МАТЕМАТИЧКА ГИМНАЗИЈА

МАТУРСКИ РАД

из предмета информатика и програмирање

Конволуцијске неуронске мреже за класификацију објеката

Ученик
Сергеј Стевановић
Одељење 4б

Ментор
проф. Милош Арсић

Београд, мај 2023.

Садржај

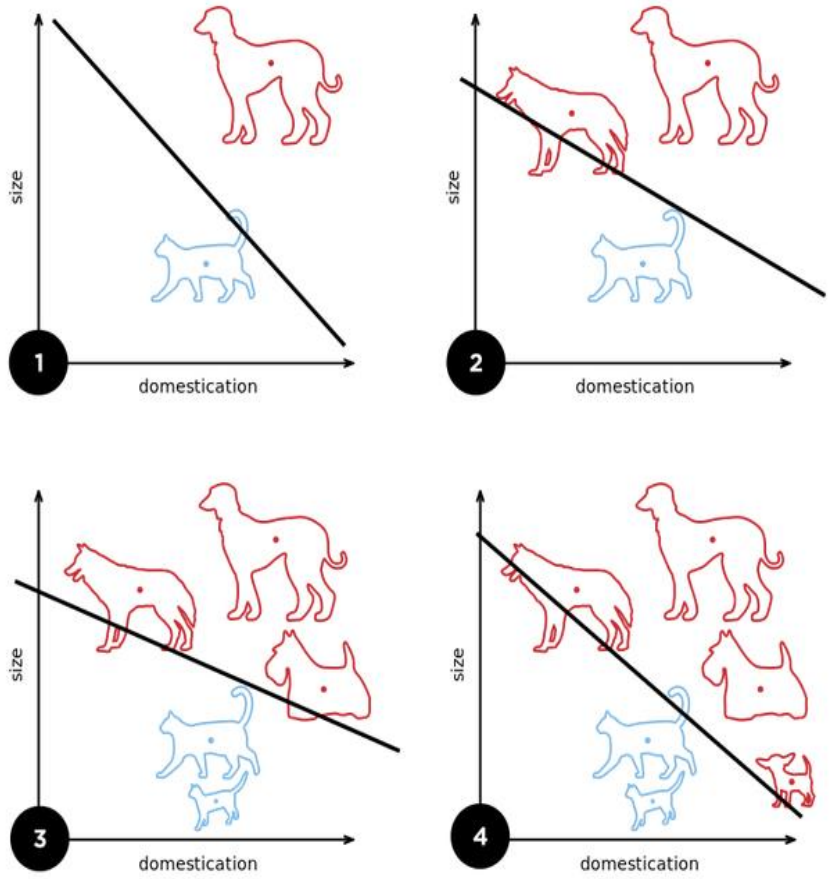
1. Увод

1.1 Инспирација

Неуронске мреже, у контексту, вештачке интелигенције, представљају модел биолошких неурона, ћелија нервног система, и веза између њих. Без дилеме двојица научника који су непроцењиво допринели области вештачке интелигенције јесу Александар Беин (Alexander Bain) и Вилијам Џејмс (William James). Њихови радови су теоретисали да су наша осећања управо продукт интеракција међу неуронима. По Беину, свака акција резултовала би ексцитацијом одређене групе неурона. Џејмсова теорија је у великој мери налик Беиновој, међутим, њих две се разликују по питању настанка меморија (сећања). Беин је сматрао да константно понављање радњи доводи до јачања веза између ексцитованих ћелија што доводи до формирања меморија. За разлику од њега, Џејмс је сматрао да ток електричне струје међу неуронима проузрокује наша осећања.

1.2 Перцептрон

Перцептрон су 1943. године дизајнирали Верен МекКулох (Warren McCulloch) и Валтер Питс (Walter Pitts) ради класификације слика. Упркос томе што је оригинално дизајниран као софтвер прва имплементација остварена је у виду IBM-704 (mainframe дигиталног рачунара) направљеног 1958. у Корнвелској ваздухопловној лабораторији и то од стране Френка Розенблата под покровитељством Америчке владе. Перцептрон представља једну од првих креација способних да врши линеарну класификацију и као такав може бити посматран као претеча свих модерних неуронских мрежа.



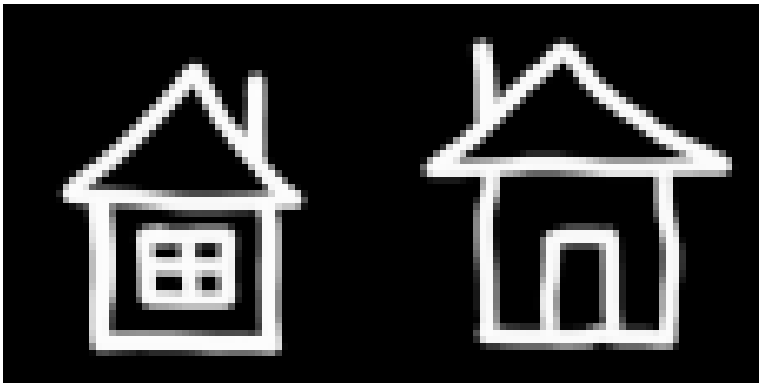
Слика 1.2.1, перцептрон класификује слике помоћу линеарне функције. Извор:

https://en.wikipedia.org/wiki/Perceptron.

Наиме, перцептрон је имао неколико недостатке, најзначајнији од њих је то што није могао да буде истрениран да класификује податке који нису линеарно одвојиви. У почетку се, супротно истини, веровало да употребом вишеслојних перцептронских мрежа не би било могуће превазићи те проблеме те је развој вештачке интелигенције у великој мери стагнирао све то осамдесетих година двадесетог века.

1.3 Мотивација

Мале фигуре које заузимају простор од свега 57 * 57 пиксела налазе се пред нама. Наш мозак нема никакав проблем да препозна да се на црно белим сликама налазе мале кућице, међутим, када бисмо хтели да направимо компјутерски програм који би препознавао да ли се на сличицама налазе кућице, бродићи или пак нешто друго, схватили бисмо колико је наш мозак напреднији и сложенији (што врло често узимамо здраво за готово) од било каквог програма који можемо направити. Пут од међусобно различитих улазних параметара, боја пиксела, па до финалне класификације у једну од многих категорија није ни мало тривијалан и лак. Наведимо пар примера улазних параметара.



