Харківський національний університет радіоелектроніки

(повне найменування вищого навчального закладу)

Кафедра штучного інтелекту

(повна назва кафедри,)

**КУРСОВА РОБОТА**

з \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_дисципліни «Машинне навчання»\_\_\_\_\_\_\_\_\_\_\_\_\_

(назва дисципліни)

на тему: «Image Classification with Tensorflow»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Студента () \_3\_курсу 17-1 групи

напряму підготовки\_\_\_\_\_ІТШІ\_\_\_\_\_\_\_\_

\_\_\_\_\_\_Гури А. О. \_\_\_

(прізвище та ініціали)

Керівник \_доцент каф. ШІ, доц., к.т.н.\_\_\_ \_\_\_\_\_\_\_\_\_Вітько О. В.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Національна шкала \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Кількість балів: \_\_\_\_\_\_\_\_\_\_Оцінка: ECTS \_\_\_\_\_

Члени комісії \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_ Вітько О. В.\_\_\_\_

(підпис) (прізвище та ініціали)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_ Кулішова Н.Є.\_\_\_

(підпис) (прізвище та ініціали)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_Філатов В.О.\_\_\_

(підпис) (прізвище та ініціали

м. Харків – 2019 рік

\_\_\_\_\_\_\_\_\_\_\_\_Харківський національний університет радiоелектронiки\_\_\_\_

# Інститут, факультет, відділення\_\_\_\_\_\_\_КН\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# Кафедра, циклова комісія\_\_\_\_\_ШІ\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Освітньо кваліфікаційний рівень\_\_\_\_бакалавр\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Напрям підготовки\_\_\_КН  «Комп’ютерні науки»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_

# 

**ЗАТВЕРДЖУЮ**

**Завідувач кафедри** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

“\_\_\_\_” \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2019 року

## **З А В Д А Н Н Я**

## **НА НА КУРСОВУ РОБОТУ з дисципліни «Машинне навчання»**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Гурі Андрію Олеговичу \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(прізвище, ім’я, по батькові)

1. Тема роботи «Image Classification with Tensorflow»\_\_\_\_\_\_\_\_\_\_\_\_\_\_

керівник роботи\_Вітько Олександра Валеріївна, к.т.н, доц.\_\_\_\_\_\_\_

2.Строк подання студентом проекту 26.12.2019\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

3. Вихідні дані до проекту:\_\_Image Processing, Tensorflow, Tensorflow Documentation, Image Classification, Keras on the top of Tensorflow, Image Classification with Tensorflow, Deep Learning, Neural Networks, Convolutional Neural Networks наукові статті, мова програмування Python, Jupyter Notebook, Kaggle Kernel, Kaggle Datasets, Youtube Lectures, Emotions Kaggle Dataset, Stanford Dog Breeds Kaggle Dataset, Intel Image Classification Dataset, Olivetti Faces

4. Зміст розрахунково-пояснювальної записки (перелік питань, які підлягають розробці): Вступ, аналіз предметної галузі та постановка задачі, теоретичні дослідження, експериментальні дослідження, висновки та аналіз результатів.

5. Дата видачі завдання \_24.10.219\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

КАЛЕНДАРНИЙ ПЛАН

|  |  |  |  |
| --- | --- | --- | --- |
| № | Назва етапів роботи | Термін  виконання етапів роботи | Примітка |
| 1 | Видача завдання | 24.10.2019 | виконано |
| 2 | Аналіз літератури та інтернет-джерел | 11.11.2019 - 18.11.2019 | виконано |
| 3 | Ознайомлення із Image Processing та Image Classification | 18.11.2019 - 21.11.2019 | виконано |
| 4 | Вивчення можливостей Tensorflow | 21.11.2019 - 25.11.2019 | виконано |
| 5 | Проведення теоретичних досліджень | 25.11.2019 - 02.12.2019 | виконано |
| 6 | Розробка програми та вивчення функціональних можливостей Tensorflow в контексті Image Classification | 02.12.2019 - 09.12.2019 | виконано |
| 7 | Проведення експериментів | 09.12.2019 - 15.12.2019 | виконано |
| 8 | Оформлення пояснювальної записки | 22.11-23.11, 04.12-05.12, 18.12-24.12. | виконано |
| 9 | Підготовка презентації | 25.12.2019 | виконано |
| 10 | Захист КР | 27.12.2019 |  |

**Студент \_\_\_\_\_\_\_\_\_\_**  Гури А. О.\_\_\_\_\_

(підпи )                  (прізвище та ініціали)

**Керівник роботи   \_\_\_\_\_\_\_\_\_\_ \_\_\_\_**Вітько О. В.\_\_\_\_\_

(підпис)               (прізвище та ініціали)

ABSTRACT

Explanatory note to the interdisciplinary course project: p. 52, fig. 71, app. 1, ref. 5

The topic of coursework – Image Classification with Tensorflow

The main goal of this coursework is to investigate the machine learning and deep learning tool(library) Tensorflow respectively to the image classification problem and practically apply tools provided with Tensorflow for classification real datasets.

The object of study is a library called Tensorflow, its tools provided for image classification, and finally real world practical application of these tools for image datasets.

The results of investigation are analyses of classification applied to several datasets using Tensorflow tools for image classification, Convolutional Neural Networks in particular.

IMAGE PROCESSING, IMAGE CLASSIFICATION, TENSORFLOW, TF, KERAS ON THE TOP OF TENSORFLOW, DATASET, MACHINE LEARNING, ML, ALGORITHM, CLASSIFICATION, CNN, CONVOLUTIONAL NEURAL NETWORKS, DEEP LEARNING, PYTHON, KAGGLE DATASET, FACE RECOGNITION, EMOTION RECOGNITION, SCIKIT LEARN

CONTENTS

Introduction...............................................................................................................6

1 Domain analysis......................................................................................................8

1.1 Classification as one of the fields of machine learning…….......................8

* 1. Start of Image classification in context of Computer Vision......................9

1.3 Start of Convolutional Neural Networks.................................................10

1.4 Formulation of the problem………………………………………..........12

2 Theoretical knowledge………………………………………………................. 13

2.1 Image Processing Introduction…………………….................................13

2.2 Types of images …………………………..............................................13

2.3 Convolutional Neural Networks applied to Image Classification ………16

3 Experimental studies…………………………………………………………….32

3.1 Introduction to practical part …………..…………………………….....32

3.2 Working on first dataset………………………….……………………..32

3.3 Second Dataset……………………..…………………………………...37

3.4 Third Dataset……………………..……………………...……………...41

Conclusion………………………………………………………………………...47

INTRODUCTION

One of the most important areas of machine learning is the classification. Classification is a supervised learning approach in which the computer program learns from the data input given to it and then uses this learning to classify new observation.

Classification is probably the most important task of Machine Learning. There are enormous amount of different algorithms implemented in order to classify demanded dataset as good as possible. Also there huge amount of areas and cases, where we can apply this powerful machine learning technique. And one of them is image classification.

Image classification is probably one of the most important abilities of each person. We are being taught of classification since our birth even without knowing it. Remember you parents talking to you about some basic concepts, like dogs or cats, or maybe moments when you saw some strange thing flying in the sky, and your dad said it was a helicopter. All those moments were also image classification learning but applied for people.

And main goal of Artificial Intelligence is simulate or pretend to be as clever and teachable as real person. That is why it was only time question, when people decided to try to learn to implement so called «Computer Vision», that is nothing else than an attempt to learn machine(computer) to see world the way people do. And I can surely state that image classification is a fundamental concept, on which based all the Computer Vision field.

Just imagine how great is this field of study! We all can assume that we are good at guessing some sort of things by its images, for example telling the breed of a dog. But it is normal for the people to make mistakes. And here comes image classification as a part of Machine Learning. It basically states that it can teach a model to predict what is displayed on some image with minimum error chance. What I want to say, is that we can learn a model to predict something with image more accurately than a person does. And not just a bit more accurately. It is possible to increase accuracy up to 100 percent.

Now you can probably imagine what a great technology it is! But if you still not impressed, think about the application of this technique to for example, medical area. Let’s say predicting pneumonia or cancer according to some x-rays. It is obvious that even the best doctor can sometimes do a mistake, but a really well-trained model can do the same task almost without mistakes. Think about it and how great it is!

That is why I am extremely eager to study this field of Machine learning. And for this purpose I am going to do some theoretical, and practical research. Mostly practical.

1. DOMAIN ANALYSIS

1.1 Classification as one of the fields of machine learning

In machine learning and statistics, classification is the problem of identifying to which of a set of categories (sub-populations) a new observation belongs, on the basis of a training set of data containing observations (or instances) whose category membership is known.

