

Факултет по математика и информатика

Катедра „Компютърна информатика“

Дипломна работа

На тема

Разпознаване на емоции на лица

**Ръководител**

доц. В. Симеонова

**Дипломант**

Щилиян Александров Узунов

ФН: 24676

Съдържание

[1 Увод 4](#_Toc44117594)

[1.1 Цел на дипломната работа 4](#_Toc44117595)

[1.2 Задачи 4](#_Toc44117596)

[1.3 Облекчаващи условия 5](#_Toc44117597)

[1.4 Използвани съкращения и термини в текста 5](#_Toc44117598)

[2 Преглед на използваните научни методи 6](#_Toc44117599)

[2.1 Дефиниция за „Разпознаване на емоции“ 6](#_Toc44117600)

[2.2 FACS Дефиниция 7](#_Toc44117601)

[2.3 Конволюционни невронни мрежи 10](#_Toc44117602)

[2.3.1 Дефиниция 10](#_Toc44117603)

[2.3.2 Градивни блокове 11](#_Toc44117604)

[2.4 Обучаващ алгоритъм за невронни мрежи - Backpropagation 16](#_Toc44117605)

[2.5 Машини на поддържащите вектори (SVM) 18](#_Toc44117606)

[2.6 Метрики за валидация 21](#_Toc44117607)

[2.6.1 Точност на класификация (accuracy) 21](#_Toc44117608)

[2.6.2 Матрица на грешките (confusion matrix) 21](#_Toc44117609)

[2.6.3 F1 резултат 22](#_Toc44117610)

[2.6.4 Средна абсолютна грешка 23](#_Toc44117611)

[2.6.5 Средна квадратична грешка 24](#_Toc44117612)

[3 Използвано множество от данни (Dataset) 24](#_Toc44117613)

[4 Имплементация и архитектура на системата 27](#_Toc44117614)

[4.1 Използвани технологии 28](#_Toc44117615)

[4.2 Използван хардуер 29](#_Toc44117616)

[4.3 Предварителна обработка на данните 30](#_Toc44117617)

[4.4 Описание на Модел 1 31](#_Toc44117618)

[4.5 Описание на Модел 2 33](#_Toc44117619)

[4.6 Обучение на модел 1 34](#_Toc44117620)

[4.7 Обучение на модел 2 36](#_Toc44117621)

[4.8 Сравнителен анализ между двата модела 39](#_Toc44117622)

[5 Резултати от работата на системата 42](#_Toc44117623)

[5.1 Резултати на Модел 1 42](#_Toc44117624)

[5.2 Резултати на Модел 2 43](#_Toc44117625)

[5.3 Резултати от използването на двата модела с камера 46](#_Toc44117626)

[6 Инструкции за инсталация и употреба на приложението 47](#_Toc44117627)

[6.1 Описание на файловата структура 47](#_Toc44117628)

[6.2 Нужен софтуер и хардуер 48](#_Toc44117629)

[6.3 Описание на потребителският интерфейс 48](#_Toc44117630)

[7 Възможности за бъдещо развитие 49](#_Toc44117631)

[8 Използвана литература 50](#_Toc44117632)

[9 Списък с всички фигури 52](#_Toc44117633)

# Увод

Автоматичното засичане на лицеви изображение се превръща във все по важна зона за научни изследвания. Задачата включва в себе си подзадачи свързани със компютърното виждане, машинно саомообучение и поведенчески науки, и намира много приложения в областта на сигурността, взаимоотношението човек-компютър, сигурност на шофьори, и здраве опазване. През изминалите години има направен голям напредък в областта, с интерес към разпознаването на лица които не са позирали, и са поставени във реалистични среди, или позиранне от много различни ъгли.

В същността си задачата е класификационен проблем със входен параметър снимка, и изход един от няколко етикета, които показват емоцията на подадената снимка. Емоциите над които се фокусираме в тази дипломна работа са 7 – ярост, отвръщение, страх, щастие, тъга, изненада и презрение. Тези Емоции се засичат на базата освен на входна снимка и на подадени опорни точки (68 на брой, всяка със х, у координати) и информация за FACS актиавции, на база на опорните точки.

## Цел на дипломната работа

Целта на настоящата дипломна работа е изграждане на система, която да може да класифицира емоцията на подадена снимка. Ще бъдат разгледани два подхода за имплементация, и ще бъде направено сравнение на резултатите между тях.

## Задачи

В настоящата дипломна работа ще бъдат изградени два модела за класификация на емоция от снимка. Първият модел ще бъде съставен само от конволюционна невронна мрежа (CNN) и ще прави класификация от край до край, т.е. вход – снимка, изход – клас на емоцията.

Вторият модел ще бъде изграден от конволюционна невронна мрежа, която да намира 68 на брой опорни точки, които след това ще бъдат подадени на машина от поддържащи вектори, която след това ще прави крайната класификация.

Като последна задаче ще бъден направен сравнителен анализ между двата модела.

## Облекчаващи условия

За програмен език е използван python. Използвани са готови имплементации на слоеве от невронни мрежи и класификационни модели на SVM от библиотеките keras, tensorflow и scikit learn. За визуализация на резултатите са използвани готови алгоритми от библиотеката matplotlib.