Слика 1.3.1

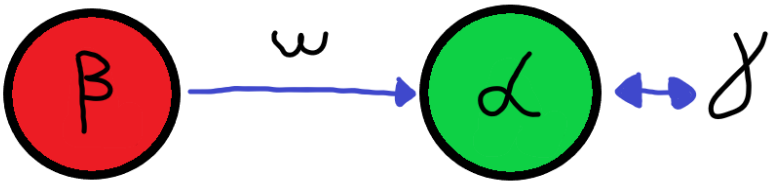
Слика 1.3.2

Слика 1.3.3

Упркос томе што су неурони који преносе акциони потенцијал потпуно различити у зависности од тога коју од датих слика посматрамо, нешто у нашем визуелном кортексу мозга ипак повезује све три наведене слике у једну категорију. Помоћу вештачке интелигенције ми покушавамо да симулирамо тај процес повезивања унутар компјутера.

2. Функционисање неуронских мрежа

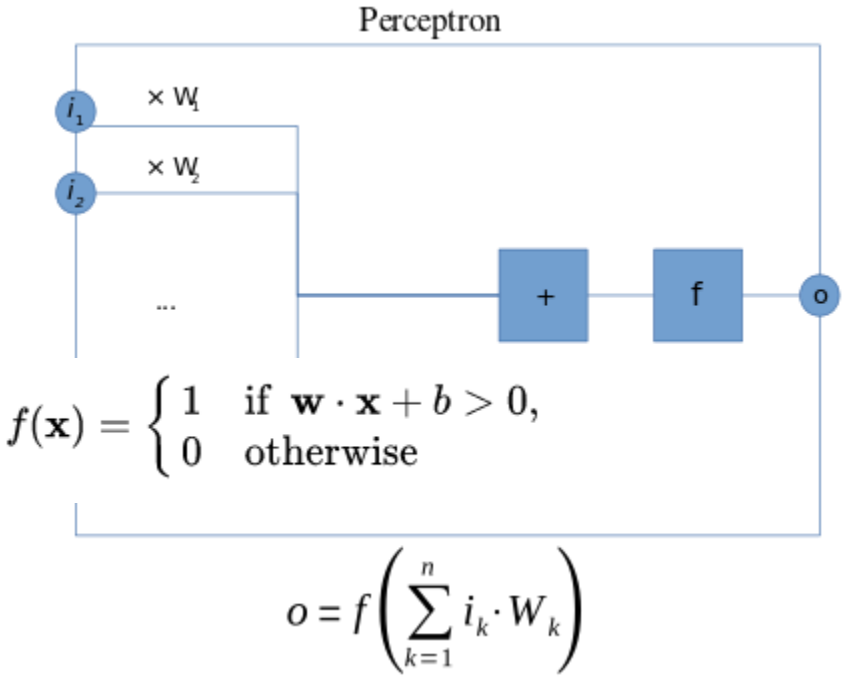
Да бисмо објаснили како неуронске мреже функционишу посматрајмо један од најједноставнијих пример. Неуроне у овом примеру посматраћемо као ентитете који садрже бројчану вредност која представља њихову ексцитацију (инспирисано структуром нервног система). Ту ексцитацију ћемо даље у раду називати активација. Нека се наша неуронска мрежа састоји од два неурона као што је приказано на слици испод.



Слика 2.1 Приказ једноставне неуронске мреже сачињене од два неурона и једног тежинског фактора.

Нека се α израчунава по следећој формули: $\alpha = \beta \times \omega$, и нека се тај резултат пореди са вредношћу γ која представљала очекивани резултат (на пример, уколико бисмо хтели да вредност α буде дупло већа од вредности β за $\beta = 1$, γ би било једнако 2). Структуру приказану изнад тада можемо назвати неуронском мрежом сачињеном од једног улазног слоја и једног излазног слоја. Као што им имена налажу, улазни слој представља место где подаци долазе на обраду а излазни слој представља место где очитавамо резултате обраде. Ова неуронска мрежа садржи један параметар и он је управо ω. Његовим мењањем се директно мења веза између α и β што се види из формуле изнад. Кроз процес такозваног тренирања, ми мењамо вредност тог параметра на начин тако да се вредности α и γ разликују све мање и мање.

Посматрајмо сада структуру перцептрона. Врло слично претходном примеру, вредност активације, то јест, излазног параметра овог израчунавања зависи од вредности производа активација и одговарајућих тежинских фактора из претходног слоја, међутим, како постоји више улазних података то се за аргумент функције узима њихова сума. Сама функција је још један новитет у односу на претходно наведен пример и њена имплементација је у великој мери инспирисана људским нервним системом, где посматрани неурони (ћелије нервног система) немају континуиран скуп могућих активација већ бинарне вредности које се могу тумачити као активан, то јест, неактиван неурон.



Слика 2.2
Структура перцептрона

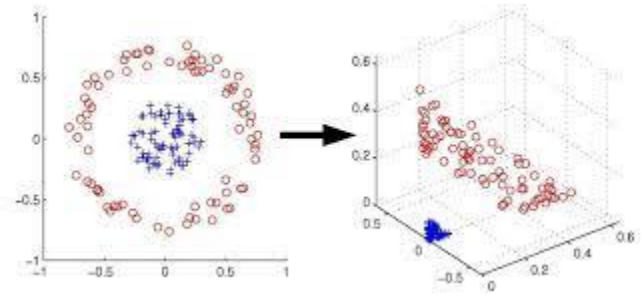
Функција која се примењује на суму производа је дефинисана на следећи начин:

Слика 2.3
Дефиниција активационе функције перцептрона из датог примера, слика са

https://en.wikipedia.org/wiki/Perceptron

Где b представља Bias самог перцептрона. Оваква структура, као што је раније поменуто, има недостатке.

Усложњавањем претходног система додавањем више слојева паралелних перцептрона међусобно повезаних са сваким перцептроном из плићег и дубљег слоја могуће је употпунити концепт нелинеарности (класификације података који нису линеарно одвојиви), који налаже да се за сваки индивидуални податак може направити нова димензија посматрања што значи да се на крају може формирати хиперраван која тачно и прецизно класификује све параметре.



