooling”. Max pooling finds the maximum value at the resulting matrix given by Kernel filters. In its turn, average pooling finds the average value at the resulting matrix given by Kernel filters.

Max pooling also can take the role of noise suppressing. Due to dimensionality reduction it’s reduces the activation noises appears along it. In other way, the average pooling allows us to reduce dimensionality as a noise

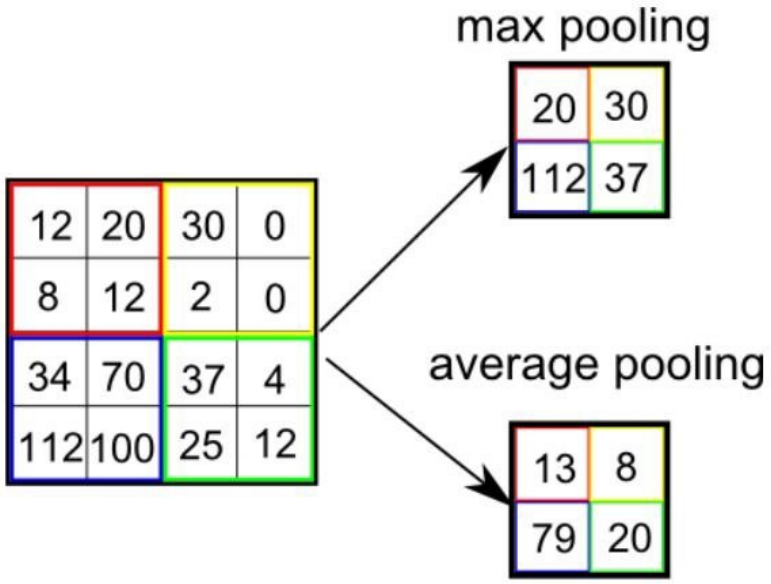


Fig 1.4 Representation of polling layers

suppressing tool. In the result we can obtain that max pooling process much more better than average pooling

Commonly pooling layers always working in para with convolutional layer and it forms the i-th convolution layer of CNN structure. Relatively to input image, its dimensionality and amount of features it will may cause on number of such convolution layers and on the computational power as well. Alright, now we're understand how the convolutional layers works from inside, its construction and working approach. Finally, the output goes straight into the flatten layer to transform it into the vector for feeding it into classifier (fully connected layer).[7] Let's introduce the subsampling layer l. In convolutional neural network l is accepted to be taken as an even number, that is $l = 2, 4, \ldots, 2\alpha$. For feature maps n we introduce the following notation: $\mathbb{w}_{m,n}^{k}$ - filter, applied to n on layer l, and b_n^l - additional threshold value. Next, we will act as follows: divide the feature map n (l - 1) -th layer into disjoint blocks of 2 × 2 pixels. Then sum up the values of four pixels in each block and as a result we obtain the matrix $\mathbb{z}_n^{l-1} = \{z_n^{l-1}(i,j)\}$, the elements of which will be the corresponding values of the sums. Thus, the formula for calculating values of matrix elements will be as follows:

$$z_n^{l-1} = y_n^{l-1}(2i-1,2j-1) + y_n^{l-1}(2i-1,2j) + y_n^{l-1}(2i,2j-1) + y_n^{l-1}(2i,2j)$$

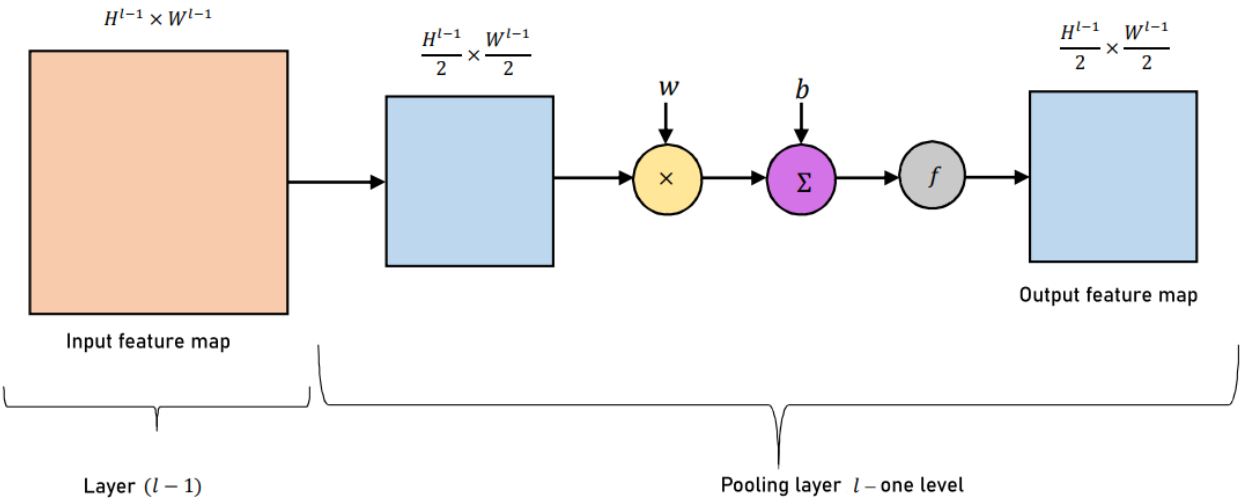


Fig. 1.5 Pooling layer schema

The feature map n subsampling layer l is calculated as:

$$y_n^l = f_l\left(\mathbb{z}_n^{l-1} \times \mathbb{w}_{m,n}^l + b_n^l\right) \qquad (1.2)$$

Thanks to the above reasoning, it becomes possible calculate size $H^l\ X\ W^l$ feature maps y_n^l downsampling layer l (Fig. 1.4.2).

$$H^l = \frac{H^{l-1}}{2}, W^l = \frac{W^{l-1}}{2} \qquad (1.3)$$

1.2.3. Fully connected layer (FC)

Fully connected layer is the common instrument to find the non-linear relations in the feature structures that we'll get from the convolutional layers. So it's always applied as a final stage of convolutional neural networks. The fully connected layer is finding the possible non-linear functions.

To classify the output there widely used the "softmax classification function". Till a lot of epochs will done the neural network will set the right weighted coefficients and be able to classify our outputs. Nowadays, there exist a huge amount of different convolutional neural network structures with different number of layers and also different performance. Most popular structures are: Le-Net, Alex-Net, VGG-Net, GoogLe-Net, Res-Net, ZF-Net.

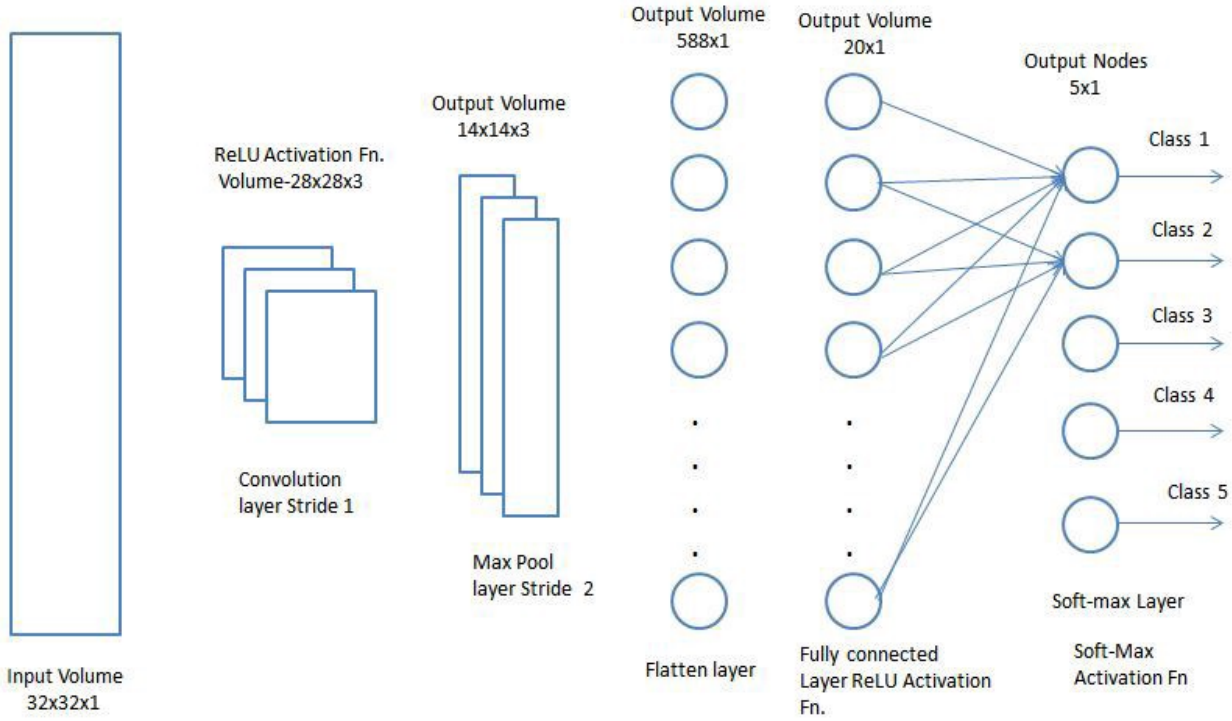


Fig. 1.6 Structural scheme of fully connected layer

1.2.4. Forming the input data

As the input commonly we will get the RNG image which means R – red color, G – greed and B – blue. Also, image can be in different color types: grayscale, HSV, RGB, CMYK, etc.

For the while, take into the mind which the computational power needed for process the image of 8K (7680×4320) sizes. The main purpose of Conv Nets is to

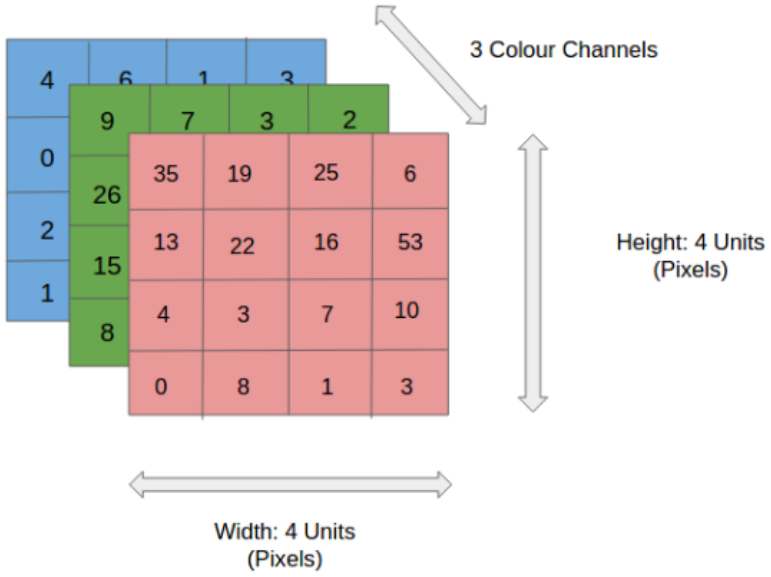


Fig. 1.7 Input data representation (RGB)

compress it into much more lighter state without losing important parts (features). It's necessary to take care when we're designing structure with low features learning possibilities and huge data sets.

1.3 Review on image processing approaches

1.3.1 Geometric-based method (Template-based method)

Face recognition algorithms divided into geometry-based ones or template-based methods. To realize practically the template based method it's necessary to use some tools for statistical calculations as SVMach (Supporting Vector Machines), PCA (Principal-Component Analysis), LDA (Linear-Discriminant Analysis). Also it can be the kernel-based approach. The geometric-based method relay on the geometric relationships between some key-features, its areas and points. This method also can be named as feature based method. [7,8]

Already over the 30 years face recognition problem hold the one of most important tasks of computers vision. It's caused by huge amount of real-life requirements for applications of face recognition in security purposes, telecommunications, sociological experiments and statistics, digital libraries, medicines, etc.

In the theory the face feature recognition and also the human face recognition is the same as simple features recognition but more complex and complicated. Faces are the completely hard structured object which hard to analyze and able only closely to frontal-face positioning. Just an unusual little change can vary the faces different. That is why in a common face features recognition area there's forming a dense setup of frontal-based faces, and the common face-pattern method will fail to work with them. [14]

Face recognition systems can be divided into two different types. The first one is perform to check that the person that appears is the one of the saved in small database of persons (at least lower that 500). Usually, this method performs with smartphones, building or program access in the security purposes. The key points

of such system are that the program working in the real-time, low latency in appearance detection and also low sensitivity for appearance changes.

Second one finding the person in the huge database with the lot of the faces there. Often this system working with databases which contains thousands of faces and its relative information and due to intense computational requirements working in offline way.

The main approach of this method is to find and analysis the key-feature points (anthropometrical local points). Then after finding their displacement they are needed to be manually approved by someone to increase the accuracy. This points displacement information stores in specially formed data bases. An example of this key points recognition shown at figure 2.6.2.[3]

Process of face feature recognition commonly consists of three main steps: finding the special face landmarks, construct the shapes which describes face features using obtained landmarks, and also classification this shapes into different subclasses.

The process of landmark displacement consist of two main parts. The first one is landmark displacement due to knowledge of its locations and second based in extracting the “gabor-jet” at the given images and estimate the approximations of this points by comparing it with the different models. For getting this system to be full-automatic firstly it’s necessary to approximate the location of few initial landmarks. To do that there is important to extract the face area. Usually, to achieve it we’re using the haarcascade methods for example haar frontalface cascade method. So, the points displaced at the middle of the eyes were founded using the haar method proposed by Viola and Jones. Than having the eyes location coordinates it’s much more easier to find another landmarks. One by one, these landmarks founded using the information about previous ones. The landmark location is then refined by comparing a Gabor jet extracted from the estimated point to a corresponding model jet from the bunch graph. Until the all landmarks will be founded the process will be executed. Then when all of them were founded

it's necessary to overwatch their changes during the time. For each next image of the video the displacement of landmarks should be founded.

Landmark Normalization

Process of landmarks normalization locks the landmarks to their uniform coordinate position at the initial frame of the video-shot, and as it's shown in expression evolves, landmarks moved accordingly. Just denote that S_i^k is the observing data for i-th landmark in the k-th sequence:

$$S_i^k = \{(x_0, y_0)_i^k, (x_1, y_1)_i^k, (x_2, y_2)_i^k, \dots, (x_{N-1}, y_{N-1})_i^k, (x_N, y_N)_i^k\} \quad (1.4)$$

where $(x_l, y_l)_i^k$ is the i-th landmarks co-ord positioning in the l-th frame of the k-th sequence. There the N is the amount of frames in the sequence.

The each landmark position relative to other ones is obtained from all the images, which are the first frames in all video-shots. All the landmarks should be normalized for every detection result and the differences between initial frame landmarks and the average ones also should be determined. This gives the displacement of the landmark, with respect to the average landmark position. Let's denote $(\delta_{x0},\delta_{y0})^k_i$ as the displacements of the i-th landmarks in the 1-st image of the k-th sequence respectively to value of average landmarks positioning:

$$(\delta_{x0}, \delta_{y0})_i^k = (\mu_{x0} - x_0, \mu_{y0} - y_0)_i^k \quad (1.5)$$

Displacements of each landmark are now added to the landmarks positioning in every image of the facial recognition sequence. The landmarks observing transformed result is now denoted by $S_i^{`k}$, and defined by:

$$S_i^{`k} = \{\left(x_0 + \delta_{x0}, y_0 + \delta_{y0}\right)_i^k, \left(x_1 + \delta_{x0}, y_1 + \delta_{y0}\right)_i^k,$$
$$\dots, \left(x_N + \delta_{x0}, y_N + \delta_{y0}\right)_i^k\} \quad (1.6)$$

The observing results are normalized with relevance that for each landmarks for all the expression sequences now starts from the one initial coordinate position.

Feature Extraction

The peculiarity of this method is that for facial recognition and also for facial expressions recognition we're using only geometrical information without

performing with any graphical texture information. The feature extraction process based on two options, one by getting the observing result for each landmarks positioning, and the second by considering the observing data for pairs of these landmarks. Let's denote (x′, y′) as the transformed landmark coordinates positioning.

$$S_i^{`k} = \{(x`_0, y`_0)_i^k, (x`_1, y`_1)_i^k, \dots, (x`_N, y`_N)_i^k\} \qquad (1.7)$$

The amount of images for each video-shots for facial recognition can be different. So, for each group class there should be created the prototypic facial expression. Therefore, for each of these expressions there should be the same amount of images in database.[11]

Features types ones are the features vector from the each individual landmarks observing sequences. Each landmarks coordinates in a sequences are subtracted from the initial landmarks coordinates, i.e., landmarks positioning in the neutral frame, for creating the types one features vector. Let's denote $(\delta_{x\,l}^{'}, \delta_{y\,l}^{'})^k{}_i$ as the differences at i-th landmarks in the l-th image, from the i-th landmarks in the initial frame of the k-th video-shot:

$$\left(\delta x_l^{`}, \delta y_l^{`}\right)_i^k = \left(x_l^{`} - x_0^{`}, y_l^{`} - y_0^{`}\right)_i^k \qquad (1.8)$$

1.3.2 Appearance-based method

The appearance-based method compares the image relatively to few another ones. There, the image presented as high-dimensional vector. This method widely used to extract and form the feature field from the image division. That means that the an image from the sample will be compared with special training set. In other way, the model-based method is representing the facial model. For the new implemented sample into the model, its parameters are used for images recognizing.