## Използвани съкращения и термини в текста

|  |  |
| --- | --- |
| Термин | значение |
| facs | Facial Acting Coding System – Система за кодиране на лицевите изражения на базата на активирани мускули |
| au | Аction Unit – единица на активация във FACS системата |
| cnn | Convolutional neural network – конволюционна невронна мрежа, използват се за класификация на снимки |
| Relu / softmax | Вид активационна функция използвана за невроните на мрежата |
| backpropagation | Алгоритъм за обучение на невронни мрежи с обратно разпространение на грешката |
| svm | Support Vector Machine – машина на поддържащите вектори е тип класификационен модел |
| Dropout | Вид слой във невронната мрежа. Използва се за налагане на регуларизация |
| kernel | Ядро, използва се като параметър при SVM моделите |
| accuracy / precision / recall / МАЕ / MSE | Всички тези съкращения съответстват на различни метрики за оценка на даден модел |

# Преглед на използваните научни методи

## Дефиниция за „Разпознаване на емоции“

Разпознаване на емоции е процесъс по идентифициране на човешката емоция. Хората често варират в тяхната точност по разпознаване на емоциите на други хора. Разпознаването на емоция използвайки технологии е сравнително нова област на науката. За момента, този проблем се решава най-добре ако човешкото лице бива заснемано от няколко позиции, използвайки видео и анотации от на лицата от професионалисти.

Точността на разпознаването на емоции се увеличава когато комбинира различни източници на човешките изражение, като текст, аудио или видео. Различни видове емоции могат да бъдат засечени чрез интеграцията на информация от лицеви изражения, движения на тялото и жестове и реч. Съществуващите подходи във разпознаването на емоции могат да бъдат разделени образно казано на три основни категории: техники базирани на знания, статистически методи или хибридни подходи.

**Техниките базирани на знания** използват знания за семантичните и синтактични характеристики, с цел да определят конкретни видове емоция. В този подход често се използват ресурси базирани на знания по време на класификацията на емоцията от типа на WordNet, SenticNet, ConceptNet, EmotiNet и други. Едно от предимствата на този подход е достъпността която идва със широката наличност на такива ресурси (базирани на знания). От друга страна недостатък на този подход е невъзможността му да се справя със концептуални нюанси и сложни лигвистични правила.

Подходите базирани на знания могат да бъдат класифицирани в 2 категории: базирани на речници и базирани на корпус от документи. Тези които са базирани на речници намират думи които показват мнение или емоция в речник и търсят за техни синоними и антоними за разпиряване на началният списък от възможни емоции. Корпус базираните подходи започват със стартов списък от думи които изразяват мнение или емоци, и разширяват базата намирайки други думи със контекстно специфични характеристики в по-голям корпус. Докато подходите базирани на корпос от документи взимат предвид контекста, те все още имах проблеми със точността в различните домейни тъй като дума в един домейм може да има различна ориентация и значение от това в друг домейн.

**Статистически методи** често включват използването на различни алгоритми за машинно самообучение, при които голям набор от анотирани данни се подава на алгоритъма и по този начин системата се научава да предстазва правилните видове емоция. Алгоритмите базирани на машинно самообучение в повечето случаи предоставят по-точни класификационни резултати, сравнено с другите подходи, но едно от предизвикателствата на тези подходи е намирането на достатъчно голяма обучаваща извадка.

Някои от най-често иползваните алгоритми за решаване на този проблем включват машини на поддържащите вектор, найвен Бейсов класификатор и максимум ентропия. Дъплбокото самообучение също се използва широко в разпознаването на емоции. Добре известни архитектури които решават такива типове проблеми са конволюционните нервонни мрежи, машини за екстремно обучение и дълго кратко трайна памет (long short-term memory LSTM). Популярността на подходите за дълбоко самообучение в тази област се дължи от част и на успеваемостта на тези подходи във полетата на компютърн зрение, разпознаване на реч и обработка на естествени езици.

**Хибридни подходи** при разпознаването на емоции са комбинация от техники базирани на знания и статистически подходи, които експроатират допълващи се характеристики за твата вида техники.

## FACS Дефиниция

FACS съкратено от facial action coding system или система за закодиране на лицевите движения, е система която има за цел да класифицира лицевите движения, по начинът им на деформация на лицето. Системата е базирана на работата на шведски анатомист на име Карл-Херман Хйортсьо. По-късно е била усвоена от Поул Екман и Уолъс В. Фриесен.

Движенията на всеки индивидуален мускул на лицето са закодирани с кодове, така че да хващат и най-малките промени в лицевото изражение. Често срещано е емоциите да бъдат закодирани със FACS кодове. Този подход намира приложения в работата на психолози и художници-аниматори.

Използвайки FACS, хората които определят кои кодове са активирани могат ръчно да опишат почети всяко лицево изражение което е анатомично възможно. Това става като то се деконструктира във спеицфични единици на действие (Action Units – AU). Тези единици са независими от от какъвто и да е вид интерпретация, и могат да бъдат използвани за вземане на решения за това каква емоция е налична. Наръчникът за FACS кодове е 500 страници и съдържа единиците на действие (AU), както и интерпретацията на Екман за значението на всяка една от тях.