Слика 2.4
Пример линеарно нераздвојивих података (с леве) и њихове репрезентације у вишедимензионом простору (с десне стране). Слика са

https://stats.stackexchange.com/questions/449791/can-non-linearly-separable-data-always-be-made-linearly-separable

Оваква неуронална мрежа, под условом да је довољно сложена (да садржи довољно слојева, где сваки слој садржи разуман број неурона), готово да нема ограничења по питању за шта може бити истренирана. Једно од пак главних ограничења јесте сам процес тренирања, то

јест, доступност података, њихова поузданост и хардверске лимитације. Због ових ограничења појављују се многи типови неуронских мрежа оптимизованих за употребу у различитим областима међу којима су неки од најпопуларнијих модела:

1. Convolutional Neural Networks;
2. Binary Neural Networks;
3. Generative Adverserial Networks;
4. Reccurent Neural Networks.

3. Популарни типови неуронских мрежа

Конволуцијске неуронске мреже су изузетно популарне приликом обраде слика (класификације, препознавања...) због своје структуре која им омогућава да огромну количину улазних параметара (најчешће у виду слика великих резолуција) обраде коришћењем конволуција (математичке операције) чиме се у великој мери смањује зависност од јаког хардвера.

Бинарне неуронске мреже су на нивои индивидуалних неурона готово идентичне структуре као први модели перцептрона у смислу да је њихова активациона функција враћа излаз дискретног типа (или 0 или 1). Будући да су оне у поређењу са осталим поменутим примерима најтолерантније по питању хардвера оне се најчешће примењују у мањим слабијим уређајима (слабијих перформанси) чија је сврха прикупљају податке који се потом користе за тренирање већег модела који потом истренириране параметре шаље назад корисничким уређајима. Овакве структуре су свој процват доживеле уз епоху CLOUD-а.

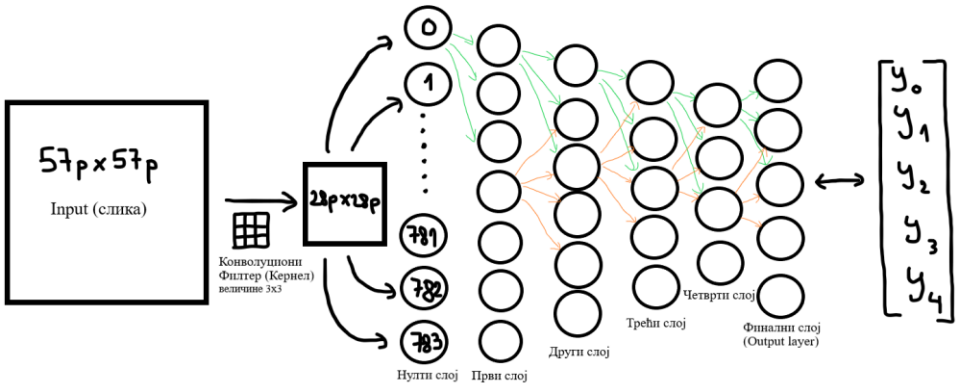
Генеративне супарничке мреже су се први пут појавиле 2014. Њихова структура подразумева две неуронске мреже које се симултано тренирају тако да је побољшање једне мреже представља проблем другој. Конкретан пример оваквих интеракција би биле мреже за генерисање слика на основу задатих појмова. Задатак једне мреже је да кроз тренинг научи како да синтетише детаље који слику чине реалистичном док је задатак друге мреже да уочава проблеме са сликом који је чине нереалистичном.

Рекурентне неуронске мреже су специјални типови мрежа код којих свака пропагација напред (израчунавање вредности активација неурона на основу улазних података све до финалног слоја) утиче на следеће израчунавање. Овакве мреже су изузетно погодне за анализирање садржаја који захтева познавање контекста. Један од примера примена оваквих система би био систем за анализирање текста и формирање аудио записа на основу записаних речи. Овакви проблеми интуитивно зависе од контекста будући да, на пример, акцентовање одређеног слога унутар аудио записа зависи од слогова пре и после.

Сви ови модели су засебно оптимизовани за одређене проблеме при чему је битно напоменути да један систем може примењивати више претходно наведених структура.

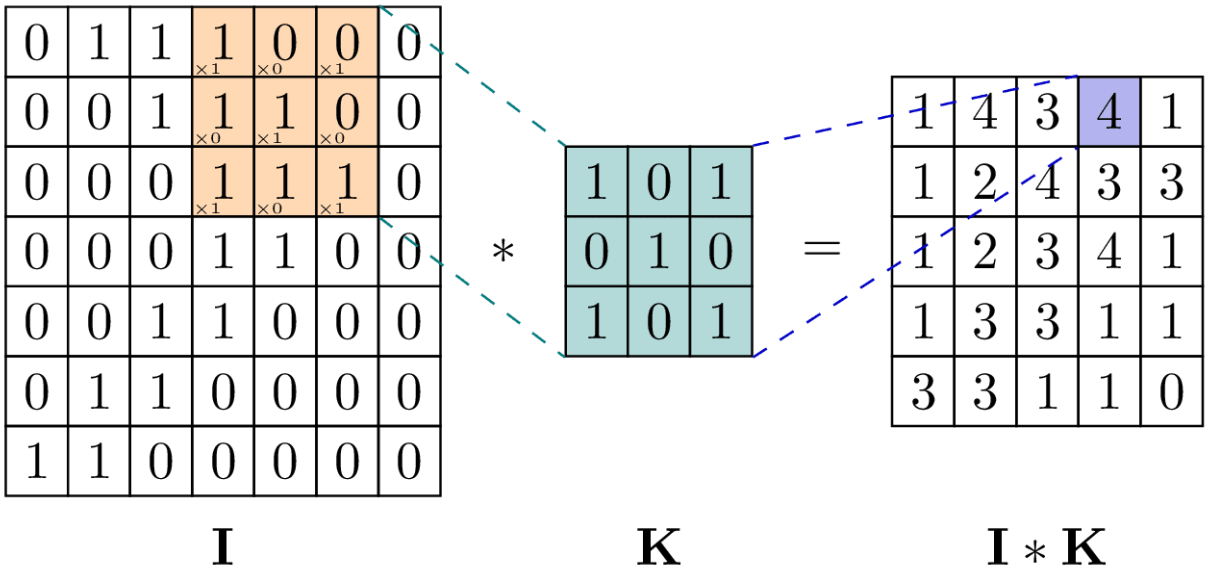
4. Структура неуронске мреже овог пројекта

Будући да је главни циљ пројекта класификација слика (што подразумева препознавање слика) готово никакво изненађење не би требало да представља мој избор типа мреже. Конволуцијске неуронске мреже се први пут појављују осамдесетих година двадесетог века дубоко инспирисане радовима Дејвида Хубела и Торстена Визела из педесетих година истог века. Оне су се кроз време показале као изузетно добро решење за проблем класификације слика због тога што је математичка операција конволуције транслационо-еквиваријантна (тумачење ,,битног" дела слике се не мења у зависности од позиције).