The appearance based approach used to classify as "linear" or "non-linear". Ex- PCA, LDA, IDA are used in direct approach whereas Kernel PCA used in

nonlinear approach. Anyway, for the model based approach can be presented as 2-D or 3-D “exelastic bunch graph matching” used.

Appearance based facial recognition methods required for huge variety of purposes, for example for applications of face recognition in security purposes, telecommunications, sociological experiments and statistics, digital libraries, medicines, etc. Obviously, human brain able to recognize faces easily, so constructing the fully-automated facial recognition systems is a huge challenge in data processing and computer vision spheres. To achieve the a cheap and high-level categorical sorting, instead just follow a guide-line based on the scientific psychological studies of human face holistics and local-features. Specifically, the proposed papers have the following categorization: “Holistics” approach and “Hybrid” approach. [17]

1.3.3 Template matching method

Template matching method is the simple approach for image processing tasks and also the facial recognition, features and edges extraction, etc. This method consists of two main approaches: “feature based” and “template based” matching. The feature based method applied to find the features and template image, such as edge and corner, as the main metric of conformity measurement, to find the best matching location of the template in the original image. A template-based or global approach uses the entire template. To recognize a human face, it is necessary to extract some features. These features include the eyes, nose, mouth, and chin, as well as the shape of the face. To determine the location of these features, the researchers proposed various methods based on facial symmetry, facial geometry and brightness, as well as pattern matching. As a rule, facial recognition based on vision can be explained as follows. Initially, the image of the object is improved and segmented. Then the contour features are extracted by the contour extraction method and compared with the image features extracted from

the database. If there is a match, then the face in the image of the subject is recognized.

1.3.4 Principal Component Analysis (PCA)

The Principal Component Analysis (PCA) is a standout amongst the best procedures that have been utilized in picture acknowledgment and pressure. PCA is a measurable strategy under the wide title of factor investigation. The motivation behind PCA is to diminish the enormous dimensionality of the information space (watched factors) to the littler characteristic dimensionality of highlight space (free factors), which are expected to depict the information financially. This is the situation when there is a solid relationship between's watched factors.

The occupations which PCA can do are expectation, repetition evacuation, include extraction, information pressure, and so forth. Since PCA is an old style method which can accomplish something in the straight area, applications having direct models are appropriate, for example, signal preparing, picture handling, framework and control hypothesis, interchanges, and so on.

Face acknowledgment has numerous pertinent territories. In addition, it tends to be sorted into face identification, face classification, or sex assurance. The most helpful applications contain group surveil-spear, video substance ordering, individual identification (ex. driver's permit), mug shots coordinating, entrance security, and so on. The principle thought of utilizing PCA for face acknowledgment is to express the huge 1-D vector of pixels built from 2-D facial picture into the minimal essential segments of the component space. This can be called eigenspace projection. Eigenspace is determined by distinguishing the eigenvectors of the covariance lattice got from a lot of facial images(vectors). The subtleties are portrayed in the accompanying area.

Once the eigenfaces have been processed, a few sorts of choice can be made relying upon the application. PCA processes the premise of a space which is spoken to by its preparation vectors. These premise vectors, really eigenvectors,

figured by PCA are toward the biggest difference of the preparation vectors. As it has been said before, we call them eigenfaces. Each eigenface can be seen a component. At the point when a specific face is anticipated onto the face space, its vector into the face space portray the significance of every one of those highlights in the face. The face is communicated in the face space by its eigenface coe□cients (or loads). We can deal with a huge information vector, facial picture, just by taking its little weight vector in the face space. This implies we can reproduce the first face with some mistake, since the dimensionality of the picture space is a lot bigger than that of face space.

1.3.5 Linear Discriminant Analysis method

Linear Discriminant Analysis unequivocally endeavors to demonstrate the distinction between the classes of information. LDA is an amazing face acknowledgment system that beats the impediment of Rule segment investigation method by applying the straight discriminant basis. This foundation attempts to augment the proportion of the determinant of the between-class dissipate grid of the anticipated examples to the determinant of the with-in class disperse network of the anticipated examples. Direct discriminant gathering pictures of a similar class and isolates pictures of various classes of the pictures.

Discriminant analysis can be used only for order not for relapse. The objective variable may have at least two classes. Pictures are anticipated from two dimensional spaces to c dimensional space, where c is the quantity of classes of the pictures. To distinguish an information test picture, the anticipated test picture is contrasted with each anticipated preparing picture, and the test picture is recognized as the nearest preparing picture. The LDA technique attempts to discover the subspace that segregates diverse face classes. The inside class disperse framework is likewise called intra-individual methods variety in appearance of a similar individual because of various lighting and face demeanor. The between-class dissipate network likewise called the additional individual

speaks to variety in appearance because of distinction in personality. Straight discriminant techniques gathering pictures of similar classes and isolates pictures of the various classes. To recognize an info test picture, the anticipated test picture is contrasted with each anticipated preparing picture, and the test picture is distinguished as the nearest preparing picture.[20]

In Direct discriminant examination we give the accompanying strides to discriminant the info pictures:

Stage 1:

We need a preparation set made out of a moderately huge gathering of subjects with assorted facial qualities. The proper choice of the preparation set straightforwardly decides the legitimacy of the last outcomes. The database ought to contain a few instances of face pictures for each subject in the preparation set and at any rate one model in the test set. These models ought to speak to various frontal perspectives on subjects with minor varieties in view point. They ought to likewise incorporate distinctive outward appearances, changed lighting and foundation conditions, and models with and without glasses. It is accepted that all pictures are now standardized to m × n exhibits and that they contain just the face locales and very little of the subjects' bodies.

Stage 2:

For each picture and sub picture, beginning with the two dimensional m × n cluster of power esteems I(x, y), we develop the vector extension Φ R m× n. This vector relates to the underlying portrayal of the face. In this way the arrangement of all countenances in the element space is treated as a high-dimensional vector space.

Stage 3:

By characterizing all occasions of a similar individual's face as being in one class and the essences of various subjects as being in various classes for all subjects in the preparation set, we set up a system for playing out a bunch detachment investigation in the element space. Additionally, having named all

occasions in the preparation set and having characterized every one of the classes, we register the inside class and between-class dissipate frameworks.

1.3.6 Locality learning projections method

Locality Learning Projections (LLP) includes straight projective maps emerging subsequent to taking care of a variational issue that is ideally safeguarded by the area structure of the database. Direct estimation of non-straight Laplacian eigenmape is presented by LPP. It is a strategy utilized in complex learning. Laplace Betrami administrator is utilized in finding the straight approximations of the eigen capacities. Nearby structure of the informational index is protected in LPP.[21] Thinking about LPP, we will talk about Laplacianfaces strategy which is utilized in face acknowledgment in area saving subspace. Complex structure of informational collection is displayed by making a contiguousness chart. This contiguousness chart is utilized to express the nearby closeness of the informational index. The picture set is first anticipated to a PCA subspace in order to make a nonsingular lattice. Clamor and undesirable information focuses is additionally utilized with the assistance of PCA preprocessing.

1.3.7 Independent Component Analysis

Independent Component Analysis (ICA) has as of late pulled in a lot of consideration in sign handling and highlight extraction fields, and has been viewed as an effective instrument for displaying and understanding the shrouded elements that underlie sets of arbitrary factors, or flag. Autonomous Part Examination (ICA) is a speculation of central segment investigation (PCA), which decorrelates the higher request snapshots of the info. In an assignment, for example, face acknowledgment, a significant part of the significant data is contained in the high request measurements of the pictures. [13]

Independent Component Analysis (ICA) is a speculation of primary segment examination (PCA), which decorrelates the higher request snapshots of the info. In an undertaking, for example, face acknowledgment, a great part of the significant data is contained in the high request insights of the pictures. An illustrative premise wherein the high request measurements are decorrelated might be more dominant for face acknowledgment than one in which just the second request insights are decorrelated, as in PCA portrayals. Utilizing ICA, one endeavors to show the fundamental information so that in the direct development of the information vectors the coefficients are as free as could be allowed. ICA bases of the extension must be commonly free while the PCA bases are only uncorrelated. ICA has been generally utilized for visually impaired source partition and visually impaired convolution. Visually impaired source detachment endeavors to isolate a couple of autonomous yet obscure source signals from their direct blends without knowing the blend coefficients.

2. MODERN CONVOLUTIONAL NEURAL NETWORKS

2.1 Review on the existing networks for image classification approach

The research in CNN is still going on and has a significant potential for improvement. It is generally observed that the significant improvements in CNN performance occurred from 2015-2019. The representational capacity of a CNN usually depends on its depth, and in a sense, an enriched feature set ranging from simple to complex abstractions can help in learning complex problems. However, the main challenge faced by deep architectures is that of the diminishing gradient. Initially, researchers tried to subside this problem by connecting intermediate layers to auxiliary learners. In 2015, the emerging area of research was mainly the development of new connections to improve the convergence rate of deep CNN architectures. In this regard, different ideas such as information gating mechanism across multiple layers, skip connections, and cross-layer channel connectivity was introduced. Different experimental studies showed that state-of-the-art deep architectures such as VGG, ResNet, ResNext, etc. also showed good results for challenging recognition and localization problems like semantic and instance-based object segmentation, scene parsing, scene location, etc. Most of the famous object detection and segmentation architectures such as Single Shot Multibox Detector (SSD), Region-based CNN (R-CNN), Faster R-CNN, Mask R-CNN and Fully Convolutional Neural Network (FCN) are built on the lines of ResNet, VGG, Inception, etc. Similarly, many interesting detection algorithms such as Feature Pyramid Networks, Cascade R-CNN, Libra R-CNN, etc., modified the architectures as mentioned earlier to improve the performance. Applications of deep CNN were also extended to image captioning by combining these networks

with recurrent neural network (RNN) and thus showed state-of-the-art results on MS COCO-2015 image captioning challenge.

Similarly, in 2016, it was observed that the stacking of multiple transformations not only depth-wise but also in parallel fashion showed good learning for complex problems. Different researchers used a hybrid of the already proposed architectures to improve deep CNN performance. In 2017, the focus of researchers was mainly on designing of generic blocks that can be inserted at any learning stage in CNN architecture to improve the network representation (Hu et al. 2018a). Designing of new blocks is one of the growing areas of research in CNN, where generic blocks are used to assign attention to spatial and feature-map (channel) information. In 2018, a new idea of channel boosting was introduced by Khan to boost the performance of a CNN by learning distinct features as well as exploiting the already learned features through the concept of TL. [8]

However, two main concerns observed with deep and wide architectures are the high computational cost and memory requirement. As a result, it is very challenging to deploy state-of-the-art wide and deep CNN models in resource-constrained environments. Conventional convolution operation requires a huge number of multiplications, which increases the inference time and restricts the applicability of CNN to low memory and time constraint applications. Many real-world applications, such as autonomous vehicles, robotics, healthcare, and mobile applications, perform the tasks that need to be carried on computationally limited platforms in a timely manner. Therefore, different modifications in CNN are performed to make them appropriate for resource-constrained environments. Prominent modifications are knowledge distillation, training of small networks, or squeezing of pre-trained networks. GoogleNet exploited the idea of small networks, which replaces the conventional convolution with point-wise group convolution operation to make it computationally efficient. Similarly, ShuffleNet used point-wise group convolution but with a new idea of channel shuffle that significantly reduces the number of operations without affecting the accuracy. In

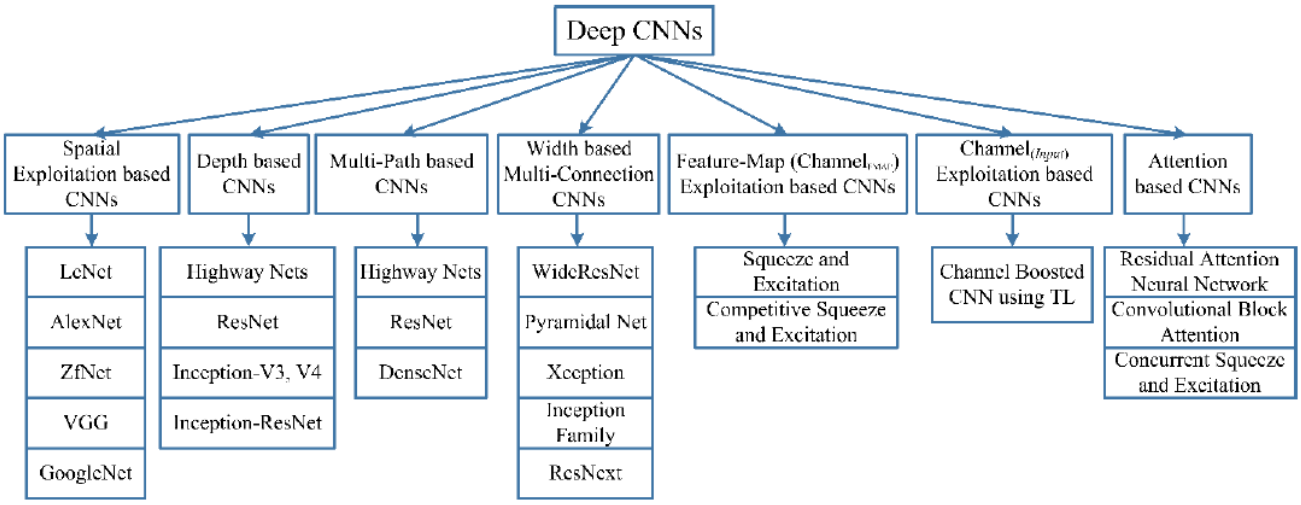


Fig. 2.1 Different categories of modern CNN architectures

the same way, ANTNet proposed a novel architectural block known as ANTBlock, which at low computational cost, achieved good performance on benchmark datasets.

From 2012 up till now, many improvements have been reported in CNN architectures. As regards the architectural advancement of CNNs, recently, the focus of research has been on designing of new blocks that can boost network representation by exploiting feature-maps or manipulating input representation by adding artificial channels. Moreover, along with this, the trend is towards the design of lightweight architectures without compromising the performance to make CNN applicable for resource constraint hardware. [13]

Different improvements in CNN architecture have been made from 1989 to date. These improvements can be categorized as parameter optimization, regularization, structural reformulation, etc. However, it is observed that the main thrust in CNN performance improvement came from the restructuring of processing units and the designing of new blocks. Most of the innovations in CNN architectures have been made in relation to depth and spatial exploitation. Depending upon the type of architectural modifications, CNNs can be broadly categorized into seven different classes, namely; spatial exploitation, depth, multi-path, width, feature-map exploitation, channel boosting, and attention-based CNNs. The taxonomy of CNN architectures is pictorially represented in figure

Architecture Name	Year	Main contribution	Parameters	Error Rate
LeNet	1998	- First popular CNN architecture	0.060 M	[dist]MNIST: 0.8 MNIST: 0.95
AlexNet	2012	- Deeper and wider than the LeNet - Uses Relu, dropout and overlap Pooling - GPUs NVIDIA GTX 580	60 M	ImageNet: 16.4
ZfNet	2014	-Visualization of intermediate layers	60 M	ImageNet: 11.7
VGG	2014	- Homogenous topology - Uses small size kernels	138 M	ImageNet: 7.3
GoogLeNet	2015	- Introduced block concept - Split transform and merge idea	4 M	ImageNet: 6.7
Inception-V3	2015	- Handles the problem of a representational bottleneck - Replace large size filters with small filters	23.6 M	ImageNet: 3.5 Multi-Crop: 3.58 Single-Crop: 5.6
Highway Networks	2015	- Introduced an idea of Multi-path	2.3 M	CIFAR-10: 7.76
Inception-V4	2016	- Split transform and merge idea Uses asymmetric filters	35 M	ImageNet: 4.01
Inception-ResNet	2016	- Uses split transform merge idea and residual links	55.8M	ImageNet: 3.52
ResNet	2016	- Residual learning - Identity mapping based skip connections	25.6 M 1.7 M	ImageNet: 3.6 CIFAR-10: 6.43
DelugeNet	2016	- Allows cross layer information flow in deep networks	20.2 M	CIFAR-10: 3.76 CIFAR-100: 19.02
FractalNet	2016	- Different path lengths are interacting with each other without any residual connection	38.6 M	CIFAR-10: 7.27 CIFAR-10+: 4.60 CIFAR-10++: 4.59 CIFAR-100: 28.20 CIFAR-100+: 22.49 CIFAR100++: 21.49

Fig. 2.2 Perfomace comparison of the recent architectures of different CNNs

about Architectural details of the state-of-the-art CNN models, their parameters, and performance on benchmark datasets are summarized. On the other hand, different online resources on deep CNN architectures, vision-related dataset, and their implementation platforms are mentioned.

CNNs have a large number of parameters and hyper-parameters, such as weights, biases, number of layers, and processing units (neurons), filter size, stride, activation function, learning rate, etc. As convolutional operation considers the neighborhood (locality) of input pixels, therefore different levels of correlation can be explored by using different filter sizes. Different sizes of filters encapsulate different levels of granularity; usually, small size filters extract fine-grained and large size extract coarse-grained information. Consequently, in early 2000, researchers exploited spatial filters to improve performance and explored the relation of a spatial filter with the learning of the network.