In the terminology of machine learning, classification is considered an instance of supervised learning, i.e., learning where a training set of correctly identified observations is available. The corresponding unsupervised procedure is known as clustering, and involves grouping data into categories based on some measure of inherent similarity or distance.

Let’s say, you live in a gated housing society and your society has separate dustbins for different types of waste: one for paper waste, one for plastic waste, and so on. What you are basically doing over here is classifying the waste into different categories. So, classification is the process of assigning a ‘class label’ to a particular item. In the above example, we are assigning the labels ‘paper’, ‘metal’, ‘plastic’, and so on to different types of waste.

Figure 1.1 – Example of classification.

There are some types of classification algorithms in Machine Learning:

a) Linear Classifiers: Logistic Regression, Naive Bayes Classifier

b) Nearest Neighbor;

c) Support Vector Machines;

1. Decision Trees;
2. Boosted Trees;
3. Random Forest;
4. Neural Networks.

As we will see later, our main classifier will be related to one of the Neural networks types.

* 1. Start of Image classification in context of Computer Vision

Although Computer Vision (CV) has only exploded recently (the breakthrough moment happened in 2012 when AlexNet won ImageNet), it certainly isn’t a new scientific field.

Computer scientists around the world have been trying to find ways to make machines extract meaning from visual data for about 60 years now, and the history of Computer Vision, which most people don’t know much about, is deeply fascinating.

One of the most influential papers in Computer Vision was published by two neurophysiologists — David Hubel and Torsten Wiesel — in 1959. Their publication, entitled «Receptive fields of single neurons in the cat’s striate cortex, described core response properties of visual cortical neurons as well how a cat’s visual experience shapes its cortical architecture.

The duo ran some pretty elaborate experiments. They placed electrodes into the primary visual cortex area of an anesthetized cat’s brain and observed, or at least tried to, the neuronal activity in that region while showing the animal various images. Their first efforts were fruitless; they couldn’t get the nerve cells to respond to anything.

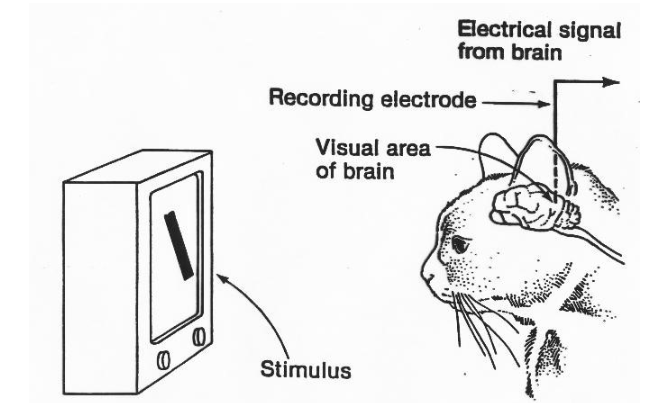
However, a few months into the research, they noticed, rather accidentally, that one neuron fired as they were slipping a new slide into the projector. This was one lucky accident! After some initial confusion, Hubel and Wiesel realized that what got the neuron excited was the movement of the line created by the shadow of the sharp edge of the glass slide.

Figure 1.2 – Experiment that made CV possible

The researchers established, through their experimentation, that there are simple and complex neurons in the primary visual cortex and that visual processing always starts with simple structures such as oriented edges.

Sounds familiar? Well, yeah, this is essentially the core principle behind deep learning.

1.3 Start of Convolutional Neural Networks

It is the year 1994, and this is one of the very first convolutional neural networks, and what propelled the field of Deep Learning. This pioneering work by Yann LeCun was named LeNet5 after many previous successful iterations since they year 1988!

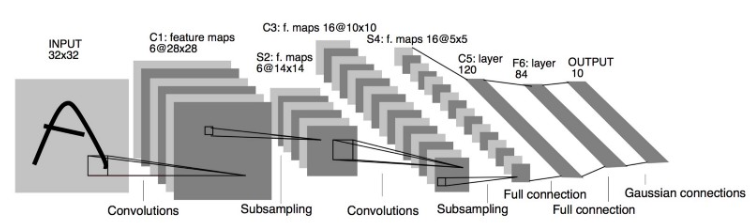


Figure 1.3 – LeNet5 Architecture

The LeNet5 architecture was fundamental, in particular the insight that image features are distributed across the entire image, and convolutions with learnable parameters are an effective way to extract similar features at multiple location with few parameters. At the time there was no GPU to help training, and even CPUs were slow.

Therefore being able to save parameters and computation was a key advantage. This is in contrast to using each pixel as a separate input of a large multi-layer neural network. LeNet5 explained that those should not be used in the first layer, because images are highly spatially correlated, and using individual pixel of the image as separate input features would not take advantage of these correlations.

LeNet5 features can be summarized as:

* convolutional neural network use sequence of 3 layers: convolution, pooling, non-linearity. This may be the key feature of Deep Learning for images since this paper!
* use convolution to extract spatial features
* subsample using spatial average of maps
* non-linearity in the form of tanh or sigmoids
* multi-layer neural network (MLP) as final classifier
* sparse connection matrix between layers to avoid large computational cost

In overall this network was the origin of much of the recent architectures, and a true inspiration for many people in the field

1.4 Formulation of the problem

The main goal of this coursework is to investigate the machine learning and deep learning tool(library) Tensorflow respectively to the image classification problem and practically apply tools provided with Tensorflow for classification real datasets.

In order to gain that goal it is important: to research some literature, find out what algorithms are used for image classification, take a closer look at these algorithms, try to use it practically with some real datasets.

Also it is important to find out pros and cons of these algorithms, do some practical applications. For this reason it is also important to choose some good datasets, process them correctly, and apply Tensorflow tools.

And finally it is important to make some assumptions based on what I have got from practical applications of Tensorflow.

2 THEORETICAL STUDIES

2.1 Image Processing Introduction

Digital Image Processing means processing digital image by means of a digital computer. We can also say that it is a use of computer algorithms, in order to get enhanced image either to extract some useful information.

Image processing mainly include the following steps:

1. Importing the image via image acquisition tools;

2. Analyzing and manipulating the image;

3. Output in which result can be altered image or a report which is based on analyzing that image.

So basically, what is an image? An image is defined as a two-dimensional function –F(x,y), where x and y are spatial coordinates, and the amplitude of F at any pair of coordinates (x,y) is called the intensity of that image at that point.

When x,y, and amplitude values of F are finite, we call it a digital image. In other words, an image can be defined by a two-dimensional array specifically arranged in rows and columns.

Digital Image is composed of a finite number of elements, each of which elements have a particular value at a particular location. These elements are referred to as picture elements, image elements, and pixels. A Pixel is most widely used to denote the elements of a Digital Image.

2.2 Types of images

So far as I have researched, there are pretty large amount of different types of images, in comparison to what I have thought. Here are some of them:

* Binary Images – The binary image as its name suggests, contain only two pixel elements i.e 0 & 1,where 0 refers to black and 1 refers to white. This image is also known as Monochrome.
* Black and white images – The image which consist of only black and white color is called Black and white images. According to what I have found out, they are quite often mixed up with grey images, which are said to be subtype of next category.
* 8 bit Color Format – It is the most famous image format. It has 256 different shades of colors in it and commonly known as Grayscale Image. In this format, 0 stands for Black, and 255 stands for white, and 127 stands for grey.
* 16 bit color – It is a color image format. It has 65,536 different colors in it. It is also known as High Color Format. In this format the distribution of color is not as same as Grayscale image. A 16 bit format is actually divided into three further formats which are Red, Green and Blue. That famous RGB format.

Here you can see the last two image types, how they look like, and how they are interpreted into matrixes, so that we can work with them.

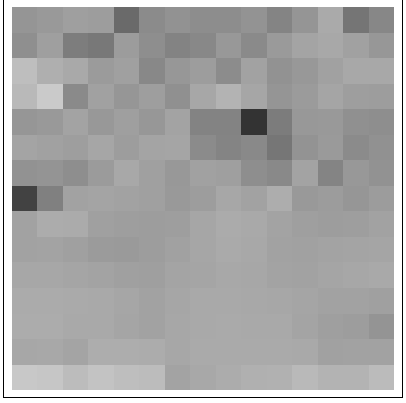


Figure 2.1 – Example of a greyscale image

Here you can see the way we usually see an image. And now how computer sees greyscale images:

Figure 2.2 – How computer sees greyscale images

As you can see and assume this is just a bunch of numbers representing the intensity of the gray color in each pixel. The intensities vary from 0 to 255 – so there are 256 possible values for each pixel to take.