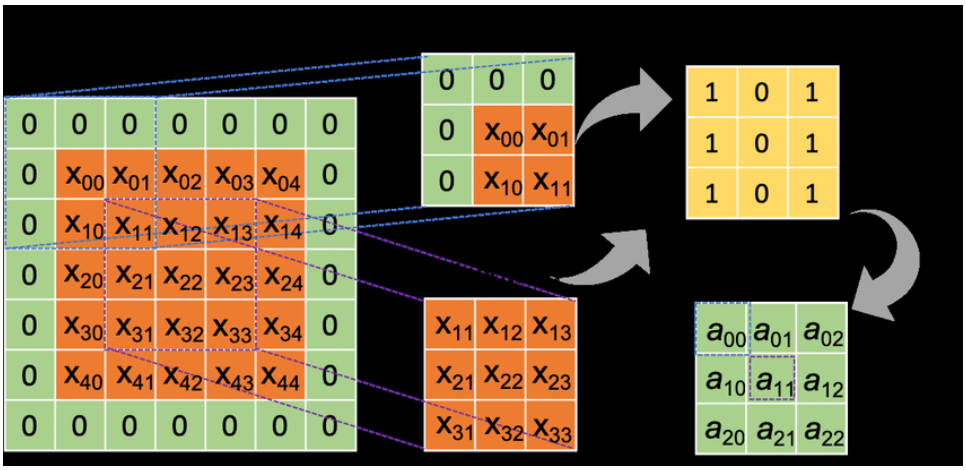


Слика 4.1 Схематски приказ неуронске мреже описане у овом раду

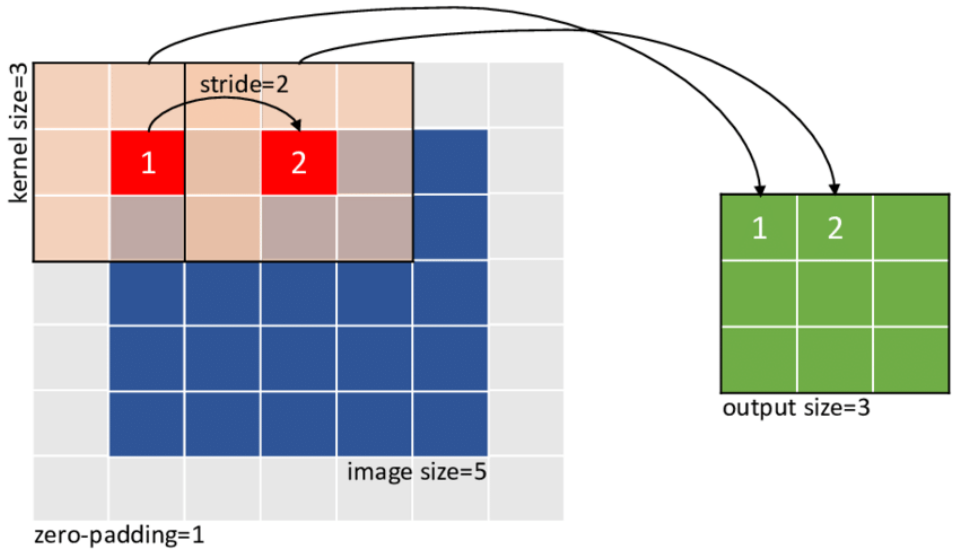
Операција конволуције може се интуитивно објаснити на следећи начин: превлачењем матрице назване филтер (у литератури се врло често јавља и реч Kernel) која садржи девет тежинских фактора (у општем случају садржи N^2 фактора, где N представља димензију филтера) и која се превлачи преко матрице улазних података. Параметри следећих слојева се добијају као збир производа одговарајућих активација и тежинских фактора из филтера ,,изнад" њих. Како би матрица резултата ове операција по димензијама била мања од полазне, филтер се након сваког израчунавања помера за по два места, при чему се број померених места означава са STRIDE. Уколико проблем захтева да, на пример, STRIDE буде једнак јединици и да се димензије матрице операцијом не промене, могуће је улазне параметре проширити такозваним PADDING-ом који представља омотач око података који не утиче на вредност израчунавање (на пример акција blur-овања).



Слика 4.2
Приказ операције конволуције извршене над матрицом. Извор: https://www.google.com/url?sa=i&url=https%3A%2F%2Ftikz.net%2Fconv2d%2F&psig=AOvVaw2ss-EdSNMETwvgYr9SyQEz&ust=1685026222926000&source=images&cd=vfe&ved=0CBEQjRxqFwoTCPClgv2Zjv8CFQAAAAAdAAAAABAE



Слика 4.3
Приказ конволуције над матрицама уз примену PADDING слоја дебљине један.
Извор слике: https://www.researchgate.net/figure/An-illustration-of-padding-and-convolution-operations-in-the-CNN-model-Suppose-that-the_fig1_332463100



Слика 4.4 Пример конволуције где је STRIDE једнак два, димензија филтера (kernel-a) три пута три и дебљина PADDING-a једнака један. Извор:

https://www.google.com/url?sa=i&url=https%3A%2F%2Fwww.researchgate.net%2Ffigure%2FExample-of-a-square-image-convolution-with-zero-padding-While-training-a-CNN-there-are_fig3_340500073&psig=AOvVaw22x_xjZMv60dUlAgraivrh&ust=1685026375020000&source=images&cd=vfe&ved=0CBEQjRxqFwoTCMDhv8Wajv8CFQAAAAAdAAAAABAx