WideResNet	2016	- Width is increased and depth is decreased	36.5 M	CIFAR-10: 3.89 CIFAR-100: 18.85
Xception	2017	- Depth wise convolution followed by point wise convolution	22.8 M	ImageNet: 0.055
Residual Attention Neural Network	2017	- Introduced an attention mechanism	8.6 M	CIFAR-10: 3.90 CIFAR-100: 20.4 ImageNet: 4.8
ResNeXt	2017	- Cardinality - Homogeneous topology - Grouped convolution	68.1 M	CIFAR-10: 3.58 CIFAR-100: 17.31 ImageNet: 4.4
Squeeze & Excitation Networks	2017	- Models interdependencies between feature-maps	27.5 M	ImageNet: 2.3
DenseNet	2017	- Cross-layer information flow	25.6 M 25.6 M 15.3 M 15.3 M	CIFAR-10+: 3.46 CIFAR100+:17.18 CIFAR-10: 5.19 CIFAR-100: 19.64
PolyNet	2017	- Experimented structural diversity - Introduced Poly Inception module - Generalizes residual unit using polynomial compositions	92 M	ImageNet: Single:4.25 Multi:3.45
PyramidalNet	2017	- Increases width gradually per unit	116.4 M 27.0 M 27.0 M	ImageNet: 4.7 CIFAR-10: 3.48 CIFAR-100: 17.01
Convolutional Block Attention Module (ResNeXt101 (32x4d) + CBAM)	2018	- Exploits both spatial and feature-map information	48.96 M	ImageNet: 5.59
Concurrent Spatial & Channel Excitation Mechanism	2018	- Spatial attention - Feature-map attention - Concurrent placement of spatial and channel attention	-	MALC: 0.12 Visceral: 0.09
Channel Boosted CNN	2018	- Boosting of original channels with additional information rich generated artificial channels	-	-
Competitive Squeeze & Excitation Network CMPE-SE-WRN-28	2018	- Residual and identity mappings both are used for rescaling the feature-map	36.92 M 36.90 M	CIFAR-10: 3.58 CIFAR-100: 18.47

Fig. 2.3 Perfomace comparison of the recent architectures of different CNNs

Different studies conducted in this era suggested that by the adjustment of filters, CNN can perform well both on coarse and fine-grained details.

2.2 List of specialized constructive convolutional neural network blocks

2.2.1 Squeeze and excitation block

The way in which works the squeeze and excitation block is presented at figure 2.4. Firstly, it's applies the feature transformation or simple convolution operation on input data X to receive the U features. After it's necessary to apply a

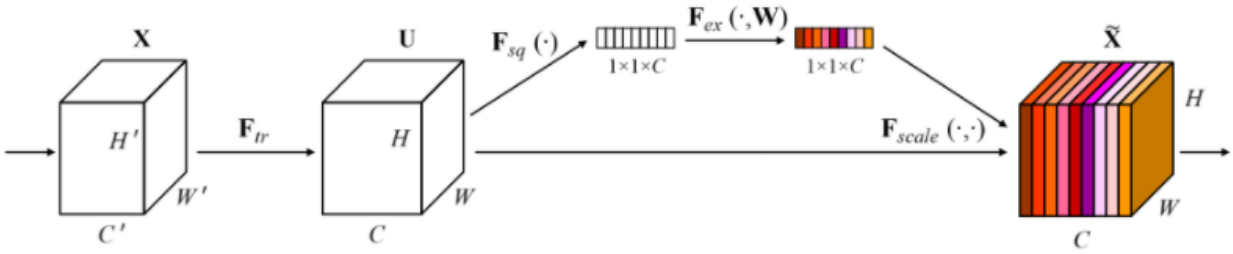


Fig. 2.4 The structure of squeeze and excitation block

squeeze operation to obtain single value of output U from each channel. Then it goes through excitation operation applied on outputs of squeeze ones to obtain per-channel weights. At the last, it rescales the feature map U with these activations to get the resulting SE block outputs.

The role this operation performs at different depths differs throughout the network. In earlier layers, it excites informative features in a class-agnostic manner, strengthening the shared low-level representations. In later layers, the SE blocks become increasingly specialised, and respond to different inputs in a highly class-specific manner. As a consequence, the benefits of the feature recalibration performed by SE blocks can be accumulated through the network.

The main idea of squeeze and excitation blocks is that it's can be implemented and used with most of already existed convolutional neural network architectures (for example, ResNet, Inception, AlexNet, DenseNet etc.). It will increase the network performances by slightly increasing the architecture complexity. [14]

On the figure 2.1.3 it is shown the implementation of this algorithm on the examples of inception and residuals blocks from the relative networks. For the inception architecture it's necessary to add SE blocks after each inception module and in the residual ones they needed to be placed in the main stream before adding the residual. For the linear architecture implementation they can be placed after convolutional 2D layers.

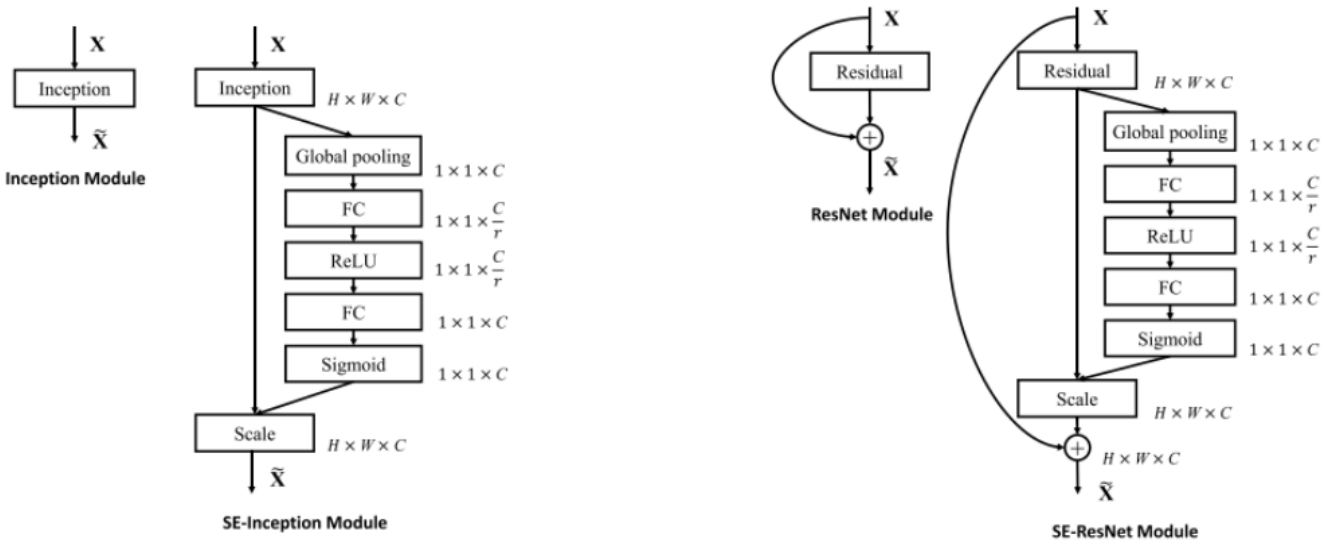


Fig. 2.5 Conceptual scheme of squeeze and excitation implementation on the inception and residual modules

The presented SE block can be applied to any given transformation as computational element $\mathbf{F}_{tr}: \mathbf{X} \to \mathbf{U}, \mathbf{X} \in \mathbb{R}^{H' \times W' \times C'}, \mathbf{U} \in \mathbb{R}^{H \times W \times C}$. Let the learned set of kernel filters be denoted as $\mathbf{V} = [\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_C]$ where the v elements is the parameters of the relative filters. So we can mark the outputs of $\mathbf{F}_{tr}$ as $\mathbf{U} = [\mathbf{u}_1, \mathbf{u}_2, \dots, \mathbf{u}_C]$, where

$$\mathbf{u}_c = \mathbf{v}_c * \mathbf{X} = \sum_{s=1}^{C'} \mathbf{v}_c^s * \mathbf{x}^s \qquad (2.1)$$

Here * denotes convolution, $\mathbf{v}_c = [\mathbf{v}_c^1, \mathbf{v}_c^2, \dots, \mathbf{v}_c^{C'}]$, $\mathrm{X} = [\mathbf{x}^1, \mathbf{x}^2, \dots, \mathbf{x}^{C'}]$ and $\mathbf{u}_c \in \mathbb{R}^{H \times W}$. $\mathbf{v}_c^s$ is a 2D spatial kernel representing a single channel of $\mathrm{v_c}$ that acts on the corresponding channel of X. Since the output is produced by a summation through all channels, the channel dependencies are implicitly embedded in $\mathrm{v_c}$, but these dependencies are entangled with the spatial correlation captured by the filters.

Squeeze operation: Global Information Embedding. The main goal of this operation is to extract global information from each of the channels of the image. So achieve that there proposed using the Global Average Pooling to reduce the C x H x W image to C x 1 x 1 to get global statistic for each channel. The formula 2 represents the squeeze (Global Average Pooling) operation.

$$z_c = \mathbf{F}_{sq}(\mathbf{u}_c) = \frac{1}{H \times W} \sum_{i=1}^{H} \sum_{j=1}^{W} u_c(i, j) \qquad (2.2)$$

Excitation operation: Adaptive Recalibration. With the help of this operation we can process the received vector output with the C length to produce the channel weights sets. The formula 3 represents the mathematical description of excitation operation.

$$\mathbf{s} = \mathbf{F}_{ex}(\mathbf{z}, \mathbf{W}) = \sigma(g(\mathbf{z}, \mathbf{W})) = \sigma\big(\mathbf{W}_2 \delta(\mathbf{W}_1 \mathbf{z})\big) \qquad (2.3)$$

where δ is the rectified linear unit (ReLU) function, σ is the sigmoid function, W_1 and W_2 are fully connected layers and z is the resulting output vector from squeeze operation.

Here, these fully connected layers at the congestion structure complete the following functions: W_1 reduces the dimensionality by the r ratio and W_2 contrariwise increase the dimensionality to restore the channel dimension of U.

As the sigmoid function returns the floats in range [0,1], the channel weights and the total result of SE block can be obtain as follows from equation 4.

$$\tilde{\mathbf{x}}_c = \mathbf{F}_{\text{scale}}(\mathbf{u}_c, s_c) = s_c \mathbf{u}_c \qquad (2.4)$$

2.2.2 Convolutional block attention module

Given an intermediate feature $\mathbf{F} \in \mathbb{R}^{C \times H \times W}$ as input, CBAM sequentially infers a 1D channel attention map $M_c \in \mathbb{R}^{C \times 1 \times 1}$ and a 2D spatial attention map $\mathbf{M_s} \in \mathbb{R}^{1 \times H \times W}$ as illustrated in Fig. 1. The overall attention process can be summarized as:

$$\begin{aligned} \mathbf{F}' &= \mathbf{M_c}(\mathbf{F}) \otimes \mathbf{F} \\ \mathbf{F}'' &= \mathbf{M_s}(\mathbf{F}') \otimes \mathbf{F}' \end{aligned} \qquad (2.5)$$

where $\otimes$ denotes element-wise multiplication. During multiplication, the attention values are broadcasted (copied) accordingly: channel attention values are broadcasted along the spatial dimension, and vice versa. F" is the final refined output. Fig. 2.12.1 depicts the computation process of each attention map. The following describes the details of each attention module.

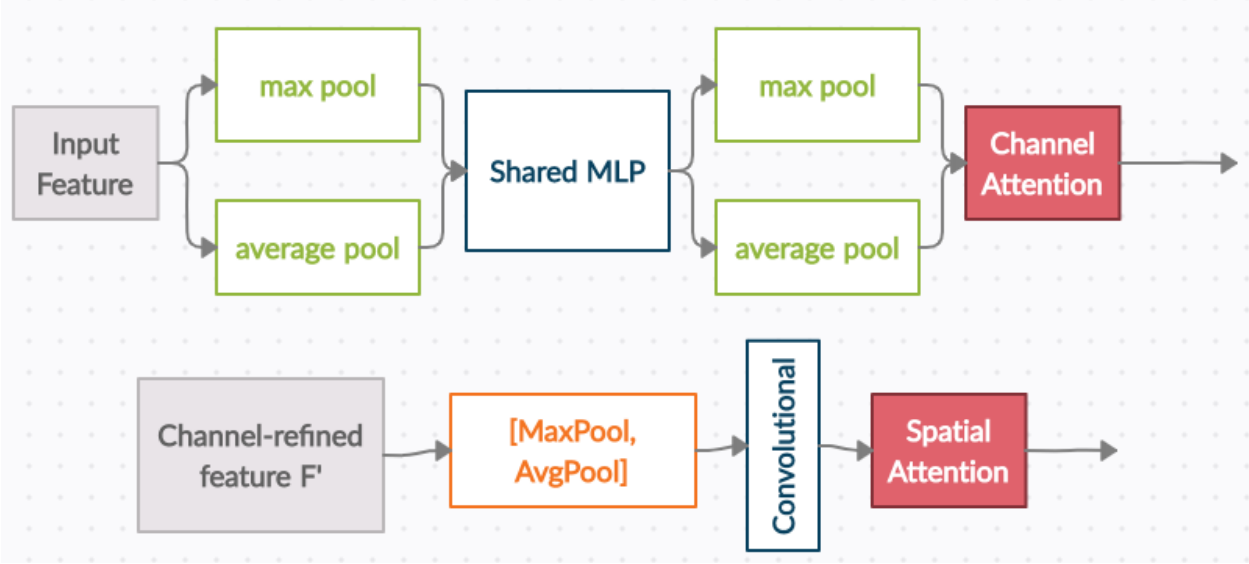


Fig. 2.6 Diagram of each attention sub-module. As illustrated, the channel sub-module utilizes both max-pooling outputs and average-pooling outputs with a shared network; the spatial sub-module utilizes similar two outputs that are pooled along the channel axis and forward them to a convolution layer.

Channel attention module. We produce a channel attention map by exploiting the inter-channel relationship of features. As each channel of a feature map is considered as a feature detector, channel attention focuses on 'what' is meaningful given an input image. To compute the channel attention efficiently, we squeeze the spatial dimension of the input feature map. For aggregating spatial information, average-pooling has been commonly adopted so far. Zhou et al. suggest to use it to learn the extent of the target object effectively and Hu et al. adopt it in their attention module to compute spatial statistics. Beyond the previous works, we argue that max-pooling gathers another important clue about distinctive object features to infer finer channel-wise attention. Thus, we use both average-pooled and max-pooled features simultaneously. We empirically confirmed that exploiting both features greatly improves representation power of networks rather than using each independently, showing the effectiveness of our design choice. We describe the detailed operation below.

We first aggregate spatial information of a feature map by using both average pooling and max-pooling operations, generating two different spatial context descriptors: $\mathbf{F}_{avg}^{C}$ and $\mathbf{F}_{max}^{C}$, which denote average-pooled features and

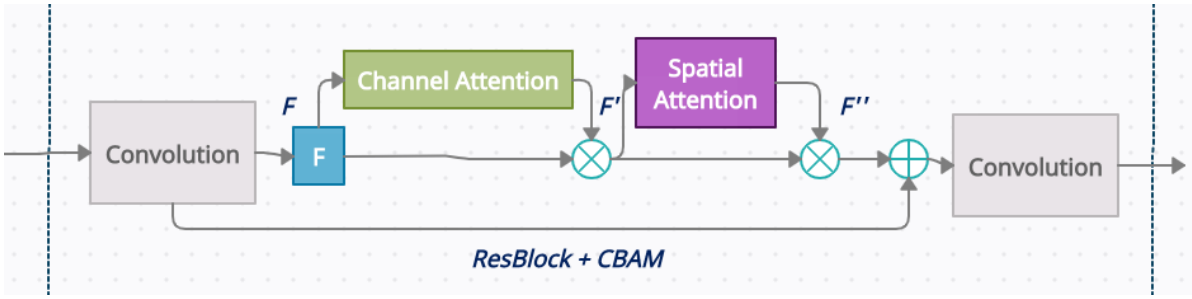


Fig. 2.7 CBAM integrated with a ResBlock in ResNet. This figure shows the exact position of our module when integrated within a ResBlock. We apply CBAM on the convolution outputs in each block.

max-pooled features respectively. Both descriptors are then forwarded to a shared network to produce our channel attention map $M_c \in \mathbb{R}^{C\times1\times1}$. The shared network is composed of multi-layer perceptron (MLP) with one hidden layer. To reduce parameter overhead, the hidden activation size is set to $\mathbb{R}^{C/r\times1\times1}$, where r is the reduction ratio. After the shared network is applied to each descriptor, we merge the output feature vectors using element-wise summation. In short, the channel attention is computed as:

$$\begin{aligned} M_c(F) &= \sigma(MLP(AvgPool(\mathbf{F})) + MLP(\text{MaxPool}(\mathbf{F}))) \\ &= \sigma\left(\mathbf{W}_1\left(\mathbf{W}_0\left(\mathbf{F}^{\mathbf{c}}_{\text{avg}}\right)\right) + \mathbf{W}_1\left(\mathbf{W}_0\left(\mathbf{F}^{\mathbf{c}}_{max}\right)\right)\right) \end{aligned} \tag{2.6}$$

where σ denotes the sigmoid function, $\mathbf{W}_0 \in \mathbb{R}^{C/r\times C}$, and $\mathbf{W}_1 \in \mathbb{R}^{C\times C/r}$. Note that the MLP weights, W_0 and W_1, are shared for both inputs and the ReLU activation function is followed by W_0.

Spatial attention module. We generate a spatial attention map by utilizing the inter-spatial relationship of features. Different from the channel attention, the spatial attention focuses on 'where' is an informative part, which is complementary to the channel attention. To compute the spatial attention, we first apply average-pooling and max-pooling operations along the channel axis and concatenate them to generate an efficient feature descriptor. Applying pooling operations along the channel axis is shown to be effective in highlighting informative regions [33]. On the concatenated feature descriptor, we apply a convolution layer to generate a

spatial attention map $\mathbf{M_s}(\mathbf{F}) \in \mathbf{R}^{H \times W}$ which encodes where to emphasize or suppress. We describe the detailed operation below.