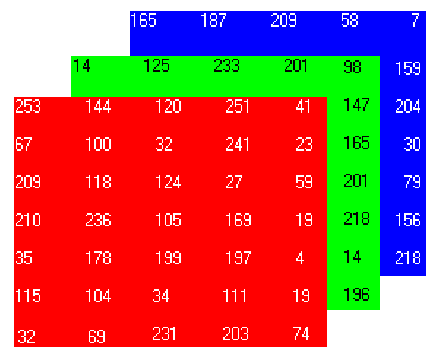
Now let’s take a look at colored image, or, talking more accurately, at RGB image:

Figure 2.3 – Example of RGB Image and how computer sees it.

Colored images are similar, except they are represented by three of those pixel intensity matrices, called channels. When an image is in RGB color system, each of those matrices represent the intensities of Red, Green and Blue colors respectively, as can be seen on the image above.

Here, too, the intensities can vary from 0 to 255 in each channel, giving us the overwhelming number of 256x256x256 = 16777216 different colors each pixel can be.

* 1. Convolutional Neural Networks applied to Image Classification

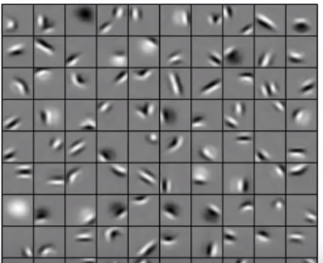
Convolutional Neural Networks can do some pretty cool things. For example, if you feed them some face images, they will learn some basic things, like edges and dots, light spots and dark spots.

Figure 2.4 – Basic concepts learned by convolutional neural network on faces dataset on first layer.

And as CNN are said to be multilayer neural networks that would be what will learn on the first layer. After the second layer they will learn some more recognizable concepts, like eyes, noses, mouse and so on.



Figure 2.5 – Patterns learned by CNN on the second layer.

 And finally, on the third layer things will look like a real faces:

Figure 2.6 – Patterns learned by CNN on the third layer.

Similarly, if you will feed them(CNNs) with a bunch of images of cars, on the lowest layer you will again get things like edges, but on further layers you will got things like tires, windows, doors, and so on. And on the last level you will get things that are clearly identify cars.

CNNs can even learn how to play computer games, by forming a patterns of pixels as they appear on the screen, and learning what is the best step/action to take.

Usually when people talk about CNNs, they do it the same way, as the would do if talking about magic. But obviously, they are not magic. What they do is based on some pretty simple ideas applied in a clever way.

To illustrate this, let’s take a look at pretty simple toy convolutional neural network as an example.

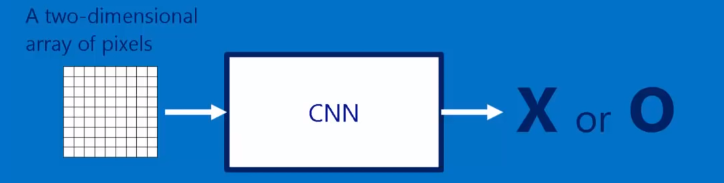
What this one does, it takes as input two dimensional array of pixels(you can think of it as a checker board), and each square on this checker board is either light or dark. And then after looking at it CNN decides whether it is a picture of X, or a picture of O.

Figure 2.7 – Example Neural Network logic.

Now, let’s take a look at two examples:

Figure 2.8 – First example of classification with X letter

 So we can see a picture of X letter, drawn with white squares on a background of black squares. The desired output for this test instance would be an X letter.

Figure 2.9 – Second example of classification with O letter

And respectively, on the second image, we want and suppose that our CNN will identify it as an O letter.

Actually this examples are seem to be quite obvious, so let’s take a look at more tricky ones:

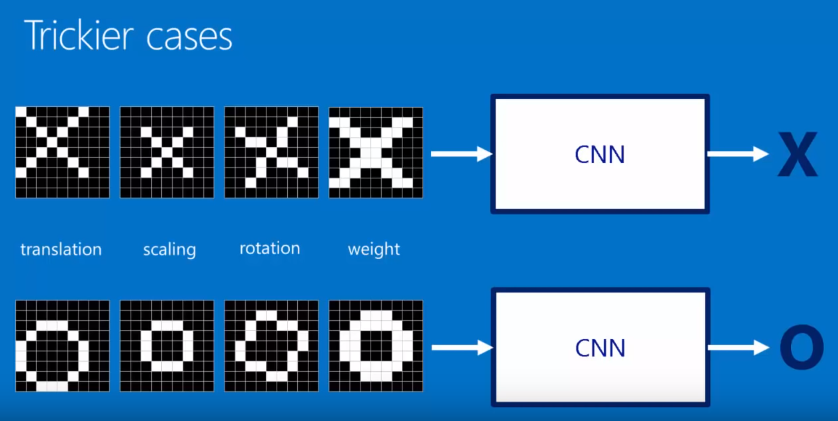


Figure 2.10 – More tricky examples

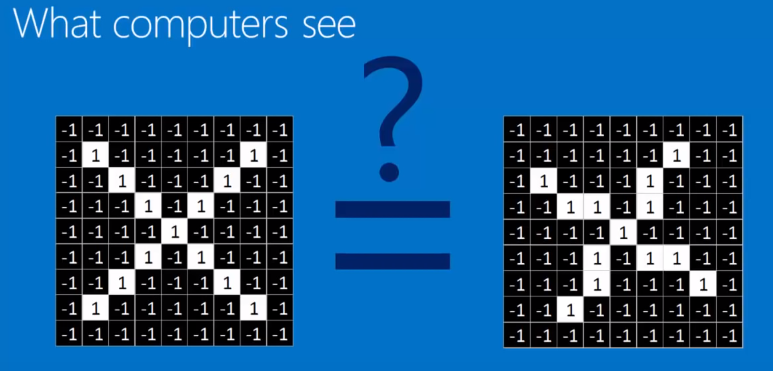
As you can notice, we add some changes and shifts to our input images, but we want to have still correct output. Now it becomes quite hard for computer to classify them, as basically it sees image in such way:

Figure 2.11 – The way computer sees two examples.

The only way computer can classify this image without using neural networks is just by comparing boxes with the same indexes.

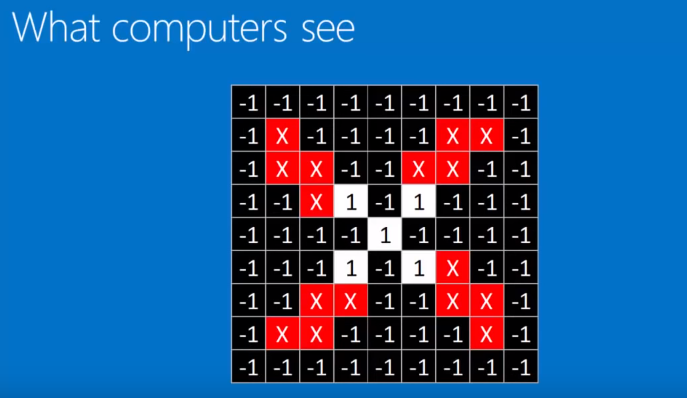


Figure 2.12 – Straightforward comparing

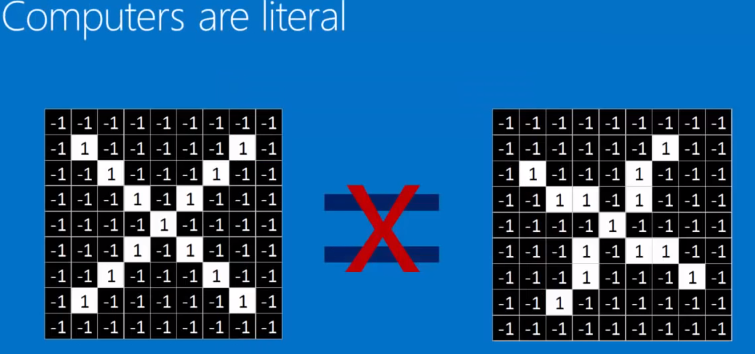


Figure 2.13 – Obvious result.

In such a way we will get a negative answer for both O and X letters.

And here comes CNN. The trick it uses to make predictions, is that it compares parts of the images, rather than a whole thing.