Въпреки че определянето на всяко от човешките изражения изисква ръчна обработка от обучени експерти, учените имат успех и със определяне на FACS кодове използвайки софтуер. CANDIDE или Artnatomy са примери за такива системи, който позволяват израженията на човешкото лице да бъдат изкуствено създадни, използвайки нужните action unit-и.

Използването на FACS е предложено и за анализи на депресия, или измерване на болка на пациенти, които не могат да изразят болка вербално.

FACS индексира лицевите изражения, но не предоставя никаква био-механична информация за степента на активация на деден мускул на лицето. Въпреки че мускулната активация не е част от FACS, основните мускули включени във лицевото изражения са добавени.

Оценка на интензитета на FACS се анотира добавяйки букви А до Е към Action Unit-а с цел - да се покаже колко „силно“ е активиран AU. Пример:

А – лека следа

В – умерено загатнато

С – маркирано

D – усилено

E – максимално

Използвайки тези букви за индикация на интензитета, Action Unit-ите изглеждат могат да изглеждат по следния начин: 1Е, 2B, 3E и т.н.

Списък от AUs и техните описания.

|  |  |  |
| --- | --- | --- |
| КОД | ИМЕ | АКТИВНИ МУСКУЛИ |
| 0 | Neutral face |  |
| 1 | Inner brow raiser | [frontalis](https://en.wikipedia.org/wiki/Frontalis_muscle) ([pars medialis](https://en.wikipedia.org/wiki/Pars_medialis)) |
| 2 | Outer brow raiser | [frontalis](https://en.wikipedia.org/wiki/Frontalis_muscle) ([pars lateralis](https://en.wikipedia.org/wiki/Pars_lateralis)) |
| 4 | Brow lowerer | [depressor glabellae](https://en.wikipedia.org/wiki/Depressor_glabellae), [depressor supercilii](https://en.wikipedia.org/wiki/Depressor_supercilii), [corrugator supercilii](https://en.wikipedia.org/wiki/Corrugator_supercilii) |
| 5 | Upper lid raiser | [levator palpebrae superioris](https://en.wikipedia.org/wiki/Levator_palpebrae_superioris), [superior tarsal muscle](https://en.wikipedia.org/wiki/Superior_tarsal_muscle) |
| 6 | Cheek raiser | [orbicularis oculi](https://en.wikipedia.org/wiki/Orbicularis_oculi) ([pars orbitalis](https://en.wikipedia.org/wiki/Orbital_part_of_frontal_bone)) |
| 7 | Lid tightener | [orbicularis oculi](https://en.wikipedia.org/wiki/Orbicularis_oculi) ([pars palpebralis](https://en.wikipedia.org/wiki/Pars_palpebralis)) |
| 8 | Lips toward each other | [orbicularis oris](https://en.wikipedia.org/wiki/Orbicularis_oris) |
| 9 | Nose wrinkler | [levator labii superioris alaeque nasi](https://en.wikipedia.org/wiki/Levator_labii_superioris_alaeque_nasi) |
| 10 | Upper lip raiser | [levator labii superioris](https://en.wikipedia.org/wiki/Levator_labii_superioris), [caput infraorbitalis](https://en.wikipedia.org/wiki/Levator_labii_superioris) |
| 11 | Nasolabial deepener | [zygomaticus minor](https://en.wikipedia.org/wiki/Zygomaticus_minor) |
| 12 | Lip corner puller | [zygomaticus major](https://en.wikipedia.org/wiki/Zygomaticus_major) |
| 13 | Sharp lip puller | [levator anguli oris](https://en.wikipedia.org/wiki/Levator_anguli_oris) (also known as [caninus](https://en.wikipedia.org/wiki/Caninus" \o "Caninus)) |
| 14 | Dimpler | [buccinator](https://en.wikipedia.org/wiki/Buccinator) |
| 15 | Lip corner depressor | [depressor anguli oris](https://en.wikipedia.org/wiki/Depressor_anguli_oris) (also known as [triangularis](https://en.wikipedia.org/wiki/Triangularis)) |
| 16 | Lower lip depressor | [depressor labii inferioris](https://en.wikipedia.org/wiki/Depressor_labii_inferioris) |
| 17 | Chin raiser | [mentalis](https://en.wikipedia.org/wiki/Mentalis) |
| 18 | Lip pucker | [incisivii labii superioris](https://en.wikipedia.org/wiki/Incisivii_labii_superioris) and [incisivii labii inferioris](https://en.wikipedia.org/wiki/Incisivii_labii_inferioris" \o "Incisivii labii inferioris) |
| 19 | Tongue show |  |
| 20 | Lip stretcher | [risorius](https://en.wikipedia.org/wiki/Risorius) w/ [platysma](https://en.wikipedia.org/wiki/Platysma) |
| 21 | Neck tightener | [platysma](https://en.wikipedia.org/wiki/Platysma) |
| 22 | Lip funneler | [orbicularis oris](https://en.wikipedia.org/wiki/Orbicularis_oris) |
| 23 | Lip tightener | [orbicularis oris](https://en.wikipedia.org/wiki/Orbicularis_oris) |
| 24 | Lip pressor | [orbicularis oris](https://en.wikipedia.org/wiki/Orbicularis_oris) |
| 25 | Lips part | [depressor labii inferioris](https://en.wikipedia.org/wiki/Depressor_labii_inferioris), or relaxation of [mentalis](https://en.wikipedia.org/wiki/Mentalis) or [orbicularis oris](https://en.wikipedia.org/wiki/Orbicularis_oris) |
| 26 | Jaw drop | [masseter](https://en.wikipedia.org/wiki/Masseter); relaxed [temporalis](https://en.wikipedia.org/wiki/Temporalis) and [internal pterygoid](https://en.wikipedia.org/wiki/Medial_pterygoid_muscle) |
| 27 | Mouth stretch | [pterygoids](https://en.wikipedia.org/wiki/Pterygoid_bone), [digastric](https://en.wikipedia.org/wiki/Digastric) |
| 28 | Lip suck | [orbicularis oris](https://en.wikipedia.org/wiki/Orbicularis_oris) |