We aggregate channel information of a feature map by using two pooling operations, generating two 2D maps: $\mathbf{F}^{s}_{\text{avg}} \in \mathbb{R}^{1 \times H \times W}$ and $\mathbf{F}^{s}_{max} \in \mathbb{R}^{1 \times H \times W}$. Each denotes average-pooled features and max-pooled features across the channel. Those are then concatenated and convolved by a standard convolution layer, producing our 2D spatial attention map. In short, the spatial attention is computed as:

$$\begin{aligned} \mathbf{M_s}(\mathbf{F}) &= \sigma\left(f^{7\times7}([AvgP_{ool}(\mathbf{F}); MaxP_{ool}(\mathbf{F})])\right) \\ &= \sigma\left(f^{7\times7}\left(\left[\mathbf{F^s_{avg}}; \mathbf{F}^{s}_{max}\right]\right)\right) \end{aligned} \tag{2.7}$$

where σ denotes the sigmoid function and $f^{7\times7}$ represents a convolution operation with the filter size of 7 x 7.

Arrangement of attention modules. Given an input image, two attention modules, channel and spatial, compute complementary attention, focusing on 'what' and 'where' respectively. Considering this, two modules can be placed in a parallel or sequential manner. We found that the sequential arrangement gives a better result than a parallel arrangement. For the arrangement of the sequential process, our experimental result shows that the channel-first order is slightly better than the spatial-first.

2.2.3 PolyNet. PolyInception Module

We develop a new family of building blocks called PolyInception modules, which generalize the Inception residual units [25] via various forms of polynomial compositions. This new design not only encourages the structural diversity but also enhances the expressive power of the residual components.

To begin with, we first revisit the basic design of a residual unit. Each unit comprises two paths from input to output, namely, an identity path and a residual block. The former preserves the input while the latter turns the input into residues

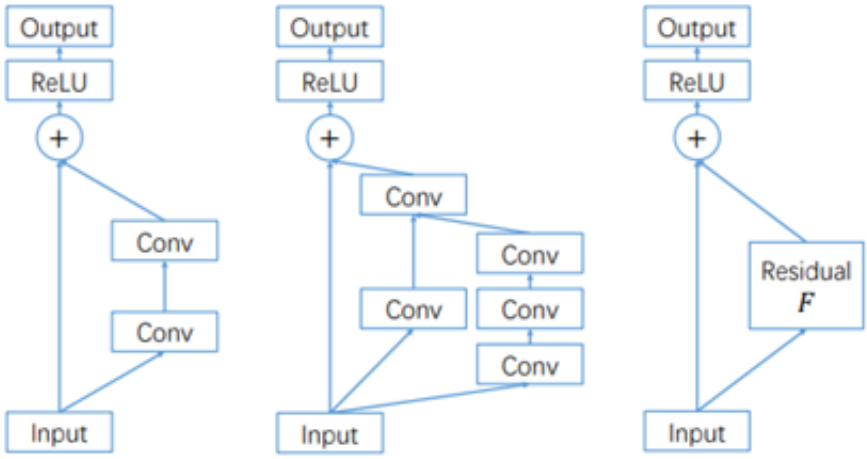


Fig. 2.8 Left: residual unit of ResNet, Middle: type-B Inception residual unit, Right: abstract residual unit structure where the residual block is denoted by F.

via nonlinear transforms. The results of these two paths are added together at the output end of the unit, as

$$(I + F) \cdot \mathbf{x} = \mathbf{x} + F \cdot \mathbf{x} := \mathbf{x} + F(\mathbf{x}) \qquad (2.8)$$

This formula represents the computation with operator notations: x denotes the input, I the identity operator, and F the nonlinear transform carried out by the residual block, which can also be considered as an operator. Moreover, F · x indicates result of the operator F acting on x. When F is implemented by an Inception block, the entire unit becomes an Inception residual unit as shown in Fig. 3. In standard formulations, including ResNet [9] and InceptionResNet [19], there is a common issue – the residual block is very shallow, containing only 2 to 4 convolutional layers. This restricts the capacity of each unit, and as a result, it needs to take a large number of units to build up the overall expressive power. Whereas several variants have been proposed to enhance the residual units [1, 28, 34], the improvement, however, remains limited. [15]

Driven by the pursuit of structural diversity, we study an alternative approach, that is, to generalize the additive combination of operators in Eq. (1) to polynomial compositions. For example, a natural extension of Eq. (1) along this line is to simply add a second-order term, which would result in a new computation unit as

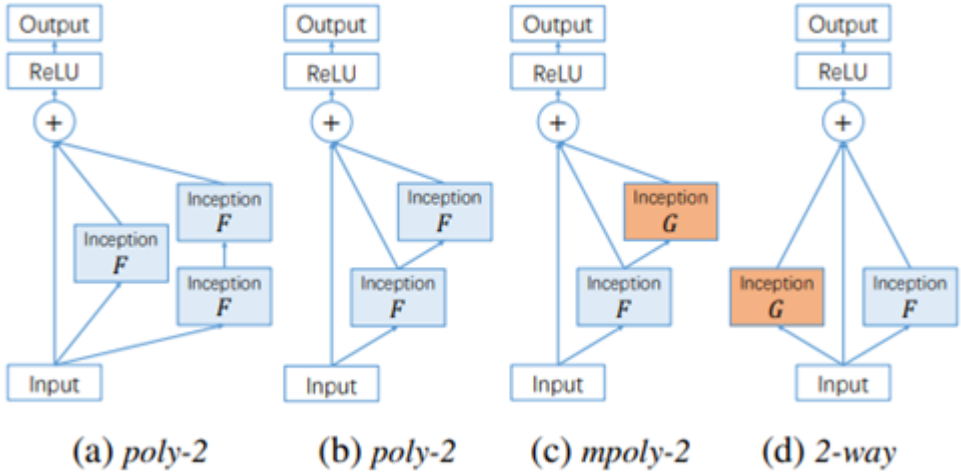


Fig. 2.9 List of PolyNet structures

$$(I + F + F^2) \cdot \mathbf{x} := \mathbf{x} + F(\mathbf{x}) + F(F(\mathbf{x})) \qquad (2.9)$$

The corresponding architecture is shown in Fig. 4 (a). This unit comprises three paths, namely, an identity path, a firstorder path that contains one Inception block, and a second order path that contains two. The signals resulted from these paths will be summed together at the output end, before the ReLU layer. Here, the second-order path allows the input signal to go through deeper transformations before being merged back to the main path. This can considerably increase the expressive power of the unit.

It is noteworthy that along both the first-order and second-order paths, the outputs of the first Inception block are the same, that is, F(x). Taking advantage of this observation, we can reduce the architecture in Fig. 4 (a) to a cascaded form as shown in Fig. 4 (b). These two forms are mathematically equivalent; however, the computational cost of the latter is only 2/3 of the former.

Note that the cascaded form can be reflected by an algebraic rewriting as I + (I +F)F. As such an extension embodies a polynomial composition of the Inception blocks, we call it a PolyInception module. The maximum number of Inception blocks along a path is referred to as the order of the module.

Figure 4 illustrates several PolyInception modules which differ in how the Inception blocks are composed and whether they share parameters. For

convenience, we refer to them as poly-2, mpoly-2, and 2-way respectively. The following discussion compares their characteristics.

• poly-2: $I + F + F^2$. This unit contains two Inception blocks, both denoted by F. This is a notational convention indicating that they share parameters. This design can increase the representation power without introducing additional parameters.

• mpoly-2: $I + F + GF$. The structure of this design is similar to poly-2, except that the two Inception blocks do not share parameters. Similar to poly-2, the computation of the first layer along both 1st- and 2nd-order paths can be shared (both denoted by F), and thus it can also be reduced into an equivalent cascaded form (I+(I+G)F), as shown in Fig. 4 (c). Compare to poly-2, it possesses stronger expressive power. At the same time, however, the parameter size also doubles.

• 2-way: $I + F + G$. This is a first-order PolyInception, with an extra first-order path incorporated into the unit. This construction is similar to the building blocks used in the Multiple Residual Network [1].

These exemplar designs can be further extended based on the relations between operator polynomials and network architectures. For instance, poly-2 and mpoly-2 can be extended to higher-order PolyInception modules, namely, poly-3 ($I + F + F^2 + F^3$) and mpoly-3 (I+F +GF +HGF). Compared to their 2nd-order counterparts, the extended versions have greater expressive power, but also incur increased computational cost. Similarly, 2-way can also be extended to 3-way.

2.2.4 ResNeXt block

We adopt a highly modularized design following VGG/ResNets. Our network consists of a stack of residual blocks. These blocks have the same topology, and are subject to two simple rules inspired by VGG/ResNets: (i) if producing spatial maps of the same size, the blocks share the same hyper-parameters (width and filter sizes), and (ii) each time when the spatial map is downsampled by a factor of 2, the width of the blocks is multiplied by a factor of

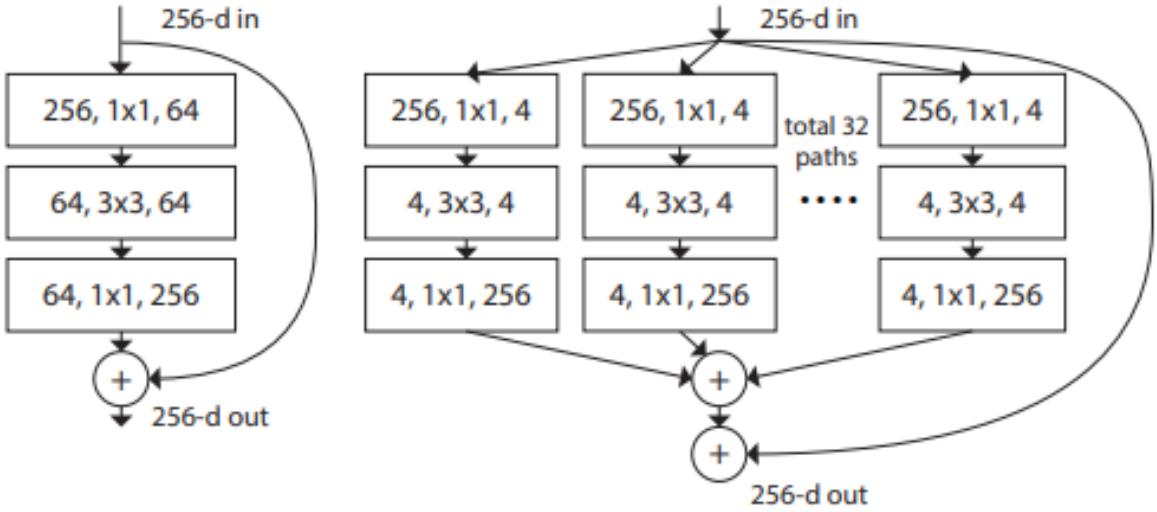


Fig. 2.10 Left: A block of ResNet, Right: a block of ResNeXt with cardinality = 32, with roughly the same complexity. A layer is shown as (# in channels, filter size, # out channels)

2. The second rule ensures that the computational complexity, in terms of FLOPs (floating-point operations, in # of multiply-adds), is roughly the same for all blocks.

With these two rules, we only need to design a template module, and all modules in a network can be determined accordingly. So these two rules greatly narrow down the design space and allow us to focus on a few key factors.

Revisiting Simple Neurons. The simplest neurons in artificial neural networks perform inner product (weighted sum), which is the elementary transformation done by fully-connected and convolutional layers. Inner product can be thought of as a form of aggregating transformation:

$$\sum_{i=1}^{D} w_i x_i \qquad (2.9)$$

where $x = [x_1, x_2, ..., x_D]$ is a D-channel input vector to the neuron and w_i is a filter's weight for the i-th channel. This operation (usually including some output nonlinearity) is referred to as a "neuron". See Fig. 2.

The above operation can be recast as a combination of splitting, transforming, and aggregating.

Splitting: the vector x is sliced as a low-dimensional embedding, and in the above, it is a single-dimension subspace x_i.

Transforming: the low-dimensional representation is transformed, and in the above, it is simply scaled: $w_i x_i$.

Aggregating: the transformations in all embeddings are aggregated by $\sum_{i=1}^{D}$.

Aggregated Transformations. Given the above analysis of a simple neuron, we consider replacing the elementary transformation ($w_i x_i$) with a more generic function, which in itself can also be a network. In contrast to "Network-in-Network" [26] that turns out to increase the dimension of depth, we show that our "Network-in-Neuron" expands along a new dimension. Formally, we present aggregated transformations as:

$$\mathcal{F}(\mathbf{x}) = \sum_{i=1}^{C} \mathcal{T}_i(\mathbf{x}) \qquad (2.10)$$

where $\mathcal{T}_i$ (x) can be an arbitrary function. Analogous to a simple neuron, $\mathcal{T}_i$ should project x into an (optionally lowdimensional) embedding and then transform it. In Eqn.(2.10), C is the size of the set of transformations to be aggregated. We refer to C as cardinality [2]. In Eqn.(2.10) C is in a position similar to D in Eqn.(1), but C need not equal D and can be an arbitrary number. While the dimension of width is related to the number of simple transformations (inner product), we argue that the dimension of cardinality controls the number of more complex transformations. We show by experiments that cardinality is an essential dimension and can be more effective than the dimensions of width and depth.

In this paper, we consider a simple way of designing the transformation functions: all $\mathcal{T}_i$'s have the same topology. This extends the VGG-style strategy of repeating layers of the same shape, which is helpful for isolating a few factors and extending to any large number of transformations. We set the individual transformation $\mathcal{T}_i$ to be the bottleneckshaped architecture [14], as illustrated in Fig. 1 (right). In this case, the first 1×1 layer in each $\mathcal{T}_i$ produces the lowdimensional embedding.

The aggregated transformation in Eqn.(2) serves as the residual function [14] (Fig. 1 right):

$$\mathbf{y} = \mathbf{x} + \sum_{i=1}^{C} \mathcal{T}_i(\mathbf{x}) \qquad (2.11)$$

where y is the output.

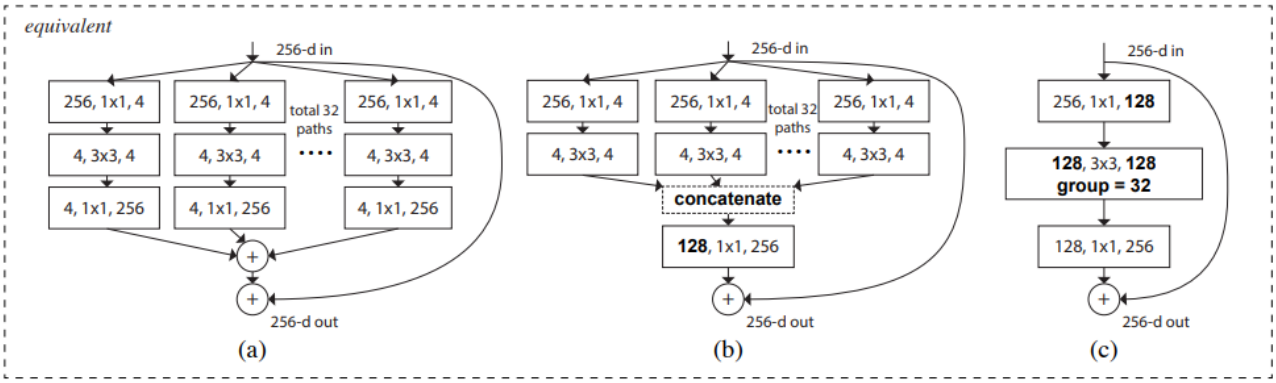


Fig. 2.11 Equivalent building blocks of ResNeXt. (a): Aggregated residual transformations, the same as Fig. 1 right. (b): A block equivalent to (a), implemented as early concatenation. (c): A block equivalent to (a,b), implemented as grouped convolutions [24]. Notations in bold text highlight the reformulation changes. A layer is denoted as (# input channels, filter size, # output channels).

Relation to Inception-ResNet. Some tensor manipulations show that the module in Fig. 1(right) (also shown in Fig. 3(a)) is equivalent to Fig. 2.11(b).3 Fig. 3(b) appears similar to the Inception-ResNet [14] block in that it involves branching and concatenating in the residual function. But unlike all Inception or Inception-ResNet modules, we share the same topology among the multiple paths. Our module requires minimal extra effort designing each path.

Relation to Grouped Convolutions. The above module becomes more succinct using the notation of grouped convolutions [24].4 This reformulation is illustrated in Fig. 3(c). All the low-dimensional embeddings (the first 1×1 layers) can be replaced by a single, wider layer (e.g., 1×1, 128-d in Fig 3(c)). Splitting is essentially done by the grouped convolutional layer when it divides its input channels into groups. The grouped convolutional layer in Fig. 3(c) performs 32 groups of convolutions whose input and output channels are 4-dimensional. The grouped convolutional layer concatenates them as the outputs of the layer. The block in Fig. 3(c) looks like the original bottleneck residual block in Fig. 1(left), except that Fig. 3(c) is a wider but sparsely connected module.