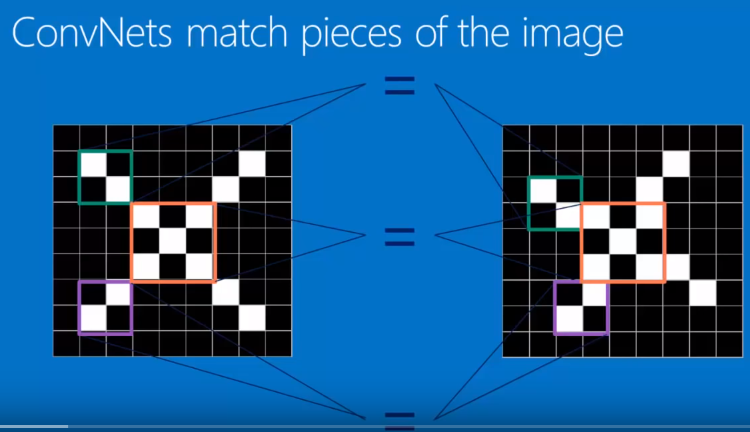


Figure 2.14 – How CNN compares images.

In that way images are broken down in smaller parts, or so called features. And then it becomes much more clear if previous two images are similar.

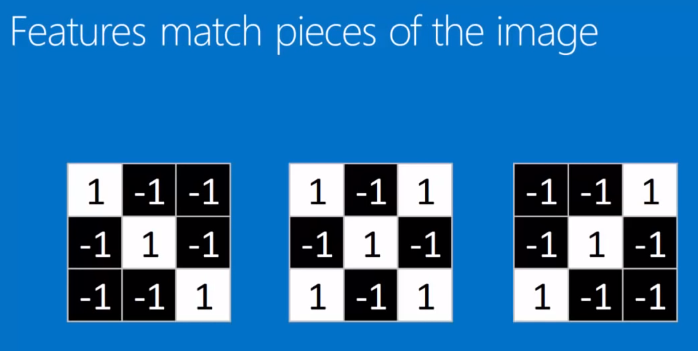


Figure 2.15 – Feature examples.

So examples of these so called features, or filters, are many little images as shown on figure above. In this case they are just 3 by 3 pixels. The one on the left is a diagonal line, slighting downwards from left to right, the one two right is same, but from right to left, and finally the middle one is like an X, or center.

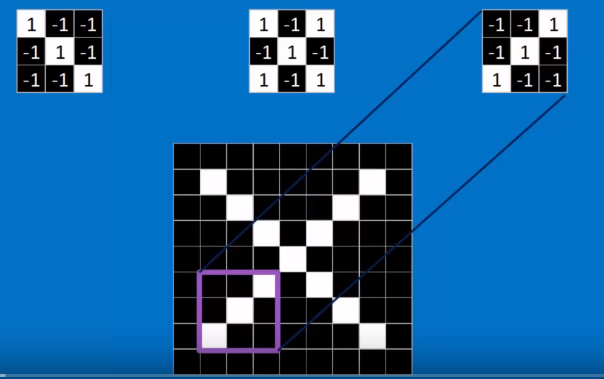


Figure 2.16 – Small features on our example.

And as you can see, as we go through and put the right feature on the right place, it matches absolutely perfect.

So now, when we have smaller pieces, or features, we can make a step deeper into what is going, and see the math behind this matching and filtering.

And the way it is done, is that the feature is line up with a little patch of the image. And the one by one the pixels are compared. They are multiplied by each other, then they are added up, and divided by total number of the pixels.

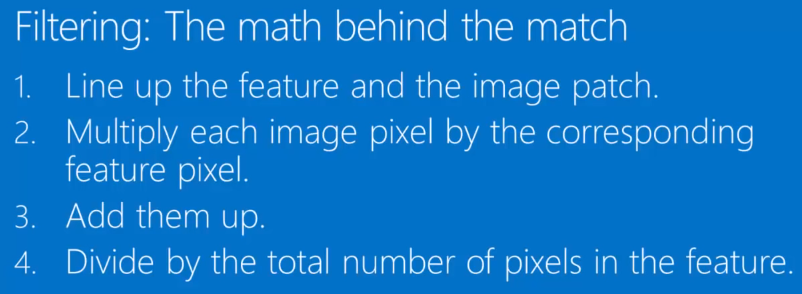


Figure 2.17 – Math algorithm behind the filtering.

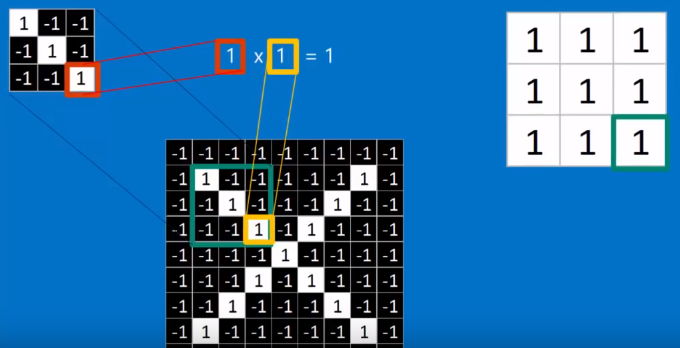
So now let’s take a brief look at some examples of using such mathematical algorithm for filtering on our previous examples.

Figure 2.18 – Step1: getting first matrix by multiplying each number of patch of image by each number of feature.

When we ended up with the first step, we take all that numbers from matrix to the right, add them up, and divide by amount of them.

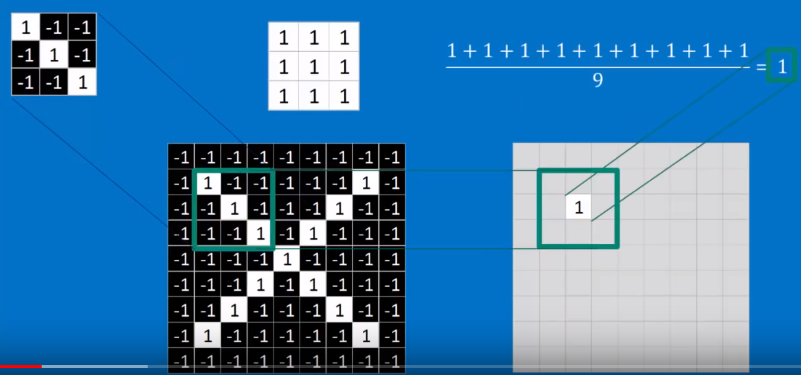


Figure 2.19 – Step 2: adding up middle matrix, and dividing it by its size.

As you can see now, after dividing our sum, we will get number in an interval between -1 and 1. In our case we got 1, and then we put this one to the new matrix of the same size, to the index corresponding to the center of patch of image.

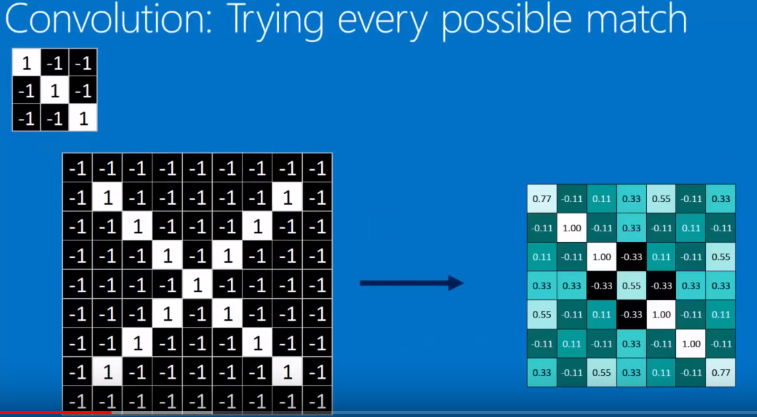
Moving our feature over the whole image, and applying this kind of filtering, we will finally get a new matrix of the same size as input image, but with new coefficients. So by moving our filter around the whole image we will get values of how good our filter matches to each pixel of input image. And finally it can be said, that it becomes a map of where each feature occurs.

Figure 2.20 – Step 3: getting a map of where each feature ocures.

So in general, by moving our filter over each possible position on our input image we actually do what is called convolution. That is just repeated application of this feature/filter over and over again. And what we get is a nice map of where each feature occurs. And if we look at it ourselves, it actually makes a lot of sense.

As we usually have more than one filter, we can apply each of them to our image. Finally we will get so many maps, as many filters we have:

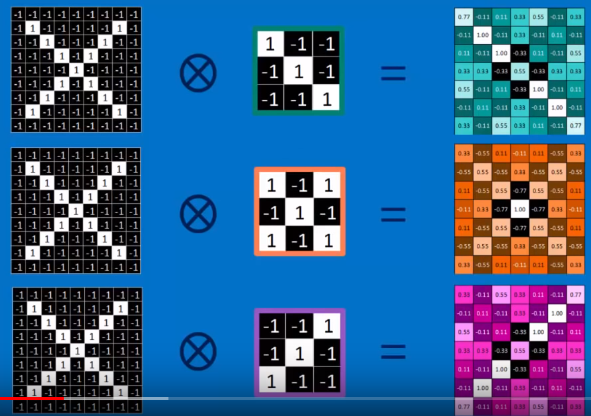


Figure 2.21 – Step 4: Applying each filter to our image, and getting corresponding maps.