Након проласка кроз један конволуцијски слој улазни параметри се убацују у први слој потпуно повезане мреже, који се састоји од нултог слоја сачињеног од 784 индивидуална неурона, првог слоја сачињеног од седам неурона, другог слоја сачињеног од шест неурона, трећег слоја сачињеног од пет неурона, четвртог слоја сачињеног од четири неурона и финалног слоја сачињеног од пет неурона. Унутар сваког од потпуно повезаних слојева сваки неурон је помоћу одговарајућег тежинског фактора повезан са сваким неуроном из следећег слоја, изузев, наравно, последњег, финалног слоја. Активациона функција коришћена приликом пролаза унапред јесте ELU (Exponential Linear Unit), која је дефинисана на следећи начин:

$$f(x) = \begin{cases} x & \text{if } x > 0, \\ a\left(e^x - 1\right) & \text{otherwise.} \end{cases}$$

Слика 4.5
Дефиниција ELU активационе функција, слика са

https://en.wikipedia.org/wiki/Rectifier_(neural_networks)#ELUs

Ово активациона функција добијена је другачијим дефинисањем за вредности негативног *x* активационе функције ReLU (Rectified Linear Unit) која за такве *x* има вредност 0.

$$f(x) = x^+ = \max(0, x) = \frac{x + |x|}{2} = \begin{cases} x & \text{if } x > 0, \\ 0 & \text{otherwise.} \end{cases}$$

Слика 4.6
Дефиниција ReLU активационе функције, слика са

https://en.wikipedia.org/wiki/Rectifier_(neural_networks)

ReLU је једна од најпопуларнијих опција при одабиру активационе функције данашњице због тога што омогућава стабилан проток активација и готово непревазиђено време конвергирања у алгоритмима за тренирање (backpropagation), међутим, она ипак има пар недостатака. Приликом употребе ове функције може доћи до такозваног проблема „умируђих неурона". Овај проблем настаје приликом употребе превеликих коефицијената учења, о којима ће бити више речи у делу о backpropagacij-и, који приликом примена алгоритама за учење доводе до потпуног прекида употребе неурона што резултује смањењем капацитета мреже. Овај проблем се у великој мери може решити променом дефиниције за негативне вредности *x* чиме се неурони ефективно ваде из мртвог стања.

5. Имплементација алгоритама за пролазак унапред

Како би се израчунала вредност активације неурона у ненултом слоју (првом слоју и надаље), потребно је применити функцију ELU, чији би аргумент био једнак збиру суме производа одговарајућих тежинских фактора и активација неурона из претходног слоја и Bias-а самог неурона коме израчунавамо активацију. Тај Bias представља додатни параметар чија се вредност с тренирањем мења и јединствен је за сваки неурон.

Како бисмо имплементирали претходно наведене алгоритме најпре је потребно да иницијализујемо све тежинске факторе и привржености (bias) неурона.

```
double[,] WeightL0 = new double[7, 784];

double[,] WeightL1 = new double[7, 6];

double[,] WeightL2 = new double[6, 5];

double[,] WeightL3 = new double[5, 4];

double[,] WeightL4 = new double[4, 5];


double[] Bias0 = new double[7];

double[] Bias1 = new double[6];

double[] Bias2 = new double[5];

double[] Bias3 = new double[4];

double[] Bias4 = new double[5];
```

Приликом првог покретања програма тежинским факторима ће вредности бити насумично додељене на следећи начин

```
FillMatrixAtRandom(ref WeightL0);

FillMatrixAtRandom(ref WeightL1);

FillMatrixAtRandom(ref WeightL2);

FillMatrixAtRandom(ref WeightL3);

FillMatrixAtRandom(ref WeightL4);
```

При чему је функција FillMatrixAtRandom имплементирана тако што се свакој позицији додељује вредност између нуле и јединице од чега се потом одузима 0.5. Упркос томе што се априори подразумева да је параметар типа double[,] референтан кроз даљи рад ће ипак бити коришћена реч ref ради једноставнијег праћења тока дешавања.

```
void FillMatrixAtRandom(ref double[,] matrix)

{

    Random random = new Random();


    for (int i = 0; i < matrix.GetLength(0); i++)

        for (int j = 0; j < matrix.GetLength(1); j++)

            matrix[i, j] = random.NextDouble() - 0.5;

}
```

Након првог покретања програма вредности свих тежинских фактора и привржености ће бити сачуване у текстуалним фајловима те употреба претходно наведене методе неће бити потребна.

```
WeightL0 = MatrixInput("WeightL0.txt");

WeightL1 = MatrixInput("WeightL1.txt");

WeightL2 = MatrixInput("WeightL2.txt");

WeightL3 = MatrixInput("WeightL3.txt");

WeightL4 = MatrixInput("WeightL4.txt");


Bias0 = ArrayInput("Bias0.txt");

Bias1 = ArrayInput("Bias1.txt");

Bias2 = ArrayInput("Bias2.txt");

Bias3 = ArrayInput("Bias3.txt");

Bias4 = ArrayInput("Bias4.txt");
```

Функције којима се учитавају параметри из текстуалних датотека као параметар имају локацију датотеке и разликују се по враћеном типу у зависности од тога да ли учитавају тежине или привржености.

```
static double[,] MatrixInput(string FilePath) //Тежине
{
    Console.WriteLine("Reading matrix from: " + FilePath);
    var sr = new StreamReader(FilePath);
    var temp = sr.ReadLine();

    double[,] matrix = new double[int.Parse(temp.Split()[0]),
int.Parse(temp.Split()[1])];
    for (int i = 0; i < matrix.GetLength(0); i++)
    {
        temp = sr.ReadLine();
        for (int j = 0; j < matrix.GetLength(1); j++)
            matrix[i, j] = double.Parse(temp.Split()[j]);
        if (sr.EndOfStream) break;
    }

    sr.Close();
    sr.Dispose();
    Console.WriteLine("End of input.  ");
    return matrix;
}
```

Привржености (Bias-и) се учитавају следећом методом:

```
static double[] ArrayInput(string FilePath)
{
    Console.WriteLine("Reading array from: " + FilePath);
    var sr = new StreamReader(FilePath);

    double[] array = new double[int.Parse(sr.ReadLine())];
    string temp = sr.ReadLine();

    for (int i = 0; i < array.Length; i++)
        array[i] = double.Parse(temp.Split()[i]);

    sr.Close();
    sr.Dispose();
    Console.WriteLine("End of input.  ");
    return array;
}
```