2.2.5 Attention Module (Residual Attention Neural Network)

Our Residual Attention Network is constructed by stacking multiple Attention Modules. Each Attention Module is divided into two branches: mask branch and trunk branch. The trunk branch performs feature processing and can be adapted to any state-of-the-art network structures. In this work, we use pre-activation Residual Unit, ResNeXt and Inception as our Residual Attention Networks basic unit to construct Attention Module. Given trunk branch output T(x) with input x, the mask branch uses bottom-up top-down structure [5] to learn same size mask M(x) that softly weight output features T(x). The bottom-up top-down structure mimics the fast feedforward and feedback attention process. The output mask is used as control gates for neurons of trunk branch similar to Highway Network [29]. The output of Attention Module H is:

$$H_{i,c}(x) = M_{i,c}(x) * T_{i,c}(x) \tag{2.12}$$

where i ranges over all spatial positions and c ∈ {1, ..., C} is the index of the channel. The whole structure can be trained end-to-end.

In Attention Modules, the attention mask can not only serve as a feature selector during forward inference, but also as a gradient update filter during back propagation. In the soft mask branch, the gradient of mask for input feature is:

$$\frac{\partial M(x,\theta)T(x,\phi)}{\partial \phi} = M(x,\theta)\frac{\partial T(x,\phi)}{\partial \phi} \tag{2.13}$$

where the θ are the mask branch parameters and the φ are the trunk branch parameters. This property makes Attention Modules robust to noisy labels. Mask branches can prevent wrong gradients (from noisy labels) to update trunk parameters. Experiment in Sec.4.1 shows the robustness of our Residual Attention Network against noisy labels.

Instead of stacking Attention Modules in our design, a simple approach would be using a single network branch to generate soft weight mask, similar to spatial transformer layer [17]. However, these methods have several drawbacks on

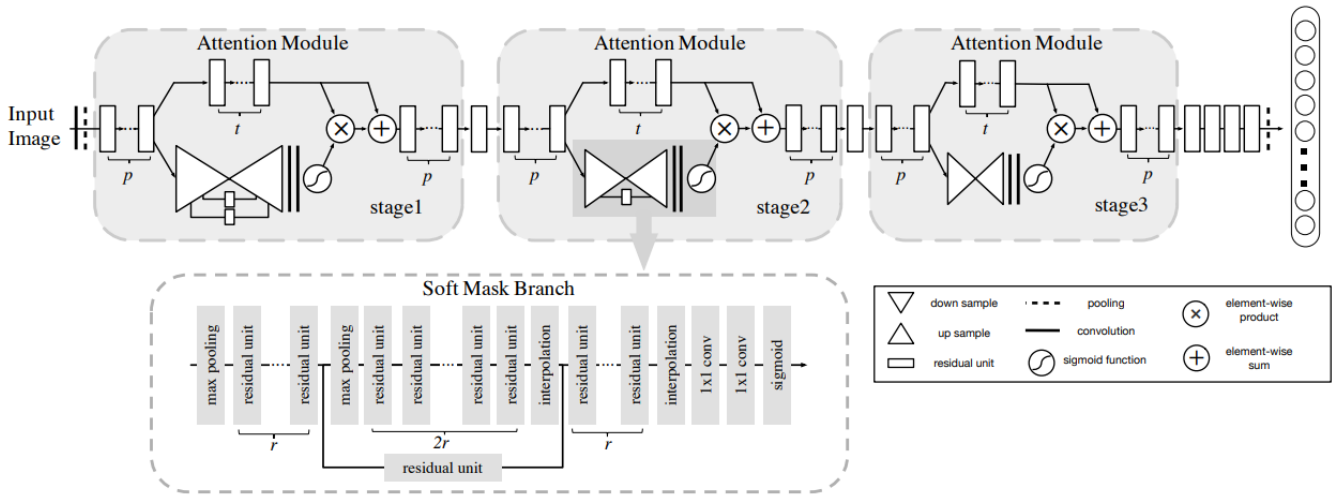


Fig. 2.13 Example architecture of the proposed network for ImageNet. We use three hyper-parameters for the design of Attention Module: p, t and r. The hyper-parameter p denotes the number of pre-processing Residual Units before splitting into trunk branch and mask branch. t denotes the number of Residual Units in trunk branch. r denotes the number of Residual Units between adjacent pooling layer in the mask branch. In our experiments, we use the following hyper-parameters setting: {p = 1, t = 2, r = 1}. The number of channels in the soft mask Residual Unit and corresponding trunk branches is the same.

challenging datasets such as ImageNet. First, images with clutter background, complex scenes, and large appearance variations need to be modeled by different types of attentions. In this case, features from different layers need to be modeled by different attention masks. Using a single mask branch would require exponential number of channels to capture all combinations of different factors. Second, a single Attention Module only modify the features once. If the modification fails on some parts of the image, the following network modules do not get a second chance.

The Residual Attention Network alleviates above problems. In Attention Module, each trunk branch has its own mask branch to learn attention that is specialized for its features. As shown in Fig.1, in hot air balloon images, blue color features from bottom layer have corresponding sky mask to eliminate background, while part features from top layer are refined by balloon instance mask. Besides, the incremental nature of stacked network structure can gradually refine attention for complex images.

However, naive stacking Attention Modules leads to the obvious performance drop. First, dot production with mask range from zero to one

repeatedly will degrade the value of features in deep layers. Second, soft mask can potentially break good property of trunk branch, for example, the identical mapping of Residual Unit.

We propose attention residual learning to ease the above problems. Similar to ideas in residual learning, if soft mask unit can be constructed as identical mapping, the performances should be no worse than its counterpart without attention. Thus we modify output H of Attention Module as

$$H_{i,c}(x) = \left(1 + M_{i,c}(x)\right) * F_{i,c}(x) \qquad (2.14)$$

M(x) ranges from [0, 1], with M(x) approximating 0, H(x) will approximate original features F(x). We call this method attention residual learning. Our stacked attention residual learning is different from residual learning. In the origin ResNet, residual learning is formulated as $H_{i,c}(x) = x+F_{i,c}(x)$, where $F_{i,c}(x)$ approximates the residual function. In our formulation, $F_{i,c}(x)$ indicates the features generated by deep convolutional networks. The key lies on our mask branches M(x). They work as feature selectors which enhance good features and suppress noises from trunk features.

In addition, stacking Attention Modules backs up attention residual learning by its incremental nature. Attention residual learning can keep good properties of original features, but also gives them the ability to bypass soft mask branch and forward to top layers to weaken mask branch's feature selection ability. Stacked Attention Modules can gradually refine the feature maps.

3. STRUCTURAL PARAMETRIC SYNTHESIS OF CONVOLUTIONAL NEURAL NETWORK

3.1 Problem definition of structural parametric synthesis

In the terms of structural parametric synthesis, it's necessary to solve the problem of generating the convolutional neural network structure, configuration of the networks parameters, training process and its practical implementation. In the previous chapter there were presented the CNN blocks and architectures which will be used and the base in our structural parametric synthesis algorithm.

The problem definition can be described as follows: complete set is specified $J = \{R_j, Y_j\} j = 1, \dots, P$ pairs of type "attribute-value", where R_j, Y_j input and output vector of neural network, respectively.

It is necessary to synthesize such an optimal CNN on the basis of a training sample J, which would provide an effective solution to the applied problem (classification, approximation, prediction). The vector optimality criterion is defined as

$$\mathbf{I} = \{I_1(x), I_2(x)\} \rightarrow \text{ opt} \qquad (3.1)$$

where $I_1(x) = E_{\text{yar}}(x)$ generalization error, which determines the magnitude of the error of solving the problem in the test sample; $I_2(x) = S(x)$ is the neural network complexity (number of interneural connections); X is a vector of significant parameters.

While the choosing the suitable neural network structure the following parameters should be counted:

- List of blocks used in CNN;
- Network depth;

- Number of blocks;
- GFLOPs value at training the neural network;
- Difficulty of input data set preparation.

After confirming the architecture the parametric synthesis should be done. There're such coefficients that should be configure:

- Convolution layer: numbers of feature maps, stride size, filter size.
- Pooling layer: kernel filter size, aggregation function.
- FC Layer: number of FC layers, layers size, encoder type.
- Reduction ratio, etc;

Finally to train and test the received network we need to prepare the input data set full of images that should require few requirements. Also, the program implementation of the CNN should process and apply such random transformations as:

- Random mirror;
- Random crop;
- Aspect ratio;
- Random rotation;
- Pixel Jitter;

3.2 Structural synthesis of convolutional neural network

Structural synthesis is the algorithm of generating the suitable structure of CNN. The one of many approaches is genetic algorithm shown below. The idea is generate the number of different architectures by putting the random blocks in each stage and then by means of testing choose few with the best performance.

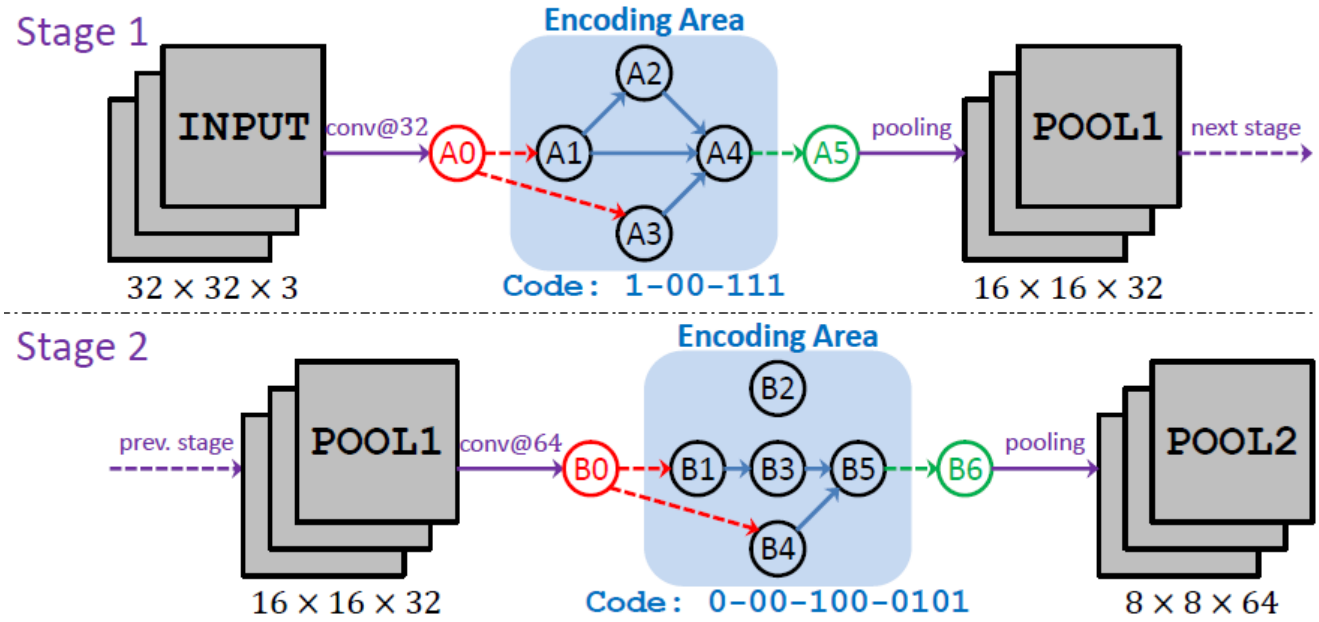


Fig. 3.1 Representation on genetic algorithm on the example of two stage network

Then we shuffle this architecture parts with a lot of different variations, testing them and again choosing ones with best performance. By doing this operations over and over we'll get the most suitable architecture.

In this work I'm proposing to use the genetic algorithms to generate the convolutional neural network architecture using the following CNN building blocks presented in Chapter 2:

- Squeeze and Excitation block based on ResNet module;
- Squeeze and Excitation block based on PolyInception module;
- Densely connected block;
- Convolutional block attention module;
- Residual attention module;
- ResNeXt block;
- and common convolutional layer, fully connected layer, etc.

The some requirements to the blocks to be able connected within each other. While connecting two random blocks the important role takes the coincidence of output data depth or dimensionality of first one and the convolutional filters depth

of the second ones. The block should be able to process that type of data that expected to come from layer before.

Now we can dive into the genetic algorithm implementation. We provide a binary string representation for a network structure in a constrained case. Note that many state-of-the-art network structures can be partitioned into several stages. In each stage, the geometric dimensions (width, height and depth) of the layer cube remain unchanged. Neighboring stages are connected via a spatial pooling operation, which may change the spatial resolution. All the convolutional operations within one stage have the same number of filters, or channels. [16]

We borrow this idea to define a family of networks which can be encoded into fixed-length binary strings. A network is composed of S stages, and the s-th stage, s = 1,2,…,S, contains K_s nodes, denoted by $v_{s,ks}$, k_s = 1, 2,…,K_s. The nodes within each stage are ordered, and we only allow connections from a lower-numbered node to a highernumbered node. Each node corresponds to a convolutional operation, which takes place after element-wise summing up all its input nodes (lower-numbered nodes that are connected to it). After convolution, batch normalization and ReLU are followed, which are verified efficient in training very deep neural networks. We do not encode the fully-connected part of a network.

In each stage, we use $1 + 2 + \ldots + (K_s - 1) = ½ K_s (K_s - 1)$ bits to encode the inter-node connections. The first bit represents the connection between ($v_{s,1}$, $v_{s,2}$), then the following two bits represent the connection between ($v_{s,1}$, $v_{s,3}$) and ($v_{s,2}$, $v_{s,3}$), etc. This process continues until the last $K_s - 1$ bits are used to represent the connection between $v_{s,1}$, $v_{s,2}$,…, $v_{s,Ks - 1}$ and $v_{s,Ks}$. For $1 < i < j < K_s$, if the code corresponding to ($v_{s,i}$; $v_{s,j}$) is 1, there is an edge connecting $v_{s,i}$ and $v_{s,j}$, i.e., $v_{s,j}$ takes the output of $v_{s,i}$ as a part of the element-wise summation, and vice versa.

Figure 2.4.1 illustrates two examples of network encoding. To summarize, a S-stage network with Ks nodes at the s-th stage is encoded into a binary string with length $L = \frac{1}{2}\sum_s K_s(K_s - 1)$. Equivalently, there are in total 2^L possible network

structures. This number may be very large. In the CIFAR10 experiments (see Section 4.2), we have S = 3 and (K_1, K_2, K_3) = (3, 4, 5), therefore L = 19 and 2^L = 524;288. It is computationally intractable to enumerate all these structures and find the optimal one(s).

The genetic algorithm process for neural network can be divided into following steps:

1. Input: the reference dataset D, the number of generations T, the number of individuals in each generation N, the mutation and crossover probabilities p_M and p_C, the mutation parameter q_M, and the crossover parameter q_C.

2. Initialization: generating a set of randomized individuals $\{\mathbb{M}_{0,n}\}_{n=1}^{N}$, and computing their recognition accuracies.

3. for t = 1, 2, …, T do:

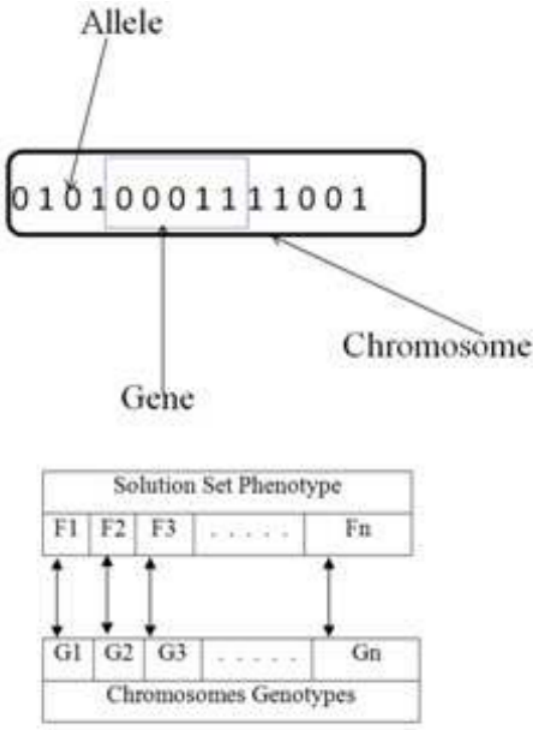


Fig. 3.2 Representation of allele, gene, chromosome, genotype and phenotype adapted from

a. Selection: producing a new generation $\{\mathbb{M}'_{t,n}\}_{n=1}^{N}$ with a Russian roulette process on $\{\mathbb{M}_{t-1,n}\}_{n=1}^{N}$.

b. Crossover: for each pair $\{(\mathbb{M}_{t,2n-1}, \mathbb{M}_{t,2n})\}_{n=1}^{\lfloor N/2 \rfloor}$, performing crossover with probability p_C and parameter q_C.

c. Mutation: for each non-crossover individual $\{\mathbb{M}_{t,n}\}_{n=1}^{N}$, doing mutation with probability p_M and parameter q_M.

d. Evaluation: computing the recognition accuracy for each new individual $\{\mathbb{M}_{t,n}\}_{n=1}^{N}$.

4. End for.

5. Output: a set of individuals in the final generation $\{\mathbb{M}_{T,n}\}_{n=1}^{N}$ with their recognition accuracies.

Basic instructions for building GAform a gene (bit strings of arbitrary length). A sequence of genes is called a chromosome. Possible solution to a problem may be described by genes without really being the answer to the problem. The smallest unit in chromosomes is called an allele represented by a single symbol or binary bit. A phenotype gives an external description of the individual whereas a genotype is deposited information in a chromosome as presented in Figure 1. Where F_1, F_2, F_3, F_4. . .F_n and G_1, G_2, G_3, G_4 . . .G_n are factors and genes, respectively.