This act of convolving an image with a bunch of filters/features and creating a stack of filtered images is actually what is called a convolution layer.

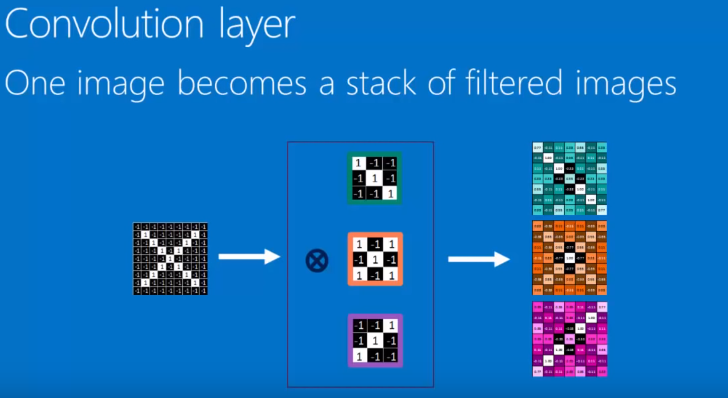


Figure 2.22 – Convolution Layer.

It is called a layer, because it is an operation that we can stack with others. After convolution one image becomes a stack of filtered images.

So convolution is only one trick that we have. The next trick we use is called Pooling, and its goal is to shrink an image stack.

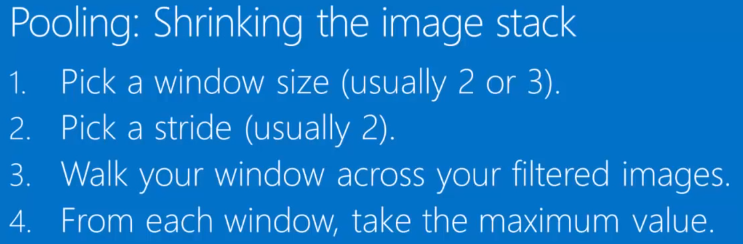


Figure 2.23 – Pooling algorithm.

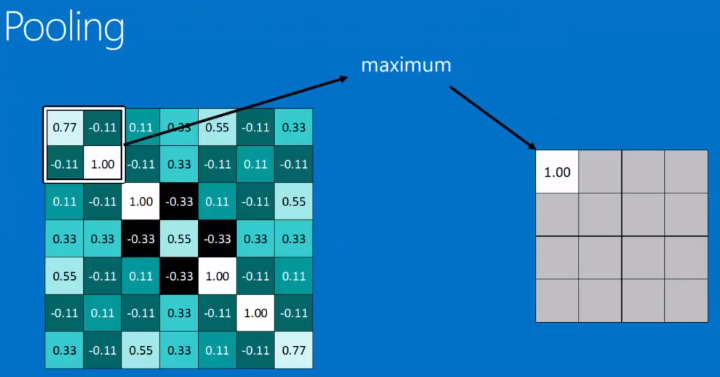
This part is probably the most straightforward. We just start with the window 2 by 2 size or 3 by 3. Then we just take this window, and stride it across our filtered images, and from each window we take the maximum value.

Figure 2.24 – Pooling example.

We are doing it for the whole image, and when we are end up with it, we get a similar pattern, but smaller. For example, when we started we had 7 by 7 matrix, and after that we get 4 by 4 one, twice smaller actually than it was. It make a great bunch of sense when we are working with really big images.



Figure 2.25 – Getting a smaller pattern after pooling.

Another important point that should be said on pooling, that it is not as sensitive to the positioning, so that it makes for us easier to identify features, as they can vary their position a bit.

All in all, we do max pooling with all of our stack of filtered images, and as a result we get the same stack, but with smaller images.

So, let’s move on to the third trick, that is called Normalization.

This step is done only in order to keep our math from blowing up, and keep it going to zero. All we do on this step, is change every negative values to zero.

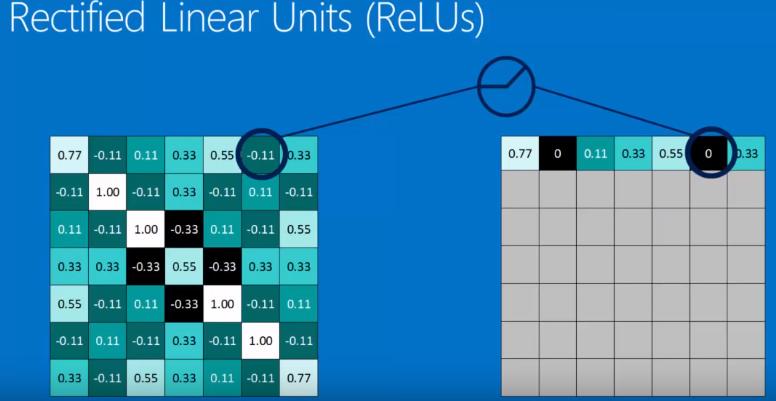
So, for instance, if we are looking back at our filtered images, we have these so called Rectified Linear Units(ReLUs). But actually all it does, is walking over our filtered images, and changes any unit that is negative to zero.

Figure 2.26 – Normalization and Rectified Linear Units.

By the time you are done, you have a similar looking filtered image, except it doesn’t have negative values anymore.

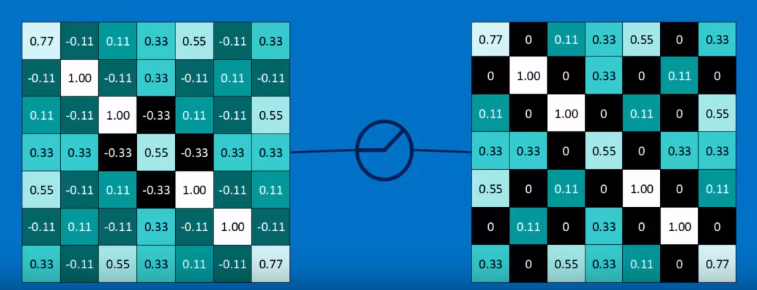


Figure 2.27 – Normalized filtered image.

Actually this Normalization process is another layer, that is called a ReLUs layer.

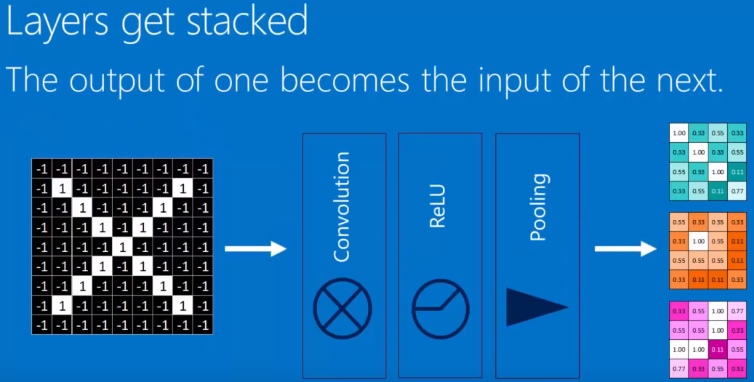
But the most important thing is that we can get our layers stacked.

Figure 2.28 – Stacked Layers.

This is possible thanks to the fact that each input and output data are just array of pixels.

What is more interesting, is that we can repeat our stacks, and that is called deep stacking.



Figure 2.29 – Deep Stacking.

Each time the image gets through a convolutional layer, it gets more filtered, and each time it gets through the pooling layer it gets smaller.

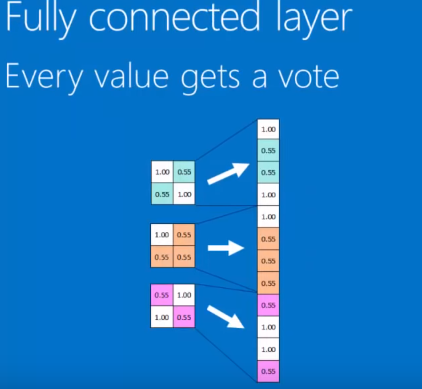
Now, the final level of our Convolutional Neural Networ is called a fully connected layer.

Figure 2.30 – Final layer (Fully connected layer)

Here every value gets a vote on what the final answer should be. Then each of those connects to one of our answers.