Примери за закодиране на емоции:

|  |  |
| --- | --- |
| Емоция | Списък от AUs |
| Гняв | 4+5+7+23 |
| Презрение | R12A+R14A |
| Отвръщение | 9+15+16 |
| Страх | 1+2+4+5+7+20+26 |
| Щастие | 6+12 |
| Тъга | 1+4+15 |
| Изненада | 1+2+5B+26 |

Пример за разчитане на таблицата – за да бъде класифицирана дадена емоция като „Гняв“ е нужно да бъдат активирани следните 4 единици на действия – 4 5 7 23, тоест да бъде занижена вътрешната част на веждата (4), да бъде повдигнати горните клепач (5), клепачите да бъдат стегнати (7), да бъдат стегнати устните (23).

## Конволюционни невронни мрежи

Невронните мрежи са множество от алгоритми, които са моделирани по подобие на човешкият мозък, и се използват за разпознаване на шаблони. Използват се във класификационни и регресивни проблеми, шаблоните които те разпознават са цифрови, съдържащи се във вектори, във които трябва да се преведе всякакъв вид данни, от реалния свят – снимки/звук/текст.

### Дефиниция

В дълбокото самообучение, конволюционна невронна мрежа (CNN) е клас дъпбока невронна мрежа, най често използвана във разпознаването на визуални изображения. Те са известни със свойствата си да игнорират изместването и големината на детайлите които търсят. Това се базира на тяхната архитектура която е базирана на споделени тегла, и пренебрегването на транслациите на различни обекти. Имат приложения във видео и синкова обработка, препоръчващи системи, класификация на снимки, анализ на медицински снимки, обработка на естествени езици, обработка на финансови времеви серии и други.

Конволщционните невронни мрежи са регуларизирани версии на многослойните персептрони. Многослоен персептрон обикновенно означава напълно свързана мрежа, т.е. всеки неврон в един слой е свъран с всички неврони в следващият слой. Тази пълна свързаност на мрежите ги прави лесни за пренагаждане към данните. Типични начини за регуларизация включват някаква форма на загуба на теглата, или деактивация на някой от невроните. CNN има различен подход към регуларизацията, те се възползват от йерархичните шаблони в данните и съставят по-сложни шаблони, използвайки по-малки под-шаблони. Затва по отношение на размерност и свързаност, CNN едно ниво по-ниско от персептроните.

Вдъхновението за CNNs идва от биологичните процеси, при които шаблонат на свързаност между невроните наподобява организацията на животинската зрителна кора. Индивидуалните неврони от кората отговарят на стимулации само във ограничен регион от визуалното поле, известно като рецептивно поле. Тези рецептивни полета на различни неврони частично се застъпват, така че покриват цялото зрително поле.

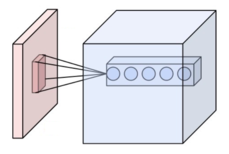
CNN нямат нужда от голяма преработка на данните, за разлика от други алгоритми за класификация на снимки. Това означава че мрежата сама научава филтри, които при традиционните алгоритми трябва да бъдат написани на ръка. Тази независимост от предишно знание, и ръчно напасване намаля нужната работа на програмиста, и позволява по-бърза разработка в дизайна на такъв тип мрежи.

### Градивни блокове

Архитектурата на CNN мрежите се формира чрез наслагване на различни слоеве които трансформират като му прилагат някаква диференцируема функция. Няколко различни типа слоеве са най-често използвани. По-долу са описани най-често използваните:

#### Конволюционен слой

Конволюционният слой е основният градивен блок на CNN мрежата. Параметрите на слоя се състоят във набор научаеми филтри (или Kernels-ядро), които имат малко поле на рецептивност, но обикалят по целият поток от входни данни. Повреме на предаването на информация напред, всеки филтър се прилага по цялата дължина и ширина на входните данни, и накрая изгражда двузимерна карта с активации на този филтър. Като резултат, мрежата научава филтри които се активират когато бъдат засечени някакви характерни форми във входните данни.