Након иницијализације и учитавања параметара потребно је креирати помоћну класу типа NeuralNetState која међу атрибутима садржи привржености и активације свакок индивидуалног неурона.

```
class NeuralNetState

{

    public double[,] InputMatrix { get; set; } = new double[57,
57];

    public double[,] AfterK0 { get; set; } = new double[28, 28];

    public double[] FirstLayer { get; set; } = new double[7];

    public double[] FirstBias { get; set; } = new double[7];


    public double[] SecondLayer { get; set; } = new double[6];

    public double[] SecondBias { get; set; } = new double[6];


    public double[] ThirdLayer { get; set; } = new double[5];

    public double[] ThirdBias { get; set; } = new double[5];


    public double[] FourthLayer { get; set; } = new double[4];

    public double[] FourthBias { get; set; } = new double[4];


    public double[] OutputLayer { get; set; } = new double[5];

    public double[] OutputBias { get; set; } = new double[5];

    ⋮
```

Прва два атрибута представљају, редом, улазне параметре програма (након конверзије) и резултат једне примене операције конволуције. Унутар ове класе такође се налази конструктор са параметрима који одговарају редоследу атрибута.

```
NeuralNetState stateOfNN = new NeuralNetState(…
```

Након креирања овог објетка потребно је учитати саму сличицу позивом следећих метода

```
Bitmap bm = new Bitmap("Kucica\\1.png"); //57p x 57p димензија

stateOfNN.InputMatrix = GetBitMapColorMatrix(bm);
```

Учитана Bitmap-а ће се методом GetBitMapColorMatrix() конвертовати у матрицу типа double[,] како би вршење манипулација над њом било лакше.

```
double[,] GetBitMapColorMatrix(Bitmap Image)

{

    int hight = Image.Height;

    int width = Image.Width;


    double[,] result = new double[hight, width];

    for (int i = 0; i < hight; i++)

        for (int j = 0; j < width; j++)

            result[j, i] = (double)(

Image.GetPixel(i, j).R + Image.GetPixel(i, j).B +
Image.GetPixel(i, j).B) / 765;

    return result;

}
```

Тада ће се приликом позива функције за испис позвати функција која ће израчунати активације сваког од неурона методом ForwardPass()

```
Console.WriteLine(string.Join(" ", ForwardPass(…));
```

Ова метода враћа податак типа double[] а од параметара захтева објекат типа NeuralNetState и више објеката типа double[,] који репрезентују, редом, конволицијски кернел и тежинске факторе.

```
double[] ForwardPass(ref NeuralNetState StateOfNN....)

{

    StateOfNN.AfterK0 = Convolution(StateOfNN.InputMatrix,
KernelL0, 2, 0);


    double[] InputVector = new double[784];

    for (int i = 0; i < 784; i++)

        InputVector[i] = StateOfNN.AfterK0[i / 28, i % 28];


    for (int i = 0; i < 7; i++)… //Петља 1

    for (int i = 0; i < 6; i++)...

    for (int i = 0; i < 5; i++)...

    for (int i = 0; i < 4; i++)...

    for (int i = 0; i < 5; i++)...


    return StateOfNN.OutputLayer;

}
```

Где свака горе наведена петља без тела изгледа као претходна, уз мале модификације индекса и атрибута унутар StateOfNN над којима се манипулише.

```
for (int i = 0; i < 7; i++) //Петља 1
    {
        StateOfNN.FirstLayer[i] = 0;
        for (int j = 0; j < 784; j++)
            StateOfNN.FirstLayer[i] += InputVector[j] *
WeightL0[i, j];
        StateOfNN.FirstLayer[i] = ELU(StateOfNN.FirstLayer[i] +
StateOfNN.FirstBias[i]);
    }
```

Враћена вредност ове функције, низ реалних вредности из последњег слоја, ће потом бити приказан као и дистрибуција вероватноће међу резултатима, израчуната путем методе SoftMaxProbabilityDistribution(),

```
double[] SoftMaxProbabilityDistribution(double[] Input, double
Base = Math.E)
{
    double sum = 0;
    for (int i = 0; i < Input.Length; i++)
        sum += Math.Pow(Base, Input[i]);
     double[] result = new double[Input.Length];
    for(int i = 0; i < Input.Length; i++)
        result[i] = Math.Pow(Base, Input[i]) / sum;

    return result;
}
```

Битно би било напоменути да сума свих вредности враћеног низа ове функције у збиру даје један за разлику од података финалног слоја неуронке мреже.

6. Објашњење и имплементација алгоритма backpropagation

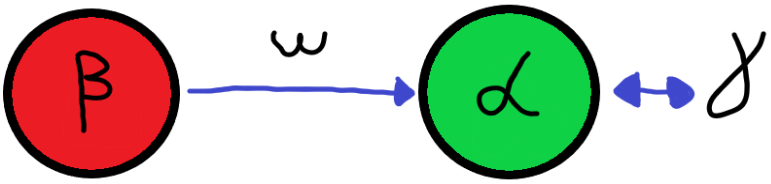
Алгоритам backpropagation је по многима један од најбитнијих алгоритама икада измишљених. Помоћу њега је могуће израчунати потребну промену параметара неуронске мреже како би се резултат израчунавања приближио жељеној вредности. Управо због тога он представља кичму готово свих неуралних мреже (међу изузецима се појављују мреже које симулирају генетичку еволуцију - на почетку насумично иницијализовани параметри успевају/не успевају да изврше задатак на основу чега се врши одабир који гени ће бити прослеђени у следећу генерацију где се врше мале мутације које доводе до разноврсности).