When we feed it to X, there would be certain values that would be high, that will predict strongly that it will be an X.

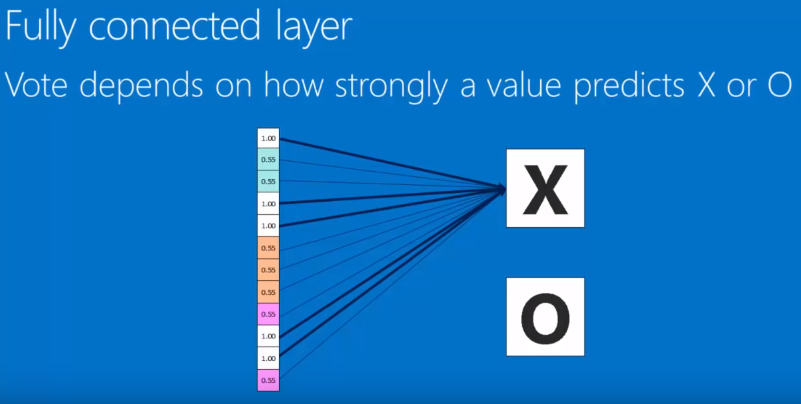


Figure 2.31 – Prediction for the X.

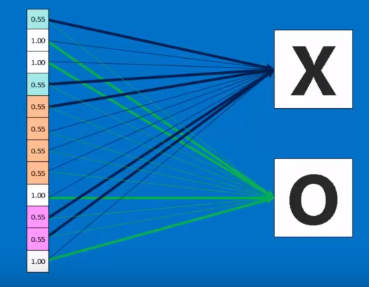
When we will feed it to the O, we will get another values, that will predict it to be an O.

Figure 2.32 – Prediction for O.

When we try to classify new values, we will get such result:

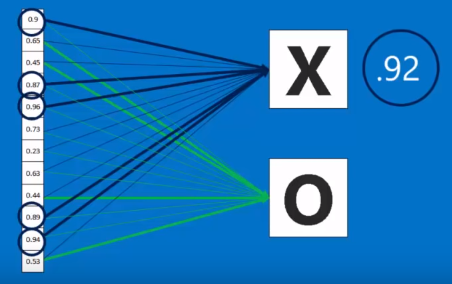


Figure 2.33 – Classifying new instance (Prediction for the X)

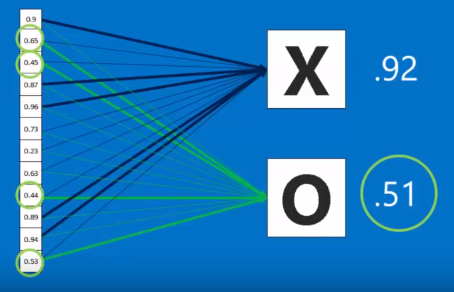


Figure 2.34 – Classifying new instance (Prediction for the O)

As we can see the better probability of classification is asinged to the X option, and it is equal to 0.92, so our CNN will classify this input instance as X.

3 EXPERIMENTAL STUDIES

3.1 Introduction to practical part

As part of experimental studies I have decided to classify three datasets with images. In the following paragraphs I will briefly talk about what I have done with those datasets.

3.2 Working on first dataset

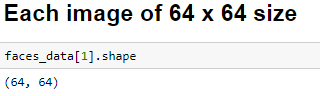
The first one is called Olivetti Faces. It contains grayscale faces 8 bit [0 – 255]. Again, it contains a few images of a several people. In total there are about 400 images, of 40 different people. Each image has a size of 64 by 64 pixels.

Figure 3.1 – Images size

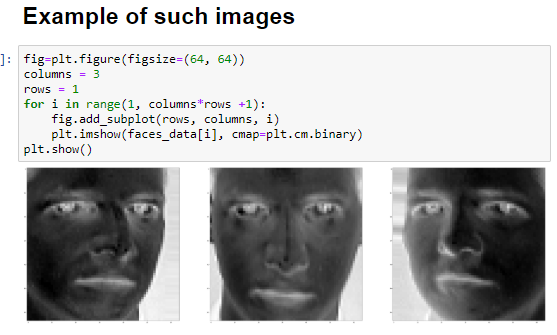
Here you can see some random examples of a pictures:

Figure 3.2 – Some examples of pictures.

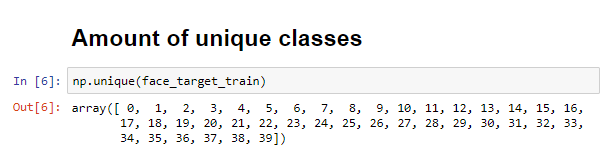
As I said before, we have got 40 unique classes, or 40 unique people, owners of faces we will work on.

Figure 3.3 – Unique target values for the 1st dataset.

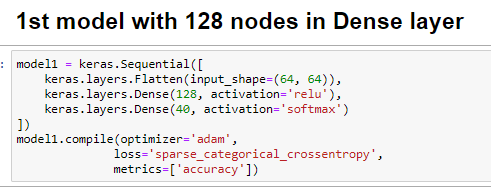
Firstly I have defined first model, that takes as an input and image of 64 x 64 pixels, that flattens it to 1D array. I also specified 128 nodes as a value for a Dense layer. And final layer has to have as many nodes, as we got classes, so it is equal to 40.

Figure 3.4 – First model Definition.

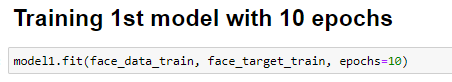
After that I started training our model, and decided to pick 10 epochs for start. Each epoch is just an entire process of learning algorithm an entire training set.

Figure 3.5 – Training model with 10 epochs.

As a result I got accuracy at about 6 percent on the last two levels, which is extremely low:

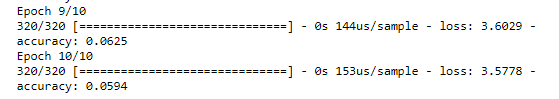


Figure 3.6 – Accuracy for the 1st model on 10 epochs.

With such a bad accuracy on a training data, I decided to train the same model on the 100 epochs, and here you can see result:

Figure 3.7 – Accuracy for the 1st model on 100 epochs.

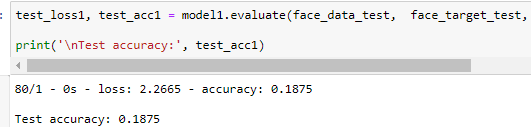
To the first glance, it can seem that it is not that ban in comparison to the second first training, but when testing on the train data, it gives only about 19 percent:

Figure 3.8 – Accuracy on test data for the 1st model on 100 epochs

That was still not enough for me, and I decided to define 2nd mode. This model has all same parameters, except penultimate layer now has 1024 values:

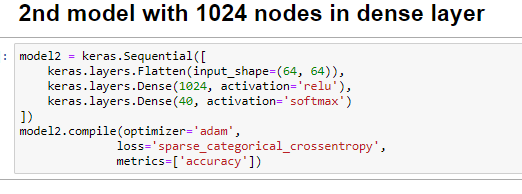


Figure 3.9 – 2nd Model definition

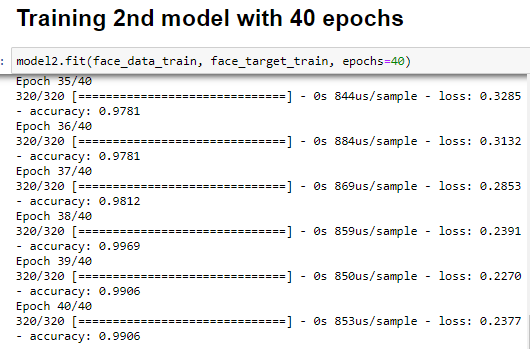
 Trying to guess optimal number of epochs for this model, I came to 40 epochs, and here are results of its training:

Figure 3.10 – Training 2nd model on 40 epochs.

As you can see accuracy on the train set gained almost 100 percent. When I was trying to train it on 100 epochs it gave me accuracy of 100 afte about 42 epoch, so I decided to reduce it to 40. Now let’s try to evaluate our model on test data:

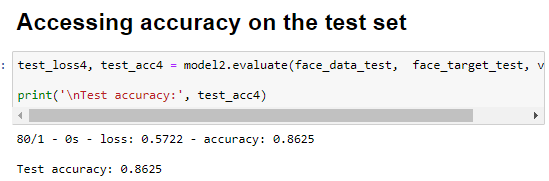


Figure 3.11 – Accesing 2nd model accuracy on test data

As you can see it gave us almost 87 percent accuracy, and I think it is a really great result.