Фигура ‑Невроните на конволюционен слой(в синьо), свързани към съответстващоот им рецептивно поле(червено)

Когато работим със входни данни със големи измерения, каквито са снимките, не е практично да свързваме всички неврони към тези в предишният слой, защото такава мрежова архитектура не взема под предвид пространствената структура на данните. Конволюционните мрежи адресират този проблем като налагат локална свързаност която е ограничена между невроните от съседните слоеве – всеки неврон е свързан само със малък регион от входни данни.

Това нещо може да бъде контролирано чрез хиперпараметър за слоят, наречен рецептивно поле на неврона. Връзките са локални по пространство, но винаги обхващат цялата площ на входните данни.

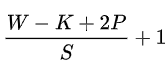


Фигура ‑Типична CNN архитектура

Три хиперпараметъра контролират размерът на изходните данни, които преминават през конволюционният слой – stride, depth и zero-padding.

* Stride (крачка) – кнтролира колко колони да бъдат алокирани във изходният формат. Когато този параметър е със стойност 1, тогава местим филтрите със 1 пиксел на всяка крачка. Това води до силно наслагване на рецептивните полета и също до изходни данни с голям обем. Когато параметърат е 2, тогава филтрите прескачат със 2 пиксела на крачка и т.н. Съответно – когато крачката е S > 0, филтърат прескача S на брой пиксела, когато обработва входните данни. На практика параметър с големина S > 3 се използва рядко.
* Depth (дълбочина) – Контролира броят на неврони във слоят които са свързани със същият регион във входнита данни. Тези неврони се учат да се активират за различни характерни черти във входните данни. Например, ако първият слой приема необработена снимка за вход, тогава различни неврони могат да се активират за да засекат наличието на конкретни ъгли или цветове.
* Zero-padding (отстояние) – Понякога е подходящо да се поставят нули във краят на входните данни, ако форматът налага някои пиксели да не бъдат обработени. Например ако имаме входни данни с формат 100х100px и размер на филтъра – 3px, то последният пиксел никога няма да бъде обработен. В такъв случаи можем да добавим padding = 1, кото би добавило по 1 пиксел в ляво и дясно (който ще е с нулева стойност) и ще можем да обработваме винаги и последният пиксел.

За изчисляване на изходният формат на конволюционният слой, използваме тези 3 параметъра, по следният начин W = големина на филтъра, S = крачка, P = големина на нулите които да сложим по рамката (zero-padding).



Формула Изчисляване на изходния формат от конволюционен слой

#### Пулинг слой(Pooling layer)

Друга важна част от CNN е пулинга или слой за обединяване, което е форма на нелинейно намаляващо селектиране. Има няколко не линейни функции за имплементиране на пулинг, измежду които макс пулинга е най-популярен. То разделя входната снимка във множество от незастъпващи се правоъгълници, и за всеки под регион изкарва най-голямата стойност, като резултат.

Интуицията за този слой е следната – точната локация на даден входен параметър е по-малко важна, от колкото приблизителната му локация, в сравнение със други входни параметри. Това е идеята за използване на пулинг във конволюционните невронни мрежи. Пулинг слоят служи за прогресивното намаляне на размерноста на данните и по този начин да намали броят на параметрите, нужната памет и изчислителна сила за работата на модела. От тук следва и че той също ограничава да известна степен мрежата от пренагаждане. Честа практика е да се постави пулинг слой между последователни конволюционни слоеве във CNN архитектурата. Пулинг операцията също добавя още 1 форма на независимост към транслации. Пулинг слоят оперира независимо върху всяко парче от входните данните, и го мащабира съответно. Най-често срещаната форма на пулинг слой е 2х2 със крачка 2. Такъв тип операция премахва 75% от активациите на предният слой.



Формула : Пулинг операция

В този случай, всяка макс операция е върху 4 числа, и от тук идват тези 75% активации които се игнорират.

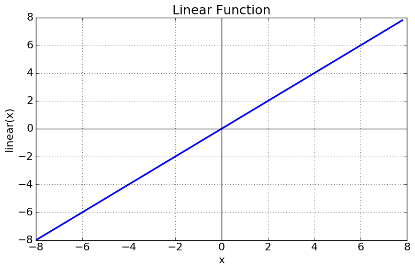
Алтернативи на макс пулинга, други фунцкии които могат да бъдат използвани са средно аритметичен пулинг или L2 нормиран пулинг. Средно аритметичният обикаля по всички входни числа, и взема средно аритметичното им, L2 прави евклидова нормализация на входните числа.

Тъй като използването на пулинг намаля драстично големината на входните данни, има тренд да се използват малки филтри, или понякога да се избягва ползването на пулинг слоеве.