Individuals in a group form a population. The fitness of each individual in the population is evaluated. Individuals with higher fitness produce more offspring than those with lower fitness. Individuals and certain information about the search space are defined by phenotype parameters.

The initial population and population size (pop size) are the two major population features in GA. The population size is usually determined by the nature

of the problem and is initially generated randomly, referred to as population initialization

Crossover. This is a randomly pointed locus in an encoded bit string and the exact number of bits before and after the pointed locus are fragmented and exchanged between the chromosomes of the parents. The offspring are formed by combining fragments of the parents' bit strings. For all offspring to be a product of crossover, the crossover probability (pc) must be 100% but if the probability is 0%, the chromosome of the present offspring will be the exact replica of the old generation.

The reason for crossovers is the reproduction of better chromosomes containing the good parts of the old chromosomes as depicted in Figure 2. Survival of some segment of the old population into the next generation is allowed by the selection process in crossovers. Other crossover algorithms include: two point, multi-point, uniform, three parent, and crossover with reduced surrogate, among others. Single point crossover is considered superior because it does not destroy the building blocks while additional points reduce the Gas performance.

Mutation. This is the creation of offspring from a single parent by inverting one or more randomly selected bits in the chromosomes of the parent as shown in Figure 3.3. Mutation can be achieved on any bit with a small probability, for

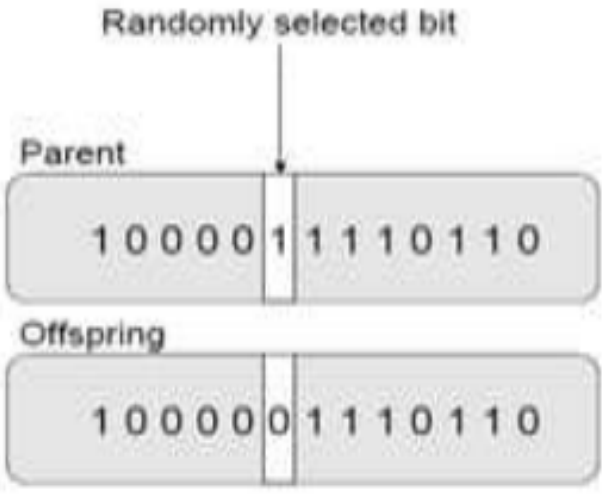


Fig. 3.3 Mutation (single point)

instance 0.001. Strings resulting from the crossover are mutated in order to avoid a local minimum. Genetic materials that may be lost in the process of crossover and the distortion of genetic information are fixed through mutation. Mutation probability (pm) is responsible for determining how frequent will be the section of chromosome subjected to mutation. Thus, the decision to mutate a section of the chromosome depends on the pm.

If mutation is not applied, the offspring are generated immediately from crossover without any part of the chromosomes being tempered. A 100% probability of mutation means the entire chromosome will be changed but if the probability is 0%, it indicates none of the chromosome parts will be distorted. Mutation prevents GA from being trapped in the local maximum.

Figure 3.3 shows mutation for a binary representation.

When a problem is given as an input, the fundamental idea of GA is that the pool of genetics specifically contains the population with a potential solution or better solution to the problem. GA use the principle of genetics and evolution to recurrently modify a population of artificial structures through the use of operators, including initialization, selection, crossover and mutation, in order to obtain an optimum solution. Normally, GA start with a randomly generated initial population represented by chromosomes. Solutions derived from one population are taken and used to form the next generation population. This is carried out with the expectation that solutions in a new population are better than those in the old population.

The solution used to generate the next solution is selected based on its fitness value; solutions with a higher fitness value have higher chances of being selected for reproduction, while solutions with lower fitness values have a lower chance of being selected for reproduction. This evolution process is repeated several times until a set criterion for termination is satisfied. For instance, the criterion could be the number in the population or the satisfaction of the improvement of the best solutions.

The problem in NN design is deciding the optimum configurations to solve a problem in a specific domain. The choice of NN topology is considered a very important aspect since inefficient NN topology will cause the NN to fall into a local minima (local minima is a poor weight that pretends to be the best, through which back-propagation training algorithms can be deceived from reaching the optimal solution). The problem of deciding suitable architectural configurations and optimum NN weights is a complex task in the area of NN design. Parameter settings and the NN architecture affect the effectiveness of the BPNN as mentionedearlier. The optimum number of layers and neurons in the hidden whereas there is no clear theory for choosing the appropriate parameter setting. GA have been widely used in different problem domains for automatic NN-topology design, in order to deviate from problems attributed to its design, so as to improve its performance and reliability. [12]

The NN topology, constitutes the learning rate, number of epochs, momentum, number of hidden layers, number of neurons; (input neurons and output neurons), error rate, partition ratio of training, validation and testing data sets. In the case of RBFN, finding the center and width in the hidden layer and the connection weights from the hidden to the output layer. The GA is applied to optimize the topology of an NN and applied it to model the spatial interaction data. GA is used to obtain the optimum configuration of the NN topology. Then, he successfully used his model to predict the rate of nitride oxidization. GA is used to obtain the optimum topology of the BPNN and developed a model for estimating the cost of building construction. Optimizing subsets of features, number of time delays and TDNN topology based on a GA search. A TDNN model was built to detect a temporal pattern in stock markets. By repeating a similar study using an ATDNN and a GA was used to optimize the number of time delays and the ATDNN topology. The result obtained with the ATDNN model was superior to that of the TDNN. A fault detection system was designed using an NN in which its topology was optimized based on a GA search.

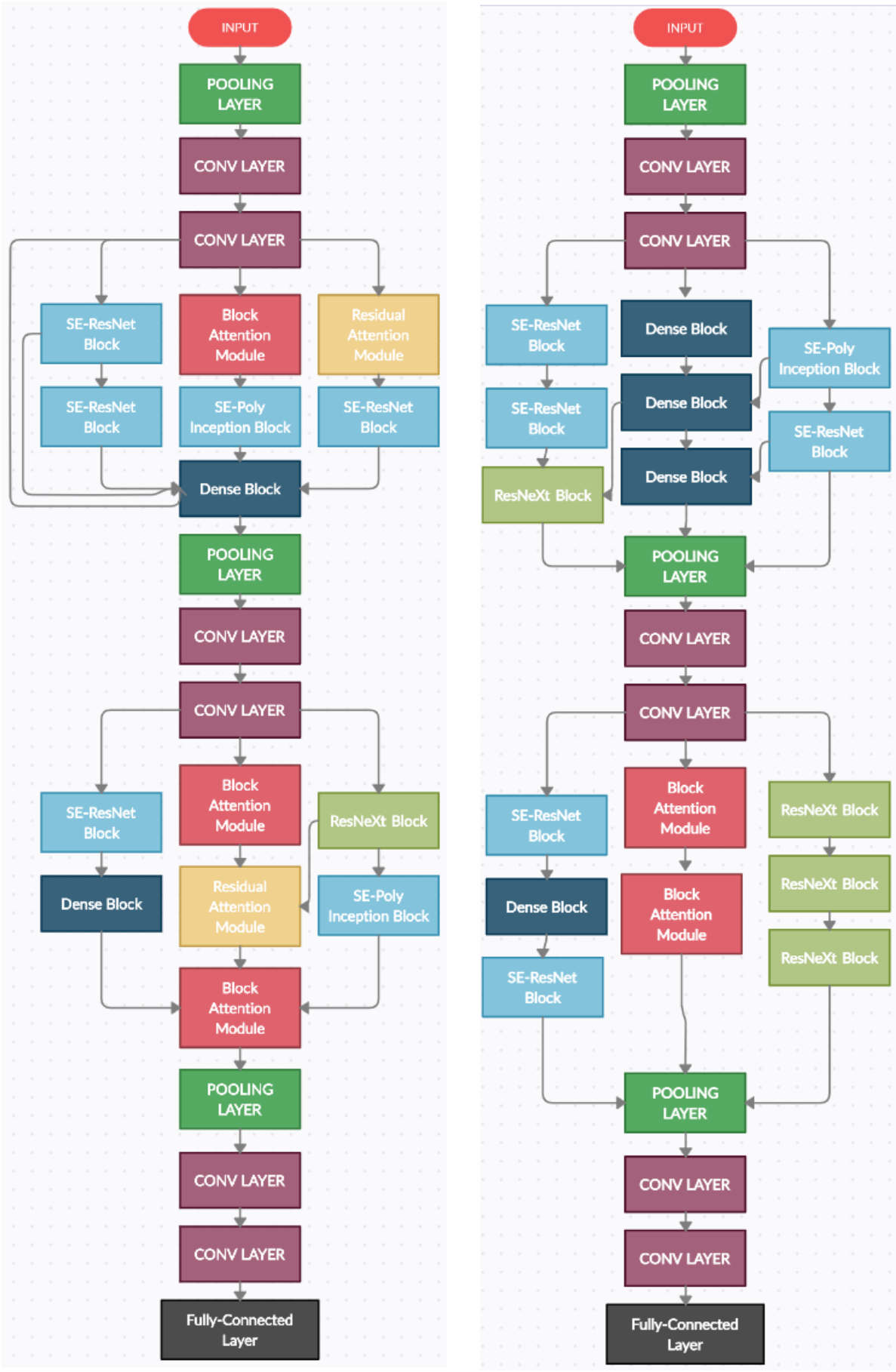


Fig. 3.4 Randomly generated CNN architectures using the genetic algorithm and special blocks

112×112	conv, 7×7, 64, stride 2
56×56	max pool, 3×3, stride 2
	$\begin{bmatrix} \text{conv}, 1 \times 1, 64 \\ \text{conv}, 3 \times 3, 64 \\ \text{conv}, 1 \times 1, 256 \\ fc, [16, 256] \end{bmatrix} \times 3$
28×28	$\begin{bmatrix} \text{conv}, 1 \times 1, 128 \\ \text{conv}, 3 \times 3, 128 \\ \text{conv}, 1 \times 1, 512 \\ fc, [32, 512] \end{bmatrix} \times 4$
14×14	$\begin{bmatrix} \text{conv}, 1 \times 1, 256 \\ \text{conv}, 3 \times 3, 256 \\ \text{conv}, 1 \times 1, 1024 \\ fc, [64, 1024] \end{bmatrix} \times 6$
7×7	$\begin{bmatrix} \text{conv}, 1 \times 1, 512 \\ \text{conv}, 3 \times 3, 512 \\ \text{conv}, 1 \times 1, 2048 \\ fc, [128, 2048] \end{bmatrix} \times 3$
1×1	global average pool, 1000-d fc, softmax

Fig. 3.5 SE CNN architecture as the presentation of generating the different numbers of convolutional and other layers by means of genetic algorithm

The structure of following network can be divided into seven different blocks. At the beginning of the flow placed the simple convolutional layer sized by 7x7 with the stride length of 2. This layer will learn 64 filters as it's early placed layer. Then there follows the 3x3 pooling layer that performs the max pool operation on the convolution output.

On the figure 3.4 presented four randomly generated CNN architectures using the block from the presented before list of neural networks and generic algorithm for structural synthesis.

On the figure 3.5 presented the structure received from genetic algorithm which shows not the relationships and random block generation as on figure 3.4 but the generating the different numbers of different layers and their base internal

configuration values. There, after the first pooling layer are placed three of the specific SE-ResNet blocks with following parameters:

- [Convolutional, 1x1, 64 filters; Convolutional, 3x3, 64 filters; Convolutional, 1x1, 256 filters; Fully-Connected 16, 256]. The output will be 56x56 pixels.

Then there four of:

- [Convolutional, 1x1, 128 filters; Convolutional, 3x3, 128 filters; Convolutional, 1x1, 512 filters; Fully-Connected 32, 512]. Output will be 28 x 28 pixels.

Now six of:

- [Convolutional, 1x1, 256 filters; Convolutional, 3x3, 256 filters; Convolutional, 1x1, 1024 filters; Fully-Connected 64, 1024]. Output will be 14x14 pixels.

Then three of:

- [Convolutional, 1x1, 512 filters; Convolutional, 3x3, 512 filters; Convolutional, 1x1, 2048 filters; Fully-Connected 128, 2048].

Finally, resulting data appears on global average pooling layer, fully-connected layer and softmax function.

3.3 Parametric synthesis of convolutional neural network

Parametric synthesis is the approach of configuring the convolutional neural network to achieve the best possible performance. The parameters list that should be configured can be divided into two groups. Fist one is the local parameters of the convolution layers, pooling layers and other local values for the partials of the CNN building blocks. The second one is the global values such as learning rate, shifts, data noising, reduction ratio, etc.

The local parameters group consists of:

• Number of filters: It is also necessary to decide for each convolutional layer, the number of filters or neurons. This hyperparameter is crucial to allow the transmission of information to the deeper layers, as a too-small number of filters can lead to significant information loss in that particular layer. Again we must account for overfitting when augmenting this value.

• Kernel size: Also for each convolutional layer, we must decide the size of the convolution or kernel. For consistency reasons, all the kernels in a layer are the same size, however, the optimal size is dependent on the problem and the structure of the rest of the network. This parameter is crucial to allow the network to learn the patterns in their corresponding scales.

• Stride size: The shift size on which the filter moves by one step.

• Also there: kernel filter size, aggregation function, convolution filter size, etc.

Now, while knowing the parameters that should be configured, using the genetic algorithm we can configure our neural network. The genetic algorithm for CNN configuring looks like:

Input: (max_gen, cross_prob, mutation_prob, max_pop, conv_parameters)

1. generation <- 0
2. Population <- Generate initial population from input parameters
3. while generation < max_gen do
    a. for individual in population do
        i. individual.accuracy <- obtain validation accuracy value from training network define by individual's chromosome and conv_parameters
    b. end for
    c. Calculate each individual fitness from population
    d. elite ← Get individual with best fitness from population
    e. children_list ← Empty list

f. next_pop ← Empty list
g. for individuals in population do
    i. if Random([0,1]) ≤ cross_prob then
        1. Choose parent1 and parent2 from population using proportional roulette wheel selection.
        2. if generation/max_gen < Random([0,1]) then
            a. child1, child2 ← Cross parent1 and parent2 sequentially
        3. else
            a. child1, child2 ← Cross parent1 and parent2 with binary list
        4. end if
        5. Add child1 and child2 to children_list
    ii. End if
    iii. if generation < max_gen/2 then
        1. next_pop ← apply mutation to children_list with probability mutation_prob
    iv. else
        1. survivors ← Select a fraction of survivors from population, according to fitness
        2. Add survivors, children_list and elite to next_pop
        3. next_pop ← apply mutation to next_pop with probability mutation_prob
    v. end if
    vi. Fill next_pop with random individuals or delete random individuals until max_pop total individuals
    vii. if elite not in next_pop then
        1. Replace random individual from next_pop with elite
    viii. End if
    ix. population ← next_population

x. generation ← generation +1

h. end for

4. end while
5. Train population and return the elite

While using such algorithm it is necessary to take view on the fitness evaluation and selection process. To calculate the fitness of an individual we must train it completely to measure its validation accuracy, as shown in line 7 of the algorithm. During the first half of the evolution process, we calculate the fitness of the individual proportionally to its validation accuracy. If we denote by Ai the validation accuracy of an individual i of the population P, its fitness fi can be computed by:

$$f_i = \frac{A_i}{\sum_{j \in P} A_j} \qquad (3.2)$$

As we can see, the fitness of an individual represents the fraction it contributes to the total accuracies accumulated in the whole population. While this approach easily differentiates individuals with poor performance from good models in early stages, after a few generations the differences between the best and the worst individual become smaller. Eventually, all individuals have a similar fitness value and the algorithms can no longer differentiate their performances. This is why after reaching the midpoint of evolution, we change the strategy to a rank-based one. To compute this fitness value, we must first rank all individuals according to their accuracies: the best individual will have a rank of 1, the second-best a rank of 2 and so on. Now, if we denote by ri the rank of the individual i, we can compute its fitness by:

$$f_i = \frac{n + 1 - r_i}{\sum_{j \in P} \left(n + 1 - r_j\right)} \qquad (3.3)$$

where n is the number of individuals. This is equivalent to the previous fitness if we replace the validation accuracy by $n + 1 - r_i$, a value that represents how good is the network compared to the rest of the population.

During the evolution process, we use an elitist approach, as shown in line 30 of the algorithm. Also, to maintain a constant number of individuals through the evolution process, we implement two different approaches. During the first half of evolution, the next generation is initialized with the best individual, the mutated individuals, and the children. Then, the desired population size is obtained by generating new random individuals. This means that the only individual that survives unaltered does so through elitism. This is done so to allow for a greater diversity during the first generations, introducing random individuals to each new generation.

During the latter half of evolution, the elite and modified individuals (mutated and children) are also part of the next generation. However, to achieve the desired population size, individuals of the previous generation have a chance to survive to the next one, proportional to their fitness value. This allows some successful individuals from a generation to survive to the next one unchanged, intensifying the search process.

Similarly to structural synthesis, we'll use genetic algorithm to get the best values for these parameters. As the input data to genetic algorithm it is necessary to generate and randomly fill the number of byte arrays with the binary data. Each section of each following byte array will represent one of the values for neural networks configuration.

Now, by using the presented algorithm we'll receive our CNN configuration and we're able to test it on our target image sample.