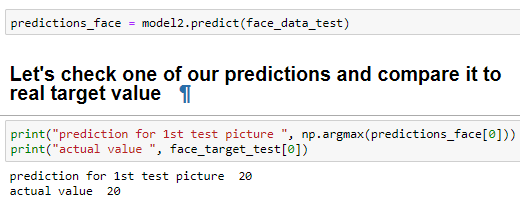
Now let’s make some predictions.

Figure 3.12 – Correct guessing

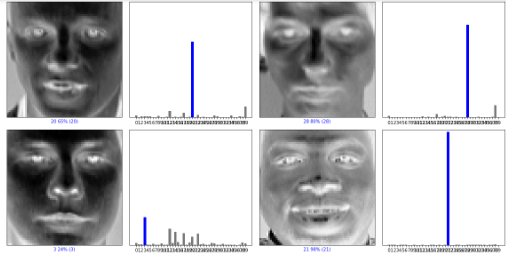
Again let’s output some more predictions:

Figure 3.13 – More predictions

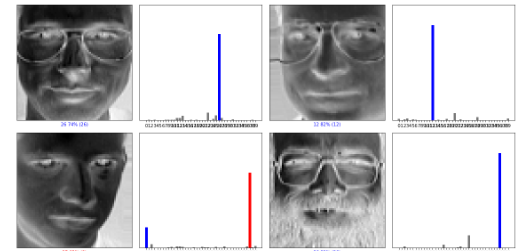
And even more predictions:

Figure 3.14 – More predictions

Correct predictions are highlighted with blue columns, and incorrect with red. When we have only blue line on histogram to the right, it means that we made a right guess, if red also appears, then we got mistake.

Now to sum up working with this dataset, my basic idea was to learn CNN to identify a person by a picture, so that if I add some more pictures of someone’s face to dataset, and that train model again, I can build a face id application based on that model. Another point I want to mention is that we had only 2D array of picture representation, as it was only grey scale picture. Later on I will show some with colored pictures.

3.3 Second dataset

The second dataset is called IntelImagClassification, and you can also find it on Kaggle under that name. This isn’t actually a dataset, that we used to, but is just a bunch of folders with images. So first of all to start working on it, we should specify how to download it correctly.

For that reason I have specified a function:



Figure 3.15 – Function for loading data

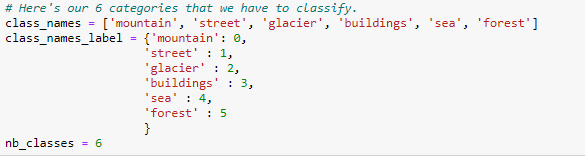
 Also, what I wanted to say, is amount om classes target function will have, and amount of data in total:

Figure 3.16 – Our classes

Figure 3.17 – Short information on our dataset

As you can see it has about 4700 images, but when I just found it, it had more than 25000, but it took too much time for my laptop to compute everything, so I was forced to reduce amount of data.

Here you can see percent of each class in train set:

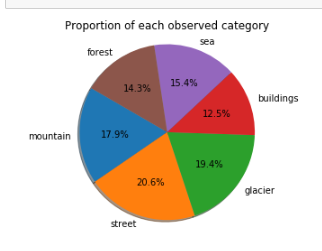


Figure 3.17 – Data visualization

Now let’s build first model:

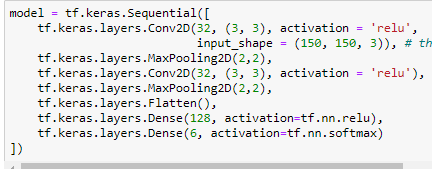


Figure 3.18 – Defining 1st model

As you can see now we are working with RGB images, and for thatreason, we specify shape in first layer that corresponds to 3D array.

´



Figure 3.19 – Training model

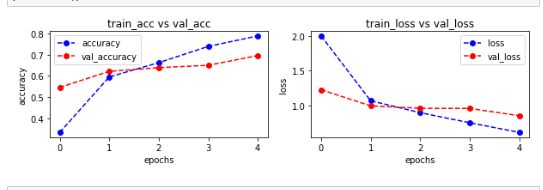
 Let’s take a look at dependency between accuracy/loss and number of epoch:

Figure 3.20 – Dependency between accuracy/loss and number of epoch

 Now let’s look at accuracy for this model on test data:

Figure 3.21 – Test data accuracy

You can see that accuracy is about 66 percent, not great, but not terrible. Let’s take a look at what classes are most often mislabeled according to our CNN:

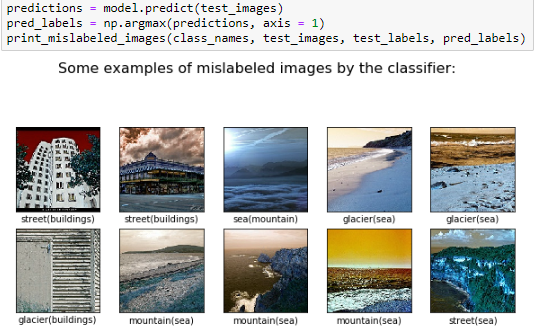


Figure 3.22 – Mispredicted classes

As you can see we mislabel mostly sea and glacier, street and building, and everything with mountain. All in all, it was an obvious result.

3.4 Third dataset

The third dataset is dedicated to the facial emotions of people. This dataset is called CK+ dataset for facial expression recognition. You can also find it on Kaggle. I decided to try to classify that dataset, because topic of facial emotion recognition seems to me really interesting.

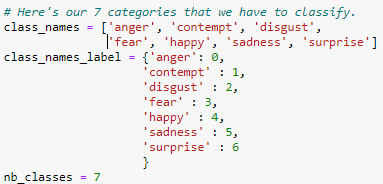
 So here is a brief description of dataset:

Figure 3.23 – Target classes

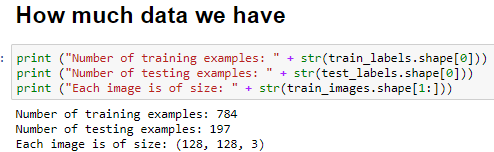
Brief information on data that we have:

Figure 3.24 Data amount

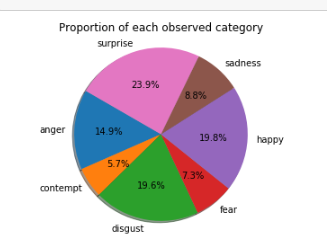
Here is a diagram to see percent of each class:

Figure 3.25 Data visualization

Let’s output some examples of our data:

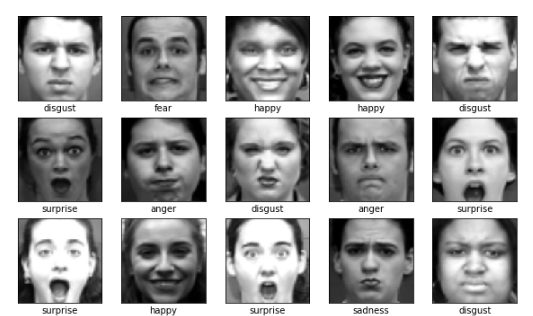


Figure 3.26 – Random images from our dataset

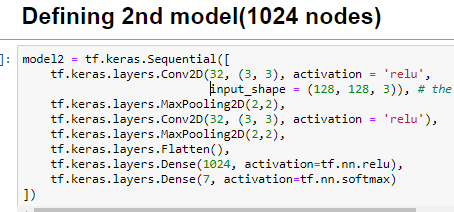
 Again, I have built several models for this dataset, but as I don’t have enough time to describe it, and enough space to fit it in my article, I will talk only about 1 model.

Figure 3.27 – Defining model

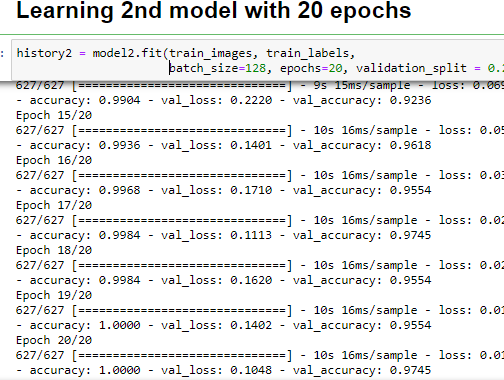


Figure 3.28 – Learning model on test data

As you can see I have decided to learn my model on 20 epochs, and on the last 20th epoch, it gave accuracy on validation set equal to almost 98 percent.