#### Пълно свързан слой и видове активиращи функции

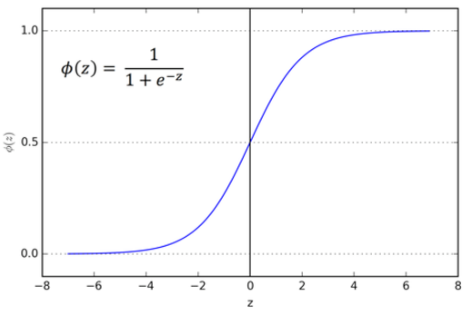
Накрая, след няколко конволюционни и пулинг слоя, логиката която научава как комбинацията от филтри разпознават даден образ се прави чрез пълно свързан слой. Невроните във пълно свързаният слой имат връзки към всеки неврон от предишния слой, точно както е във стандартните (не конволюционни) невронни мрежи. Във пълно свързаният слой имаме различни набори от активиращи функции, които можем да използваме. Ето и примери за някои активиращи функции:

* Линейна активация – f(x) = x с обхват (-∞;+∞). Това е най-простият вид активация. Може да бъде използван като изходна активация за последен слой, ако задачата която се решава е регресивен проблем.



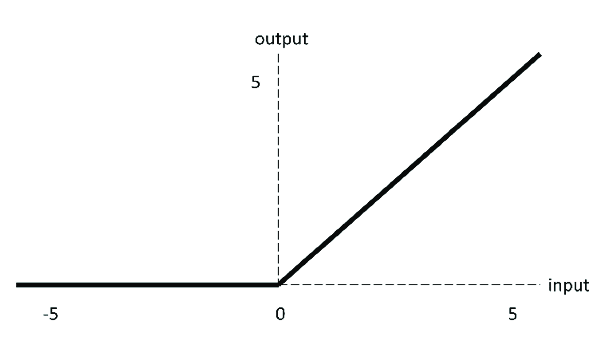
Фигура ‑ Линейна активация

* Сигмоидна активация – Основната причина поради която се използва този тип активация е защото обхватът и е от (0;1) за това много често тя се използва когато изходът трябва да бъде някаква вероятност. И тъй като вероятноста на каквото и да е е число между 0 и 1, сигмоидната активация е правилният избор за такъв тип проблем.



Фигура ‑ Сигмоидна активация

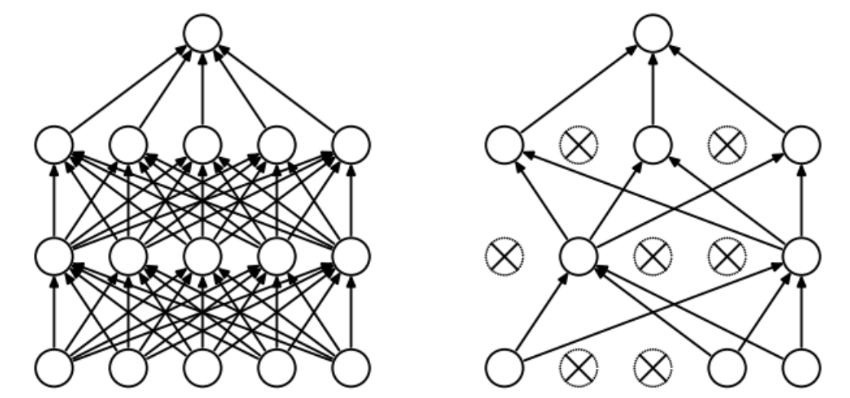
* ReLU активация – Това е най-често използваната активация за скрити слоеве в невронните мрежи. Формулата за ReLU е: f(x) = max(0, x) като предимствата и са няколко, първо има рядко разпределена активация при случайно инициализиране на невронната мрежа – около 50% от невроните се активират, второ по-добро предаване на грешката в задна посока по време на backpropagation – тази функция няма проблемът със изчезващият градиент както при сигмоидната активация например, трето – лесна за изчисление, нужните операции са само сравнение, събиране и умножение. Като недостатъци могат да се посочат следните – не е диференцируема в нулата, няма граница за максимална стойност, не е центрирана около нулата.



Фигура ‑ ReLU активация

#### Dropout слой

Честа практика е пълно свързаните слоеве да бъдат следвани от dropout слой. Този вид слой има за цел да адресира проблема със пренагаждането към обучаващите данни във невронните мрежи. Той работи като на всеки неврон от предишният слой (пълно свързаният) се добавя вероятност да бъде изпуснат в следващия стадии на изчисления. По този начин някои тегла и връзки се губят, и невроните се научават да работят самостоятелно един от друг, и да научават по-общи поведения. Пример за работата на dropout слой може наглендо да се види в следващата фигура.



Фигура ‑: Ляво - невронна мрежа без Dropout. Десно - със Dropout

## Обучаващ алгоритъм за невронни мрежи - Backpropagation

В машинното обучение, backpropagation или алгоритъм с обратно разпространение на грешката е най-широко използваният алгоритъм за обучение на невронни мрежи за обучение с учител. Backpropagation съществува като генерализиран тип алгоритми за обучение на невронни мрежи и като цяло за различни видове функции. По време на напасването на невронната мрежа към данните, backpropagation ичислява градиента на функцията която оптимизиране по отношение на теглата на мрежата, за всеки входно-изходен пример. Ефективността на този метод го прави подходящ за обучение на многослойни мрежи, използвайки градиентни методи и обновявайки теглата за минимизиране на целевата функция – градиентно спускане и стохастично градиентно спускане са най-често използваните оптимизационни похвати. Backpropagation работи изчислявайки градиента на целевата функция по отнишение на теглата следвайки верижното правило, изчислявайки градиента на всеки един слой, итерирайки назад от последният слой за избягване на излишни изчисления на междинните стойности във верижното правило.