4. PRACTICAL IMPLEMENTATION OF STRUCTURAL PARAMETRIC SYNTHESIS AND RECEIVED NEURAL NETWORK BY MEANS OF PYTHON

4.1 Choosing the python CNN implementation library

In Python, you should be mindful so as to comprehend reactions. For instance, the cheap capacity to add a component to a rundown, in particular attach, changes the rundown. In a useful language like Lisp, adding another component to a rundown, without changing the first rundown, is a modest task. For instance if x is a rundown containing n components, adding an additional component to the rundown in Python (utilizing annex) is quick, yet it has the symptom of changing the rundown x. To develop another rundown that contains the components of x in addition to another component, without changing the estimation of x, involves replicating the rundown, or utilizing an alternate portrayal for records. In the looking through code, we will utilize an alternate portrayal for records consequently.

The package list that can be used is following:

- Scipy with Numpy
- Matplotlib
- Theano
- Keras
- TensorFlow
- Sci-Kit Learn
- PyTorch
- Caffe

- Theano

The detailed information about then can be founded in other sources. In this work the “Tensorflow” and “Keras” will be used.

4.2 Preparing the image set data

Before giving the information to the AI calculations we have to preprocess our graphical information.

Pursue these means to preprocess the information in Python:

Step 1: Importing the valuable bundles: If we are utilizing Python then this would be the initial step for changing over the information into a specific arrangement, i.e., preprocessing. It very well may be done as pursues:

```
import numpy as np
from sklearn import preprocessing
```

Here we have utilized the accompanying two bundles:

NumPy: Basically NumPy is a broadly useful cluster preparing bundle intended to productively control enormous multi-dimensional varieties of self-assertive records without giving up a lot of speed for little multi-dimensional exhibits.

Sklearn.preprocessing: This bundle gives numerous regular utility capacities and transformer classes to change crude component vectors into a portrayal that is increasingly appropriate for AI calculations.

Step 2: Defining test information: After bringing in the bundles, we have to characterize some example information with the goal that we can apply preprocessing strategies on that information. We will currently characterize the accompanying example information:

```
Input_data = np.array([2.1, -1.9, 5.5],
[-1.5, 2.4, 3.5],
[0.5, -7.9, 5.6],
[5.9, 2.3, -5.8]])
```

Step 3: Applying preprocessing procedure: In this progression, we have to apply any of the preprocessing methods. The accompanying area depicts the information preprocessing methods.

4.3 Genetic algorithm implementation

These are only two dependencies for this program. This is as the neural network infrastructure implemented is a simple version that I created myself. To implement more complex networks, it can be imported "Keras" or "Tensorflow".

```
import random
import numpy as np
```

This is the creation of the class "genetic_algorithm" that holds all the functions that concerns the genetic algorithm and how it is supposed to function. The main function is the execute function, that takes pop_size, generations, threshold, X, y, network as parameters. pop_size is the size of the generated population, generations is the term for epochs, threshold is the loss value that you are satisfied with. X and y are for applications of genetic algorithms for labelled data. You can remove all instances of X and y for problems with no data or unlabelled data. Network is the network structure of the neural network.

```
class genetic_algorithm:
    def execute(pop_size,generations,threshold,X,y,network):
        class Agent:
            def __init__(self,network):
```

This function creates the first population of agents that will be tested:

```
def generate_agents(population, network):
        return [Agent(network) for _ in range(population)]
```

```
class neural_network:
                    def __init__(self,network):
                        self.weights = []
                        self.activations = []
                        for layer in network:
                            if layer[0] != None:
                                input_size = layer[0]
                            else:
                                input_size =
network[network.index(layer)-1][1]
                            output_size = layer[1]
                            activation = layer[2]

self.weights.append(np.random.randn(input_size,output_size))
                            self.activations.append(activation)
                    def propagate(self,data):
                        input_data = data
                        for i in range(len(self.weights)):
                            z = np.dot(input_data,self.weights[i])
                            a = self.activations[i](z)
                            input_data = a
                        yhat = a
                        return yhat
                self.neural_network = neural_network(network)
                self.fitness = 0
```

Fig. 4.1 This script describes the initialization of the weights and the propagation of the network for each agent's neural network

As the example that I am using utilizes labelled data. The fitness function is merely calculating the MSE or the cost function for the predictions.

```
def fitness(agents,X,y):
    for agent in agents:
        yhat = agent.neural_network.propagate(X)
        cost = (yhat - y)**2
        agent.fitness = sum(cost)
    return agents
```

This function mimics the theory of selection in evolution: The best survive while the others are left to die. In this case, their data is forgotten and is not used again.

```
def unflatten(flattened,shapes):
    newarray = []
    index = 0
    for shape in shapes:
        size = np.product(shape)
```

newarray.append(flattened[index : index + size].reshape(shape))

index += size

return newarray

To execute the crossover and mutation functions, the weights need to be flattened and unflattened into the original shapes. The crossover function is one of the most complicated functions in the program. It generates two new "children" agents, whose weights that are replaced as a crossover of wo randomly generated parents. This is the process of creating the weights:

- Flatten the weights of the parents
- Generate two splitting points
- Use the splitting points as indices to set the weights of the two children agents

On the figure 4.2 presented the full process of the crossover of agents.

```
def mutation(agents):
            for agent in agents:
                if random.uniform(0.0, 1.0) <= 0.1:
                    weights = agent.neural_network.weights
                    shapes = [a.shape for a in weights]


flattened = np.concatenate([a.flatten() for a in weights])
                    randint = random.randint(0,len(flattened)-1)
                    flattened[randint] = np.random.randn()


newarray = [a ]
                    indeweights = 0
                    for shape in shapes:
                        size = np.product(shape)
                        newarray.append(flattened[indeweights :
indeweights + size].reshape(shape))
                        indeweights += size
                    agent.neural_network.weights = newarray
            return agents
```

Fig. 4.2 The process of crossover of agents at the crossover and mutation functions

This is the mutation function. The flattening is the same as the crossover function. Instead of splitting the points, a random point is chosen, to be replaced with a random value.

4.4 Convolutional neural network blocks implementation

First of all, while implementing the constructive block of an CNN it is necessary to define the set of function input parameters. These parameters are mostly the same for all the building blocks of a convolutional neural network.

The list of parameters is following:

- **initial_conv_filters**: number of features for the initial convolution;
- **depth**: number or layers in the each block, defined as a list.

  ResNet-50 = [3, 4, 6, 3]

  ResNet-101 = [3, 6, 23, 3]

  ResNet-152 = [3, 8, 36, 3];
- **filter**: number of filters per block, defined as a list;
- **width**: width multiplier for the network (for Wide ResNets);
- **bottleneck**: adds a bottleneck conv to reduce computation;
- **weight_decay**: weight decay (l2 norm);
- **include_top**: whether to include the fully-connected layer at the top of the network;
- **weights**: `None` (random initialization) or `imagenet` (trained on ImageNet);
- **input_tensor**: optional Keras tensor (i.e. output of `layers.Input()`) to use as image input for the model;
- **input_shape**: optional shape tuple, only to be specified if `include_top` is False (otherwise the input shape has to be `(224, 224, 3)` (with `tf` dim ordering) or `(3, 224, 224)` (with `th` dim ordering). It should have exactly 3 inputs channels, and width and height should be no smaller than

```
init = input_tensor
channel_axis = 1 if K.image_data_format() == "channels_first" else -1

x = BatchNormalization(axis=channel_axis)(input_tensor)
x = Activation('relu')(x)

if strides != (1, 1) or _tensor_shape(init)[channel_axis] != filters * k:
    init = Conv2D(filters * k, (1, 1), padding='same', kernel_initializer='he_normal',
                  use_bias=False, strides=strides)(x)

x = Conv2D(filters * k, (3, 3), padding='same', kernel_initializer='he_normal',
           use_bias=False, strides=strides)(x)
x = BatchNormalization(axis=channel_axis)(x)
x = Activation('relu')(x)

x = Conv2D(filters * k, (3, 3), padding='same', kernel_initializer='he_normal',
           use_bias=False)(x)

# squeeze and excite block
x = squeeze_excite_block(x)

m = add([x, init])
return m
```

Fig. 4.3 The function of the SE-ResNet block

On the example of the squeeze and excitation block based on residual module we can create relative function as it shown on figure 4.3. With the same approach we can modulate all the constructive blocks for our CNN implementation.

4.4 Creating the whole convolutional neural network implementation

A convolution duplicates a grid of pixels with a channel lattice or 'portion' and entireties up the increase esteems. At that point the convolution slides over to the following pixel and rehashes a similar procedure until all the picture pixels have been secured. [23]

Stacking the information dataset acknowledged as following:

from keras.datasets import mnist

#download mnist data and split into train and test sets

(X_train, y_train), (X_test, y_test) = mnist.load_data()

Presently we should investigate one of the pictures in our dataset to perceive what we are really going after. We will plot the principal picture in our dataset and check its size utilizing the 'shape' work.

```
import matplotlib.pyplot as plt
#plot the first image in the dataset
plt.imshow(X_train[0])
#check image shape
X_train[0].shape
```

As a matter of course, the state of each picture in the mnist dataset is 28 x 28, so we won't have to check the state of the considerable number of pictures. When utilizing genuine world datasets, you may not be so fortunate. 28 x 28 is likewise a genuinely little size, so the CNN will probably keep running over each picture before long.

Next, we have to reshape our dataset inputs (X_train and X_test) to the shape that our model expects when we train the model. The main number is the quantity of pictures (60,000 for X_train and 10,000 for X_test). At that point comes the state of each picture (28x28). The last number is 1, which implies that the pictures are greyscale.

```
#reshape data to fit model
X_train = X_train.reshape(60000,28,28,1)
X_test = X_test.reshape(10000,28,28,1)
```

We have to 'one-hot-encode' our objective variable. This implies a section will be made for each yield class and a double factor is inputted for every classification. For instance, we saw that the primary picture in the dataset is a 5. This implies the 6th number in our exhibit will have a 1 and the remainder of the cluster will be loaded up with 0.

```
from keras.utils import to_categorical
#one-hot encode target column
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)
```

y_train[0]

Now we're able to form the model. As an example it's will be CNN with two convolutional layers.

```
model.add(Conv2D(64, kernel_size=3, activation='relu',
input_shape=(28,28,1)))
model.add(Conv2D(32, kernel_size=3, activation='relu'))
model.add(Flatten())
model.add(Dense(10, activation='softmax'))
```

The model sort that we will utilize is Sequential. Successive is the most straightforward approach to assemble a model in Keras. It enables you to fabricate a model layer by layer. We utilize the 'include()' capacity to add layers to our model.

Our initial 2 layers are Conv2D layers. These are convolution layers that will manage our information pictures, which are viewed as 2-dimensional frameworks. 64 in the principal layer and 32 in the second layer are the quantity of hubs in each layer. This number can be changed in accordance with be higher or lower, contingent upon the extent of the dataset. For our situation, 64 and 32 function admirably, so we will stay with this until further notice.

Portion size is the extent of the channel network for our convolution. So a portion size of 3 implies we will have a 3x3 channel framework. Allude back to the presentation and the main picture for an update on this.

Enactment is the initiation work for the layer. The initiation work we will use for our initial 2 layers is the ReLU, or Rectified Linear Activation. This enactment capacity has been demonstrated to function admirably in neural systems.

Our first layer additionally takes in an info shape. This is the state of each information picture, 28,28,1 as observed prior on, with the 1 connoting that the pictures are greyscale.

In the middle of the Conv2D layers and the thick layer, there is a 'Level' layer. Smooth fills in as an association between the convolution and thick layers.

'Thick' is the layer type we will use in for our yield layer. Thick is a standard layer type that is utilized much of the time for neural systems.

We will have 10 hubs in our yield layer, one for every conceivable result (0–9).

The actuation is 'softmax'. Softmax causes the yield aggregate to up to 1 so the yield can be translated as probabilities. The model will at that point make its forecast dependent on which choice has the most elevated likelihood.

The streamlining agent controls the learning rate. We will utilize 'adam' as our optimizer. Adam is commonly a decent enhancer to use for some cases. The adam enhancer alters the learning rate all through preparing.

The learning rate decides how quick the ideal loads for the model are determined. A littler learning rate may prompt increasingly exact loads (up to a specific point), yet the time it takes to register the loads will be longer.

We will utilize 'categorical_crossentropy' for our misfortune work. This is the most well-known decision for characterization. A lower score demonstrates that the model is performing better. To make things much simpler to translate, we will utilize the 'exactness' metric to see the precision score on the approval set when we train the model.

#compile model using accuracy to measure model performance

model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

Presently we will prepare our model. To prepare, we will utilize the 'fit()' work on our model with the accompanying parameters: preparing information (train_X), target information (train_y), approval information, and the quantity of ages.

For our approval information, we will utilize the test set gave to us in our dataset, which we have part into x_test and y_test.

The quantity of ages is the occasions the model will push through the information. The more ages we run, the more the model will improve, up to a

specific point. After that point, the model will quit improving during every age. For our model, we will set the quantity of ages to 3.

CHAPTER 5

Occupation Safety

5.1 Analysis of harmful and dangerous production factors

The worker that performs with the convolutional neural network (CNN) can performs his responsibilities at following working place. The working place is the room of 1 – 12 working places. The room sizing ranges from six square meters and additionally 4.5 square meters per person. There at least one window for natural lighting and manual air conditioning, the ceiling lamps of day lighting for at least 200lx and quartz lamps for disinfection.

For an air conditioning there the common complex system with the regulatory air exchanges ratio (60 square meters per hour for one person at the working place), the functions of filtering the air flow from dust and large particles without external noises and controlling of air temperature. The standard temperature value kept by system is 20-25C that relay of list of factors such as humidity, air exchange ratio etc.

The working place supplied with the one of computer devices such as personal computer, mono-integrated computing system, laptop, etc. Also, there are two IPS computer monitors, printer, scanner and tabletop lamp. Besides of exact working place in the office room placed coffee machine, microwave and manual heater.

The list of dangerous factors which can impact on worker:

- Increased or decreased air temperature of the working area (without air conditioning or heat systems – 8 – 32C);
- Lack or absence of natural light;

- Insufficient lighting of the working area;
- High or low humidity;
- Increased or decreased air mobility (without air conditioning system is 55 square meters on whole working place);
- Sensitizing (Eyes influence, to make the rest for each 1.5 hours of active working with 10 minutes of the rest);
- Neuropsychiatric overload – monotony of work;
- Neuropsychiatric overload – emotional overload.

Therefore, the main dangerous factors are interconnected with the working process with the computers (computer monitor) and the specific of the tasks that cause its monotony and emotional overload. To prevent this it's necessary to responsively approach the work, regularly take breaks and perform special praxis.

5.2 Measures to reduce the impact of harmful and dangerous production factors

One of the main factors is the lack or absence of natural light and insufficient lighting of the working area. To prevent this it's necessary to check and calculate the lighting intensity in the room, what light sources should be applied and at what relative placing.

In our work process there no low-sized objects so it will be enough 150-200 lux of lighting intensity. Also it's necessary to prevent the lighting contrast between working area lighting and whole room (not more than 25%).

Lighting is calculated for a room with an area of 18 m^2, the width of which is 6m, and the height is 3m. The number of fixtures is determined by the luminous flux method. For this, the luminous flux F incident on the surface is determined:

$$F = \frac{E \cdot K \cdot S \cdot Z}{n} \qquad (5.1)$$

where F - calculated luminous flux lx, E - normalized minimum illumination, lx (the work of a programmer belongs to the category of precise work = 300lx), S - illuminated area, Z - ratio of average illumination to minimum, K - safety factor taking into account the decrease in the luminous flux of the lamp as a result of contamination of the lamps during operation, n - utilization rate.

Utilization coefficient taken from the table of lighting flow values.

$$I = \frac{S}{h \cdot (A + B)} = \frac{18}{2.92 \cdot (3 + 6)} = 0.6849 \ (n = 0.22)$$

$$F = \frac{300 \cdot 1.5 \cdot 18 \cdot 1.1}{0.22} = 40500 \ lx$$

For lighting, fluorescent lamps of the LB40-1 type are selected, the luminous flux of which = 4320 lx. The required number of lamps is calculated using the formula:

$$N = \frac{F}{F_n} = \frac{40500}{4320} = 9 \ units. \qquad (5.2)$$

To prevent listed before harmful factors to impact on the workers it's necessary to supply the working area with some preventive measures.

For the air conditioning system is the: 60 m^3 per person with filtering and temperature controls functionality.

For increased or decreased air temperature of the working area there is air temperature control at conditioning system (keeps on 20-25C), integrated heating system with heating battery and manual heating electric device.

For sensitizing harmful factor it is the regular breaks from using the computer monitor each one and half hour of its active usage.

To prevent dangerous effect from work specific such as monotony of work and emotional overload it's necessary to prevent the workers working overtime and normalizing the limits of expected results for its performance time.

5.3 Occupation safety instruction

To provide the safety condition at the working place everyone should follow such general requirements as acknowledgement about emergency action plan, maintain fire protection and prevention equipment, how to use it properly and where it placed in the office. The Occupational Safety and Health Administration require workers to have a fire safety emergency action plan in place that outlines the actions employees and employers must take in the event of a fire. Employers are responsible for keeping fire protection equipment in working order. This includes portable fire extinguishers and fire suppression systems, which have to be regularly inspected, maintained and tested. Portable fire extinguishers that have been used once need to be recharged. There required that employers provide employees with training on the proper use of fire safety equipment. Training sessions must be provided to employees at least once per year.

Before starting the work, worker should check the socket in which computer and other devices are plugged-in. Check the startup process and it's stability.

During the work be sure to not affect the wires and computer sockets, keep the unbounded liquid sources out of working place. During the working process every one and half hour worker should take a ten minutes break from the computer device to provide his eyes safety.