Да бисмо објаснили како овај алгоритам функционише прво је потребно дефинисати његов циљ. Циљ било каквог учења представља смањење грешке, у потпуности, уколико је то могуће (приликом класификовања слика објеката то би подразумевало разврставање слика у групе тако да што мање слика заврши у погрешној категорији). Битно би било напоменути да није увек могуће остварити потпуну прецизност, управо због тога потребно је дефинисати такозвано цену неуронске мреже. Ова функција зависи од самих параметара неуронске мреже и може се дефинисати на многе начине па ће се даље мислити на дефиницију коришћену у овом раду.

$$C = \sum_{i=0}^{4} (A_i - y_i)^2$$

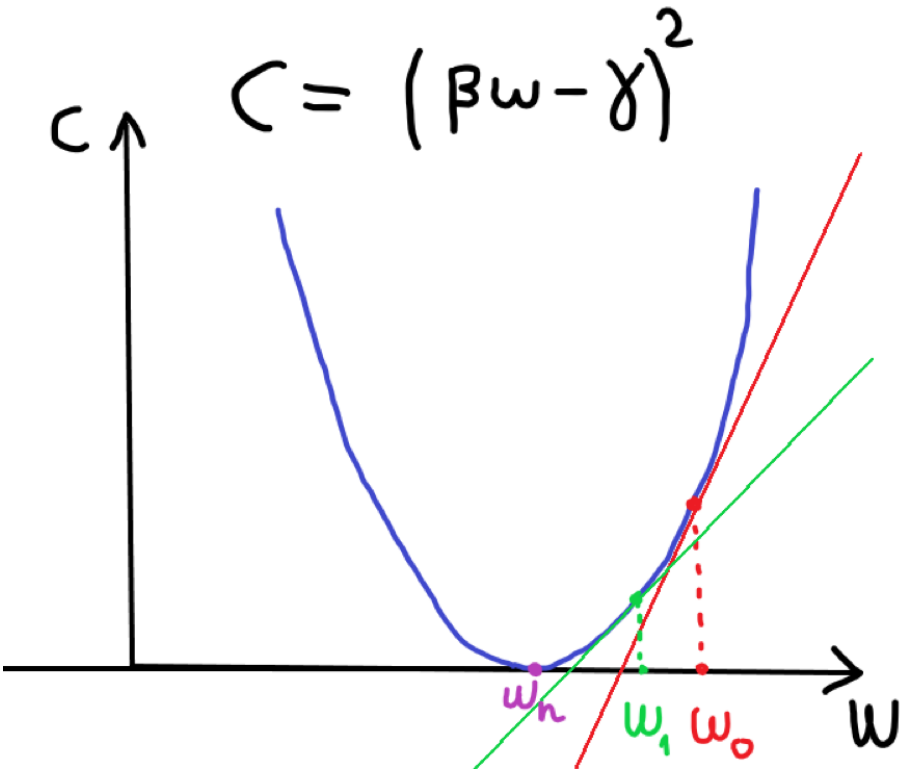
Слика 6.1 Дефиниција функција цене

Где први члан заграде представља активацију неурона последњег слоја са индексом i а други жељену вредност тог параметра (уколико би улазни податак мреже била слика кућице жељена вредност десног параметра са индексом $i =$ 4 би била једнака 10 док би остале вредности тог члана биле једнаке 0). Да бисмо размотрили како треба променити параметре ове неуронске мреже посматрајмо једну појединачну везу између два неурона претпоследњег и последњег слоја.



Слика 6.2
Приказ везе неурона из претпоследњег и последњег слоја мреже описане у раду

Скицирањем графика зависности цене од тежинског фактора можемо уочити нешто скоро занимљиво колико и брилијантно.



Слика 6.3
График зависности цене од тежинског фактора.

Приметимо да се коефицијент правца тангенте на график смањује с приближавањем параметра w вредности у којој би вредност функције имала минимум. Тада се може закључити да нам знак и вредност коефицијента правца говоре у ком смеру и у којој мери је потребно померити тежински фактор како бисмо се примакли вредности у којој се достиже минимум функције. Како је коефицијент правца тангенте на график у некој тачки једнак изводу у тој тачки то се израчунавање потребне промене своди на проналажење потребног извода $-L \times \frac{dC}{dw}$, где L представља коефицијент учења. Његова улога је да спречи да промењена вредност тежинског фактора „прескочи" тражени фактор чиме би дошло до проблема познатог као „експлодирајући градијент". Проблем умирућег неурона представља директну последицу прескакања који након пар итерација може да остане заробљен у стању нуле. Имајући то на уму можемо закључити да се вредности параметара неуронске мреже мењају у зависности од њиховог доприноса цени (парцијалног извода) и коефицијента учења. Погледајмо сада имплементацију овог алгоритма у коду.

Најпре, подразумева се да је пре извршавања овог алгоритма позвана функција за пролаз унапред која је доделила вредности активацијама свих неурона. Тада се позива функција Backpropagation у следећем облику

```
for (int i = 0; i < 10; i++)
    Backpropagation(...
```

Где 10 представља број итерација (примена) ове методе. Посматрајмо сада сам дизајн функције.

```
void Backpropagation(ref NeuralNetState StateOfNN, ref double[,]
Kernel0, ref double[,] WeightL0, ref double[,] WeightL1, ref
double[,] WeightL2, ref double[,] WeightL3, ref double[,]
WeightL4, double LearningRate, double[] DesiredResult)
{
//L4
    for (int i = 0; i < 4; i++)
        for (int j = 0; j < 5; j++)…
//L3
    for (int i = 0; i < 5; i++)
        for (int j = 0; j < 4; j++)…
//L2
    for (int i = 0; i < 6; i++)
        for (int j = 0; j < 5; j++)…
//L1
    for (int i = 0; i < 7; i++)
        for (int j = 0; j < 6; j++)…
//L0
    for (int i = 0; i < 784; i++)
        for (int j = 0; j < 7; j++)…
//Kernel0
    for (int i = 0; i < 784; i++)…
ForwardPass(ref StateOfNN, Kernel0, WeightL0, WeightL1,
WeightL2, WeightL3, WeightL4);
```