Терминът backpropagation се отнася стрикно само за алгоритъма за изчисляван на градиента, а не за това как градиента се използва, но терминът често се използва за целият обучаващ алгоритъм, включително за това как градиента се използва. Backpropagation генерализира изчислението на градиента във делта правилото, което е еднослойната версия на backpropagation.

Преглед на параметрите и алгоритъма.

Нека Х – входен вектор от атрибути

У – изходен вектор ( за класификация изходът ще съдържа вероятности, например (0.1, 0.7, 0.2) а целевият вектор ще бъде “one-hot encoded” със стойности (0, 1, 0)

С – целева функция. При класификациони проблеми това в повечето случаи е крос ентропия, докато за регресия най-често се ползва квадратна грешка.

L – брой слоеве

Wl – Теглата между слоеве l-1 и l

fl – Активационни функции на слой l. За класификационни задачи последният слой обикновенно използва логистична функция за двоична класификация и софтмакс за много класова класификация, докато скритите слоеве използваме ReLU

По време на изчислениято на производните на алгоритъма се използват и други междинни стойности, тези стойности са описани по-долу. За целите на backpropagation-а, конкретно избраната целева функция и активационни функции нямат значение, стига техните производни да могат да бъдат изчислени по ефективен начин. Целият процес е комбинация от умножение на матрици и композиция на функции. Крайният резултат g(x) може да бъде формализиран по следният начин:



За обучаващото множество ще бъдат налични двойки от входно-изходни стойности {(xi, yi)}. За всяка от входно изходните двойки загубата за модела се изчислява като разликата между предсказаният изход g(xi) и целевият изход yi:

C(yi, g(xi))

По време на изчислението на модела, теглата са фиксирано, докато входните данни варират (и целевият изход е неизвестен), и мрежата приключва със изходния слой (той не включва целевата функция). По време на обучението, входно-изходните двойки са фиксирани, докато теглата варират, и изчисленията приключват със целевата функция.

Backpropagation изчислява градиента за фиксирани входно-изходни двойки (xi, yi), където теглата W могат да варират. Всеки индивидуален компонент на градиента  може да бъде изчислен по вреижното правило за диференциране, но изчилението по този начин за всяко отделно тегло е неефективно. Ефективното изчисление на градиентите избягвайки дублиращи се изчисления и изчисляванто на междинните стойности се прави, като се изчислява градиента на всеки слой – по-точно, градиента на претегленият вход за всеки слой – отбелязан със dl отзад напред.

Неформално начинът по който дадено тегло W влияе на целевата функция е чрез ефектът си на следващият слой, и това става линейно, dl е единствената информация която е нужна за изчисляване на градиентите на теглата на слой l, и след това можем да изчислим предишният слой dl-1 и да повторим рекурсивно. Това решава проблема със не ефективността по 2 начина. Първо, избягва се дубликирането защото когато изчислим градиента на слой l, нямаме нужда да изчисляваме наново всички производни на следващите слоеве l+1, l+2 всеки път. Второ, избягват се ненужни междинни изчисления защото на всеки етап директно се изчисляват градиентите на теглата по отношение на крайният изход (на целевата функция), отколкото ненужното изчисление на производни и на слоевете на скритите слоеве по отношение на промените в теглата .

По отношение на целевата функция, тя е такава функция която съпоставя стойности на една или повече променливи към някакво реално число, представяйки някакъв вид „цена“ асоциирана със тези стойности. За backpropagation, целевата функция изчислява разликата между изходът на мрежата и желаният изход, след като даден пример е преминал от край до край.

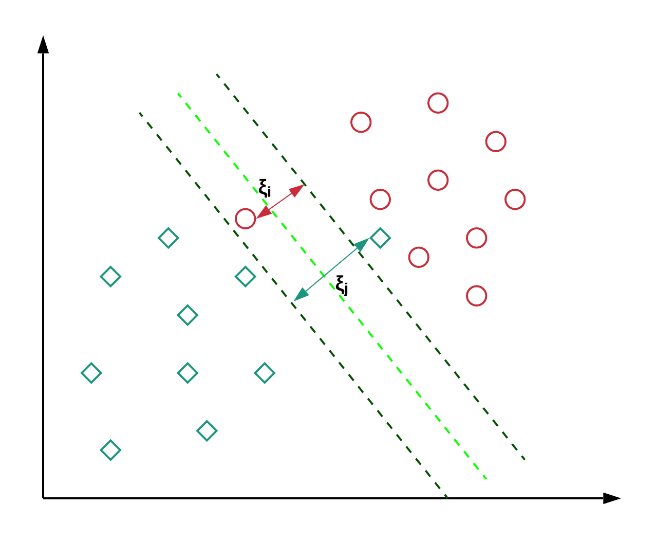
Математическите свойства които трябва да притежава целевата функция са 2, за да може да бъде използвана във backpropagation. Първо е че трябва да бъде записана като средното  върху дадени функции които изчисляват грепки Ex за n на брой обучаващи примера x. Нуждата от това свойство е че backpropagation изчислява градиента на целевата функция за един пример, който трябва да се генерализира към общата целева функция. Второто нужно свойство е тя да може да бъде записана като функция на изходите на невронната мрежа. Примери за целеви функции са крос ентропия, средна грешка и средна квадратична грешка.