After the work all the electrical devices should be taken off, the computer should be normally taken off without manually cutting the power source.

CHAPTER 6

Environmental Protection

6.1 Convolutional neural networks for environmental monitoring

Image classification plays an essential role in several research areas and application domains. Land-use recognition is one important research field, where satellite and radar images are usually employed for further identification of deforesting areas, farms, and roads, among others. In the past years, drones have also been used for monitoring purposes either. New onboard cameras with high spatial resolution and infra-red devices are now in lockstep with lightweight and more autonomous drones that are currently used for both surveillance and personal matters. Drone can also cover wide areas and monitor deforesting areas, wildfire, and illegal land-use. Kim et al. proposed to use Convolutional Neural Networks (CNN) to classify doppler-acquired images using drones. The work merged both time- and frequency-domain images to classify indoor and outdoor images with recognition rates nearly to 95%. also used drone images for the classification of land-cover classes using Random Forests. The proposed approach was compared against Support Vector Machines and Decision Trees and achieved an overall accuracy of 91.23%. Ventura et al. employed a low-cost drone to monitor marine areas and further identify coastal fish nursery grounds.

The authors argued that watching coastal areas with some regularity is costly and sometimes unfeasible. In this work, they make use of a small drone that can produce fine-scale images for further image classification. Han et al. [6] focused on the analysis the error range of drone-based image classification when compared with field measurements (i.e., ground-truth data). The authors noticed that one

could obtain pretty much precise results when we set the drones accordingly to the primary goal of the work.

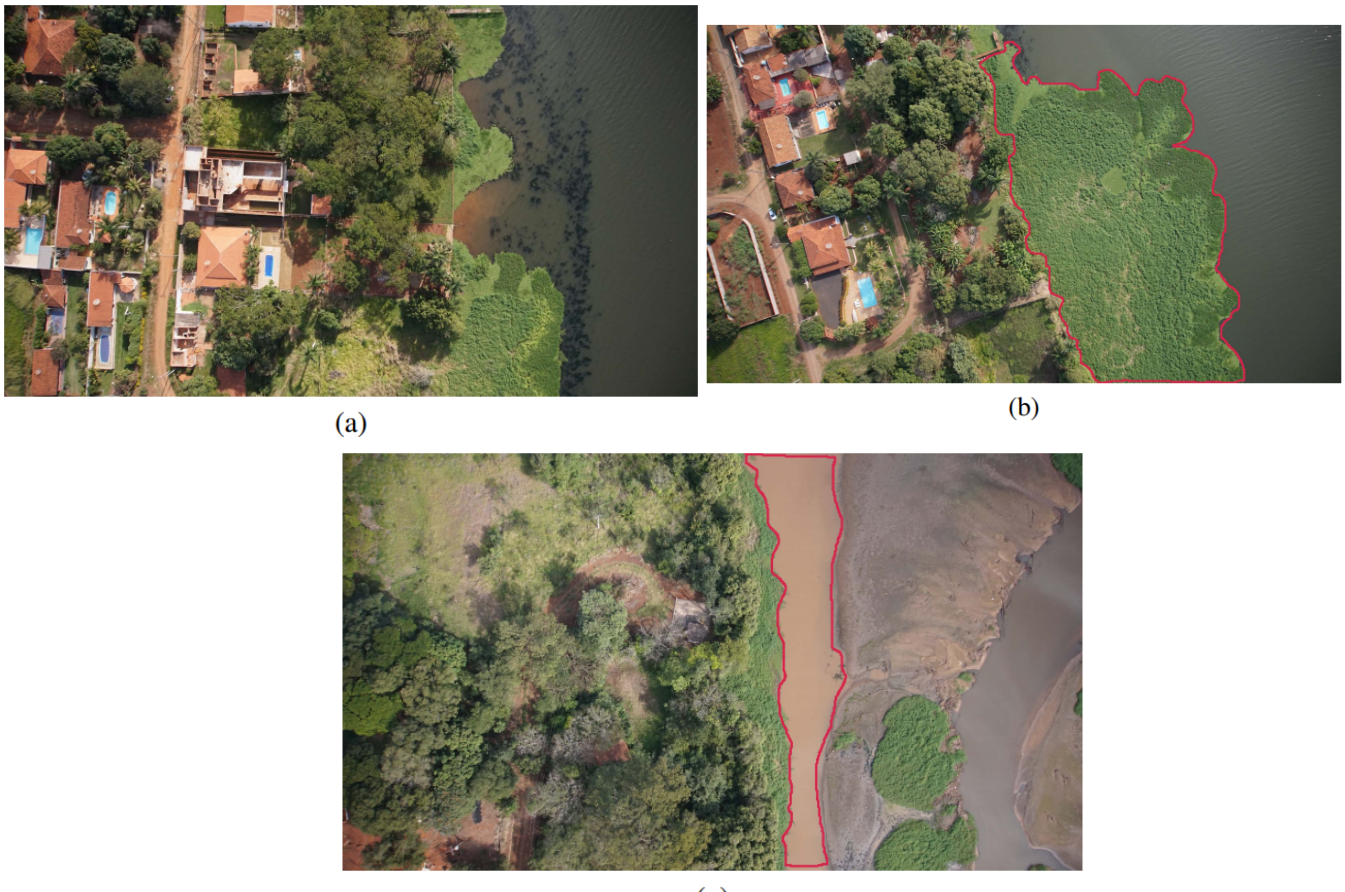
Rewick et al. also employed drones for pollution monitoring in urban areas. The work aimed at combining three dimensional surface terrain models with pollution data acquired by the drone to reconstruct a three-dimensional model of potential pollution sources. Besides very recently, Chen et al. presented an approach to detect illegal drones in the wild using deep learning. In this case, the authors state that the main problem concerns the lack of data for training purposes. Such drawback was addressed using data augmentation with synthetic images and video sequences extracted from the Internet mainly.

Therefore, as one can observe, most of the works surveyed in this paper are pretty much recent, thus highlighting the importance of this source of information for monitoring and controlling activities using drones. However, to the best of our knowledge, we have not observed any work that attempted at using drones and deep learning to monitor illegal land-use for environmental controlling purposes, which turns out to be the main contribution of this work. Particularly, we are interested in using drones to identify illicit activities (e.g., non-authorized buildings, crop plantation, and fishing, among others) in the neighborhood of a private area that it is maintained by a Brazilian energy company. With such an automated system, the company can better control the territory, thus avoiding damages to the natural resources.

Everyone are interested in using a drone to monitor the area surrounding a power plant from a Brazilian energy company. Recent protocols regulate the rights and obligations concerning the natural resources in the area the energy company is required to maintain. In this context, tracking illegal activities, such as new buildings (without permission), deforesting areas, wildfire, and unregulated fishing is of crucial importance.

It is first considered four main classes of interest:

• buildings: it comprises houses and their nearby areas, such as swimming pool and parking lot;



(a) (b)

(c)

Fig. 6.1.1 Some samples from the dataset: (a) “good quality” image, (b) presence of algae along the riverside, and (c) mudlike river. The images were obtained from a city in the countryside of Sao Paulo State, Brazil.

- deforesting areas: it comprises grazing areas and unpaved roads;
- water: it comprises rivers, lakes, and lagoons.
- forest: it comprises mainly large trees and bushes.

However, classifying drone images in those labels in the wild is quite more complicated that one can expect. Some shortcomings may be noticed, such as shadows, swimming pools (they must be considered from the “building” class, not from the water), and the rivers and small lagoons mainly. Depending on the season, the water may have its color changed due to the presence of algae. Also, some species usually get stuck nearby the riverside and are likely to be mistaken for grass or land.

Figure 6.1.1 displays three sample images from the ones captured using the drone. Figure 6.1.1a depicts an image that is considered of “good quality”, i.e., with minor or no problems that may require some preprocessing (i.e., shadows),

and Figure 6.1.1b depicts an image with the presence of algae along the riverside (highlighted area). Finally, Figure 6.1.1c displays a mud-like river (highlighted area) that might be easily mistaken for land (i.e., “deforesting area” class). Another problem we faced and was also pointed out in the review of the literature concerns the lack of labeled data for training the classifiers. Therefore, we manually created and labeled a dataset with 100 samples (patches of different sizes), being 25 samples for each aforementioned class. When extracting the patches, we carefully considered maximizing the inter-class variability, as well as also presenting challenging situations for training the model. Figure 2 presents some patches extracted from the original images.

As we are dealing with colored images, we opted to use an ImageNet pre-trained convolutional neural network under the Keras1 environment. To establish the desired goal, we have chosen two out of several consolidated models, i.e., VGG16 and VGG19. Figure 3 displays the VGG16 architecture, while

VGG19 can be seen as an evolved VGG16 version, having three more convolutional layers, being each one added at every max pooling step, beginning right after the second max pooling layer.

Additionally, after the last pooling layer, we discarded all fully connected and softmax layers to fine-tune a new classification model based on ImageNet already-trained weights.

We used the previous layers to predict the so-called “bottleneck features” from our 96 training samples (i.e., we used 96 images for training and 4 for validating purposes), which were now resized to 150 × 150 patches2. The new classification model stands for a 256-output ReLU-activated fully connected layer followed by a 4-output fully connected layer, and finally a softmax classifier on top of that. Note that these new layers were fine-tuned for 50 epochs using a default parameter rms-prop optimizer and a cross-entropy loss function under a validation set with 4 images (i.e., one per class).

Further, the aforementioned trained model was used to predict the unseen images captured by the drones, which where spliced into 150×150 patches for

classification purposes (i.e., the same size used during training). During training, both neural models achieved a similar performance, with around 95% of recognition rates.

6.2 A recent ecological application of convolutional neural networks

In recent years, a wealth of material has been accumulated on the influence of unfavorable environmental factors on public health, new scientific results have been obtained on the relationship between environmental factors and public health. However, the accumulation of information does not lead to new knowledge. The rate of evolution of information technology is significantly ahead of the evolution of the methodological foundations of ecology, protection and management of the state of the environment - the existing methods and standards are based on outdated traditional concepts and concepts, the technologies of preparation and decision-making lag behind the achievements in mathematical modeling, information and computing technologies.

In addition, a feature of the development of the technosphere in recent years is a change in its systemic properties: the emergence of risks caused by long causal relationships, their interdisciplinary nature, global changes of a technogenic nature, high sensitivity to "weak influences", etc. This inevitably leads to the need the use of a systematic approach to the analysis of technogenic risks and technosphere safety in general using the methods of system analysis and information technology.

Over the past few years, there has been an active development of artificial intelligence technologies that imitate the activity of brain neurons - artificial neural networks (ANN). The use of artificial neural networks provides

a number of advantages over the traditional approach, allowing simultaneously taking into account a large number of influencing parameters affecting a set of dependent quantities, and automatically synthesizing highly complex analytical models from available databases, which most fully reflect the causal relationships between the parameters characteristic of the system under study, to automatically assess the degree of influence of each of the influencing parameters on the dependent values and to correct the obtained analytical model with the appearance of new data by "learning" the neural network.

An artificial neural network is a mathematical apparatus that allows you to build algorithms for information processing, which has a unique ability to learn by examples and "recognize" in the stream of "noisy" and often contradictory information, signs of previously encountered images and situations. ANNs allow finding hidden dependencies between input and output data, which are beyond the attention of classical methods.

Simulation of systems using ANN is carried out in three stages: training, assessment of learning outcomes and the use of trained networks for forecasting. On the training set, the network is configured, i.e. correction of neuron weights in proportion to the output error. The control set data is used for cross-checking - at each step of training the network, the error for the entire set of observations from the control set is calculated and compared with the error on the training set. The ANN learning algorithm is aimed at minimizing the error at the output of the network, which is estimated using statistical indicators (such as the mean absolute error). The network with the least error is recognized as the most efficient.

To test the acceptability of neural network technologies for assessing environmental risk, a model of the impact of harmful emissions on the health of the Krasnoyarsk population was built. As indicators of the state of the environment, we used the concentration of pollutants in the air according to the data of the Center for Environmental Pollution Monitoring of the State Institution “Krasnoyarsk TsGSM-R”, which monitors the air quality at 8

stationary posts in Krasnoyarsk. The yearbooks “The state of atmospheric air pollution in cities on the territory of the Krasnoyarsk Territory, the Republics of Khakassia and Tyva” were processed from 1999 to 2010.

The data of sanitary and demographic statistics of the Territorial Body of the Federal State Statistics Service for the Krasnoyarsk Territory were used as an indicator of population health.

By varying the structure and parameters of the neural network and the learning algorithm, several network models were obtained, from which the best one in terms of generalization property (the smallest error on the test sample) was selected. The selected network has a layer-by-layer organization and direct signal propagation (multilayer perceptron) with three layers: 1 layer - 3 neurons, 2 and 3 layers - 2 neurons each with a sigmoidal activation function of neurons. The average relative error for all results was 0.40%, the average absolute error was 0.93%, i.e. the network provides good convergence of calculated and actual values.

CONCLUSIONS

1. In this work were analyzed the approaches into image recognition and classification. Their features and practical implementations.
2. The fundamental complex comparison and review on modern convolutional neural network was done. The constructive blocks of different CNNs and their practical implementation were shown.
3. In this work were done the analysis of modern literature and scientific papers about the structural parametric synthesis of convolutional neural networks and the main algorithms was fetched out.
4. The types and implementation of genetic algorithms were discovered and formed the strict algorithm for its implementation for generation the structure of convolutional neural network and its global and local parameters.
5. The idea of combining the genetic algorithms in tandem of different constructive blocks that mostly all from the modern convolutional neural networks was developed to generate the unique CNN architecture that will suit any highly specialized task of image recognition.
6. Proposed complex system for structural parametric synthesis of CNN that was formed using the newly proposed and modified genetic algorithm in the para of data preparation algorithm. In result the high-accurate image recognition system was received.

REFERENCE

1. *T. Kawaguchi, D. Hidaka & M. Rizon,"Robust Extraction of Eyes From Face", IEEE on Pattern Recognition 15th International Conference, 2000, pp.1109-1114.*
2. *Gaurav Mittal ,SreelaSasi ," Robust Preprocessing Algorithm for Face Recognition"Proceedings of the 3rd Canadian conference on Computer and Robot vision, United States ofAmerica , 2006.*
3. *P.S.Hiremath, S. Shivashankar, and JagadeeshPujari "Wavelet based features for color texture classification with application to cbir", IJCSNS International Journal of Computer Science and Network Security, VOL.6 No.9A, September 2006.*
4. *Lin Sadrina .W Gao Yang, Liu Ray K.J "Template matching for image prediction: A game-theoretical approach", IEEE ICASSP, 2012.*
5. *Xia Xing Su , Yuan GuoTui , Hua Chen Tian " Study an optimal wavelength decomposition level in infrared and visual light imagefusion" IEEE, International conference on measuring technologies and electronics automation,2010.*
6. *Smeets Dirk, Claes Peter, HermansJeroen , Vandermeulen Dirk, Suetens Paul, "A comparative Study of 3-D Face recognition under expression variations", IEEE transactions on system , man, andCybernetics,vol-42,no-5,2012.*
7. *SakaliMustafa, Lam kin-man , Yan Hong "A faster converging snake algorithm to locate object boundaries, IEEEtransactionson imageprocessing,vol-15,no-5,2006.*
8. *Aleix M.Matinez and Avinash C. Kak," PCA versus LDA"IEEE Transactions on Pattern Analysis and Machine Intelligence,Vol.23,No. 2, February 2001.*
9. *M. Turk and A. Pentland, "Face Recognition Using Eigenfaces," Proc. IEEE Conf on Computer Vision and Pattern Recognition , 1991, pp. 586 – 591.*
10. *Juwei Lu,Kostantinos N. Plataniotis and Anastasios N. Venet sanopoulos, "Face Recognition Using LDA- Based Algorithms" IEEE Transactions in Neural*

Network,Vol.14 No.1,January 2003.

11. P.N.Belhumeur ,J.P.Hespanha and D.J. Kriegman,"Eigenfaces Vs Fisherfaces : Recognition using Class specic linear projection" IEEE Trans. Pattern Anal . Machine Intell, Vol 19, pp 711-720,July1997.

12. F. Chollet. Keras. https://github.com/fchollet/keras, 2015.

13. S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In Proceedings of The 32nd International Conference on Machine Learning, pages 448–456, 2015.

14. J. Jin, A. Dundar, and E. Culurciello. Flattened convolutional neural networks for feedforward acceleration. arXiv preprint arXiv:1412.5474, 2014.

15. A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In Advances in neural information processing systems, pages 1097–1105, 2012.

16. M. Lin, Q. Chen, and S. Yan. Network in network. arXiv preprint arXiv:1312.4400, 2013.

17. F. Mamalet and C. Garcia. Simplifying ConvNets for Fast Learning. In International Conference on Artificial Neural Networks (ICANN 2012), pages 58–65. Springer, 2012.

18. O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, et al. Imagenet large scale visual recognition challenge. 2014.

19. L. Sifre and S. Mallat. Rotation, scaling and deformation invariant scattering for texture discrimination. In 2013 IEEE Conference on Computer Vision and Pattern Recognition, Portland, OR, USA, June 23-28, 2013, pages 1233–1240, 2013.

20. C. Szegedy,W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 1–9, 2015.

21. C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the

inception architecture for computer vision. arXiv preprint arXiv:1512.00567, 2015.

22. *T. Tieleman and G. Hinton. Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning, 4, 2012. Accessed: 2015- 11-05.*

23. *M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Man´e, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Vi´egas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. Tensor- Flow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.*