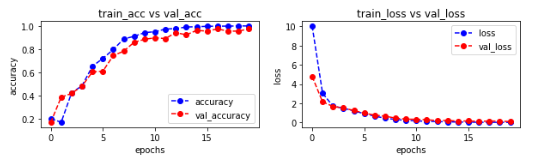


Figure 3.29 – Dependency between accuracy/loss and epoch number



Figure 3.30 – Accuracy on test data set is also very good

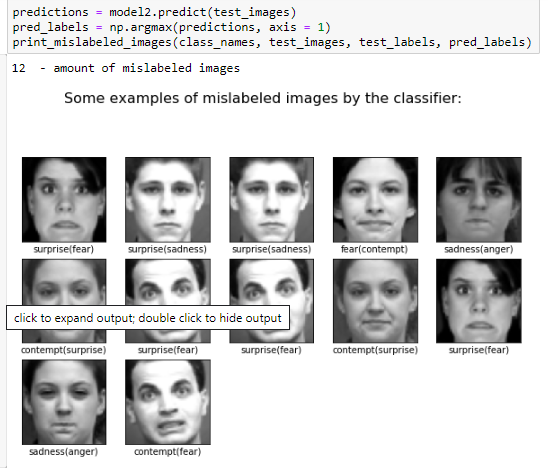
 Now let’s take a look at some mislabeled data:

Figure 3.31 – Mislabeld data

As you can see, we got only 12 incorrectly predicted images on total number of 197 images of test data. I think it is a very good result.

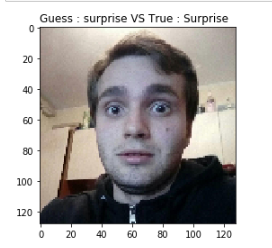
 Now, let’s try to predict emotions on some real pictures:

Figure 3.32 – Misha trying to express surprise

Let’s take a look at another instace:

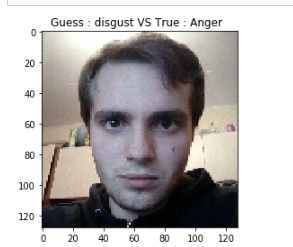


Figure 3.33 – Misha trying to be angry

And another one:



Figure 3.34 – CNN says that it is contempt emotion.

All in all, this area is very uncertain (Facial emotion expression recognition) because it is pretty hard even for person to correctly classify what facial emotion expression is displayed on each picture, and we can’t be sure, that person correctly trying to express each filling. Some people may have very slight differences in facial changes with emotional changes, other have obvious changes.

CONCLUSIONS

While doing this course project I have got myself familiar with a whole bunch of very interesting topics. I found out how images are viewed by computer; what types of images exists. Also I have found out what is the best method for image classification (obviously it is convolutional neural networks). In addition, I have learned how CNNs are working, I studied very precisely each step of this algorithm, an understood almost everything.

Moreover, I got incredibly good experience of learning something new for me, and very complicated in general, from scratch. I managed to use what I have learned practically in a way that was very interesting to me.

Also while doing this course project, I have been thinking a lot on areas of application of image classification, and came up with some interesting ideas, such as building an application to recognize faces. Not just detect, but recognize.

I actually did quite a lot more practical experiments, than I have described, and I think I will continue studying this area of Artificial Intelligence.

I think the most important thing of all that I have mentioned is that I more or less understood how Convolutional Neural Networks works.

REFERENCES

1. Tom M. Mitchell Machine Learning // McGraw-Hill Science/Engineering/Math, 1997. – 453 c.
2. Онлайн портал MachineLearning.ru [Електронний ресурс] – Режим доступу до ресурсу: <http://www.machinelearning.ru/>.
3. Відкритий курс з Machine Learning [Електронний ресурс] – Режим доступу до ресурсу:<https://medium.com/open-machine-learning-course/>.
4. Nishant Shukla Machine Learning with TensorFlow // Manning Publications, 2017. – 244 c.
5. Nick McClure TensorFlow Machine Learning Cookbook // Packt Publishing, 2017. – 483 c.

APPLICATION A

Program text

from \_\_future\_\_ import absolute\_import, division, print\_function, unicode\_literals

# TensorFlow and tf.keras

import tensorflow as tf

from tensorflow import keras

# Helper libraries

import numpy as np

import matplotlib.pyplot as plt

print(tf.\_\_version\_\_)

from scipy.io import loadmat

from sklearn.model\_selection import train\_test\_split

from scipy.io import loadmat  
faces\_data = np.load("olivetti\_faces.npy")

faces\_target = np.load("olivetti\_faces\_target.npy")

faces\_data[1].shape

fig=plt.figure(figsize=(64, 64))

columns = 3

rows = 1

for i in range(1, columns\*rows +1):

fig.add\_subplot(rows, columns, i)

plt.imshow(faces\_data[i], cmap=plt.cm.binary)

plt.show()

face\_data\_train, face\_data\_test, face\_target\_train, face\_target\_test = train\_test\_split(faces\_data, faces\_target, test\_size=0.2, random\_state=42)

np.unique(face\_target\_train)

model1 = keras.Sequential([

keras.layers.Flatten(input\_shape=(64, 64)),

keras.layers.Dense(128, activation='relu'),

keras.layers.Dense(40, activation='softmax')

])

model1.compile(optimizer='adam',

loss='sparse\_categorical\_crossentropy',

metrics=['accuracy'])

model1.fit(face\_data\_train, face\_target\_train, epochs=10)

test\_loss, test\_acc = model1.evaluate(face\_data\_test, face\_target\_test, verbose=2)

print('\nTest accuracy:', test\_acc2)

model1.fit(face\_data\_train, face\_target\_train, epochs=100)

test\_loss1, test\_acc1 = model1.evaluate(face\_data\_test, face\_target\_test, verbose=2)

print('\nTest accuracy:', test\_acc1)

model2 = keras.Sequential([

keras.layers.Flatten(input\_shape=(64, 64)),

keras.layers.Dense(1024, activation='relu'),

keras.layers.Dense(40, activation='softmax')

])

model2.compile(optimizer='adam',

loss='sparse\_categorical\_crossentropy',

metrics=['accuracy'])

model2.fit(face\_data\_train, face\_target\_train, epochs=10)

test\_loss2, test\_acc2 = model2.evaluate(face\_data\_test, face\_target\_test, verbose=2)

print('\nTest accuracy:', test\_acc2)

model2.fit(face\_data\_train, face\_target\_train, epochs=100)

test\_loss3, test\_acc3 = model2.evaluate(face\_data\_test, face\_target\_test, verbose=2)

print('\nTest accuracy:', test\_acc3)

model2.fit(face\_data\_train, face\_target\_train, epochs=40)

test\_loss4, test\_acc4 = model2.evaluate(face\_data\_test, face\_target\_test, verbose=2)

print('\nTest accuracy:', test\_acc4)

predictions\_face = model2.predict(face\_data\_test)

print("prediction for 1st test picture ", np.argmax(predictions\_face[0]))

print("actual value ", face\_target\_test[0])

def plot\_image(i, predictions\_array, true\_label, img):

predictions\_array, true\_label, img = predictions\_array, true\_label[i], img[i]

plt.grid(False)

plt.xticks([])

plt.yticks([])

plt.imshow(img, cmap=plt.cm.binary)

predicted\_label = np.argmax(predictions\_array)

if predicted\_label == true\_label:

color = 'blue'

else:

color = 'red'

plt.xlabel("{} {:2.0f}% ({})".format(predicted\_label,

100\*np.max(predictions\_array),

true\_label),

color=color)

def plot\_value\_array(i, predictions\_array, true\_label):

predictions\_array, true\_label = predictions\_array, true\_label[i]

plt.grid(False)

plt.xticks(range(40))

plt.yticks([])

thisplot = plt.bar(range(40), predictions\_array, color="#777777")

plt.ylim([0, 1])

predicted\_label = np.argmax(predictions\_array)

thisplot[predicted\_label].set\_color('red')

thisplot[true\_label].set\_color('blue')

# Plot the first X test images, their predicted labels, and the true labels.

# Color correct predictions in blue and incorrect predictions in red.

num\_rows = 20

num\_cols = 2

num\_images = num\_rows\*num\_cols

plt.figure(figsize=(2\*2\*num\_cols\*2, 2\*num\_rows\*2))

for i in range(num\_images):

plt.subplot(num\_rows, 2\*num\_cols, 2\*i+1)

plot\_image(i, predictions\_face[i], face\_target\_test, face\_data\_test)

plt.subplot(num\_rows, 2\*num\_cols, 2\*i+2)

plot\_value\_array(i, predictions\_face[i], face\_target\_test)

plt.tight\_layout()

plt.show()