Како је потребно ажурирати сваку активацију која зависи од промењеног тежинског фактора (и привржености) најједноставније је кренути са пропагацијом од краја мреже. Погледајмо како изгледа прва петља у низу.

```
//L4

    for (int i = 0; i < 4; i++)

        for (int j = 0; j < 5; j++)

        {

            WeightL4[i, j] -= 2 * LearningRate *
(StateOfNN.OutputLayer[j] - DesiredResult[j]) *
ELUDerivative(StateOfNN.OutputLayer[j]) *
StateOfNN.FourthLayer[i];

            StateOfNN.OutputBias[j] -= 2 * LearningRate *
(StateOfNN.OutputLayer[j] - DesiredResult[j]) *
ELUDerivative(StateOfNN.OutputLayer[j]);


            #region StateOfNN updating

            for (int i1 = 0; i1 < 5; i1++)

            {

                StateOfNN.OutputLayer[i1] = 0;

                for (int j1 = 0; j1 < 4; j1++)

                    StateOfNN.OutputLayer[i1] +=
StateOfNN.FourthLayer[j1] * WeightL4[j1, i1];

                StateOfNN.OutputLayer[i1] =
ELU(StateOfNN.OutputLayer[i1] + StateOfNN.OutputBias[i1]);

            }

            #endregion

        }
```

Како свака од активација унутар последњег слоја директно зависи од тежинских фактора (и привржености такође) неопходна промена се може израчунати директно, без превише муке. i представља индекс неурона из претпоследњег слоја а j индекс тежине која повезује активацију i из претпоследњег и активацију j из последњег слоја. Фактор привржености мења се на врло сличан начин, наиме, извод $\frac{\partial C}{\partial b}$ разликује се по томе што се након извода активационе функције резултат не множи са активацијом будући да је $\frac{b}{\partial b} = 1$. Након мењања промене вредности одговарајућих параметара потребно је ажурирати активације слојева који су зависили од њих. У овом случају то би био само последњи слој. Након тога било би неопходно померити параметре доњих слојева.

```
//L3
    for (int i = 0; i < 5; i++)
        for (int j = 0; j < 4; j++)
        {
            double temp = 0;
            for (int k = 0; k < 5; k++)
                temp += (StateOfNN.OutputLayer[k] -
DesiredResult[k]) * ELUDerivative(StateOfNN.OutputLayer[k]);


            WeightL3[i, j] -= 2 * LearningRate * temp *
StateOfNN.ThirdLayer[i] *
ELUDerivative(StateOfNN.FourthLayer[j]);
            StateOfNN.FourthBias[j] -= 2 * LearningRate * temp *
ELUDerivative(StateOfNN.FourthLayer[j]);
```

```
            #region StateOfNN updating
            for (int i1 = 0; i1 < 4; i1++)
            {
                StateOfNN.FourthLayer[i1] = 0;
                for (int j1 = 0; j1 < 5; j1++)
                    StateOfNN.FourthLayer[i1] +=
StateOfNN.ThirdLayer[j1] * WeightL3[j1, i1];
                StateOfNN.FourthLayer[i1] =
ELU(StateOfNN.FourthLayer[i1] + StateOfNN.FourthBias[i1]);
            }
            for (int i1 = 0; i1 < 5; i1++)
            {
                StateOfNN.OutputLayer[i1] = 0;
                for (int j1 = 0; j1 < 4; j1++)
                    StateOfNN.OutputLayer[i1] +=
StateOfNN.FourthLayer[j1] * WeightL4[j1, i1];
                StateOfNN.OutputLayer[i1] =
ELU(StateOfNN.OutputLayer[i1] + StateOfNN.OutputBias[i1]);
            }
            #endregion
        }
```

Слично претходном ажурирању, потребно је израчунати допринос тежина и привржености цени неуронске мреже (путем парцијалних извода). Прича се, за разлику од раније, компликује по томе што цена не зависи директно од ових параметара. Како сваки тежински фактор утиче на активацију једног неурона директно потребно је одредити утицај те активације на дубље слојеве неуронске мреже. Применом правила диференцирања сложених функција добија се да је парцијални извод $\frac{\partial C}{\partial w}$ једнак производу суме парцијалних извода $\frac{\partial C}{\partial A}$ и $\frac{\partial A}{\partial w}$. Тада је тривијално (неистина) направити петљу која итеративно рачуна суму парцијалних извода и њен резултат помножити одговарајућим коефицијентима и члановима приликом померања вредности одговарајућих чланова.

Из претходног се може закључити да је за ажурирање слојева неуронске мреже ближих улазном слоју потребно све више и више времена због потребе за израчунавањем парцијалних извода претходних слојева што додаје на сложености. Ово је и више него очигледно када се узме у обзир да једна итерација кроз овај алгоритам без последње, најзахтевније петље за подешавање тежинских фактора конволуцијског филтера у просеку омогућава програму да се изврши око 50 пута брже.

7. Закључак

Применом алгоритма backpropagation, неуронска мрежа описана у овом раду успешно је истренирана да препознаје и исправно класификује две различите слике кућица и две различите сличице бродића. Ипак, пут до овог резултата био је поплочан многим тешкоћама. Наиме, како се алгоритмом за тренирање проналази локални минимум функције цене није загарантовано да је тај минимум упоредив са глобалним минимумом, те је за добре резултате, поред оптимизованих алгоритама, неопходна и велика количина среће при одабиру почетних вредности параметара. Поред тога, само тренирање неуронских мрежа поставља врло реална временска ограничења будући да једна потпуна итерација (укључујући тренирање конволуцијског кернела) траје у просеку 5 секунди. Једно од решења овог проблема би била имплементација наведених алгоритама за пропагирање напред и назад у Nvidi-ном CUDA Framework-у који омогућава паралелно извршавање израчунавања на CUDA језгрима графичких картица, међутим, таква имплементација није разматрана у оквиру овог рада.

8. Литература

But what is a neural network? | Chapter 1, Deep learning

Gradient descent, how neural networks learn | Chapter 2, Deep learning

What is backpropagation really doing? | Chapter 3, Deep learning

Backpropagation calculus | Chapter 4, Deep learning

https://en.wikipedia.org/wiki/Convolutional_neural_network

https://en.wikipedia.org/wiki/Neural_network

The Absolutely Simplest Neural Network Backpropagation Example

Building a neural network FROM SCRATCH (no Tensorflow/Pytorch, just numpy & math)

Why Neural Networks can learn (almost) anything

Backpropagation in Convolutional Neural Networks from Scratch | The Weights

Neural Network Backpropagation Basics For Dummies

https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6

https://en.wikipedia.org/wiki/Activation_function

https://en.wikipedia.org/wiki/Perceptron

https://en.wikipedia.org/wiki/Generative_adversarial_network