Като ограничения на алгоритъма може да бъде посочено че няма гаранция че той винаги ще намери глобалният минимум на целевата функция, а само локален минимум. Също така съществуват проблеми със минаването през „плата“ на функцията – това ще рече продължителни проски участъци.

## Машини на поддържащите вектори (SVM)

В машинното самообучение, машина на поддържащите вектори(SVM) представлява метод за учение с учител който може да бъде използван за решаване на класификационни и регресивни проблеми. При даден набор от обучаващи примери, всеки от които маркиран като принадлежащ на една от 2 категории, SVM обучаващият алгоритъм изгражда модел, който присвоява на новите примери една от двете начално зададени категории, правейки го не вероятностен двуичен линеен класификатор.

SVM моделът представя примерите като точки в n-мерното пространство, така разпределени, че примерите от отделни категории са разделени от права, и отстоящото разстояние на примерите до правата е възможно най-голямо. Новите примери след това се поставят в същото пространство, и предсказването се определя в зависимост от коя страна на научената граница стои новият пример.



Фигура 2‑7: SVM разделяне на множества

В допълнение към извършването на линейна класификация, SVM може също да извършва не линейна класификация, използвайки различни видове ядра (kernels).

По-формалнo, машините на поддържащи вектори конструират хиперравнина, или множество хипер равнини във многомерното пространство, които се използват за класификация или регресия. Добро разделяне на данните се постига когато хиперравнината която има най-голямо отстояние от най-близките обучаващи точки, на който и да е от класовете. От тук следва и че колкого по-голямо е отстоянието, толкова по-малка е и грешката от генерализация на класификатора.

При подадено обучаващо множество от n на брой точки,във формата (x1, ­y1),…,(xn, yn) където yi е или 1 или -1, всеки индикиращ класът към който точката xi принадлежи. Всеки xi е p-измерен вектор от реални числа. Искаме да намерим хипер равнина с максимално отстояние, която разделя групата точки xi за които yi = 1 от групата от точки за които yi = -1. Всяка хиперравнина може да бъде изразена като множество от точки x удовлетворяващо следното уравнение:



Където w е нормалният(нормализиран) вектор към хиперравнината. Това е като нормалната форма на Хесе, с изключение че W не е задължително да бъде базисен вектор. Параметърът b/||w|| определя отстоянието на хиперравнината от центъра по отношение на нормализирания вектор w.

Ако обучаващите данни са линейно разделими, можем да изберем 2 паралелни хиерравнини които разделят двата класа данни, така че разстоянието между тях да е възможно най-голямо. Регионът който е между тези 2 хиперравнини наричаме „отстояние“, и хиперравнината която е със максимално отстоние от двата класас е тази която стои посредата на тези 2 равнини. Със нормализирано или стандартизирано множество данни, тези хиперравнини могат да бъдат описани със следните уравнения

 всичко на или над тази равнина е от клас 1

 всичко на или под тази равнина е от клас -1

Геометрично, разстоянието между тези 2 хиперравнини е 2/||w|| така че за да максимизиране разстоянието между равнините искаме да минимизираме вектора w. Разстоянието се изчислява използвайки уравнението за отстояние на точка до равнина. Също така ограничаваме данните от попадането върху отстоянието, като дабавяме следните ограничения за всяко i:





Тези ограничения могат да бъдат записани като



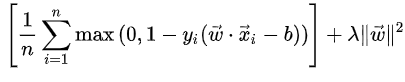
Тези ограничения налагат всяка точка да лежи на правилната страна на хиперравнината. С това оптимизационният ни проблем се првръща в минимизиране на векторът w по отношение на .

Стойностите на w и b които решават този проблем представляват нашият класификатор. Важна последица от това геометрично описание е че хиперравнината с максимално разстояние е изцяло определена от тези точки x които лежат нак-близо до нея. Тези точки x се наричат поддържащи вектори.

За разширяване на случаите в които данните не са линейно разделими се използва hinge loss функция.



Тази функция е 0 ако ограничението в (1) е изпълнено, с други думи ако x лежи на правилната страна на хиперравнината. За данни от грешната страна на равнината, стойноста на функцията е пропорционална на отстоянието от хиперравнината. От тук ние се стремим да оптимизираме:



Където параметърът λ определя нарастващото отстояние и осигурява че x е от правилната страна на хиперравнината. За това за значително малки стойности на λ вторият параметър на целевата функция ще стане пренебрежимо малък, и от тук ще се държи както във случая със линейно разделимите данни.

## Метрики за валидация

### Точност на класификация (accuracy)

Точността на класификация обикновено е това, което се има предвид, когато се използва терминът точност. То е съотношението между правилно предсказаните стойности и броя на всички направени прогнози.



Формула Точност (Accuracy)

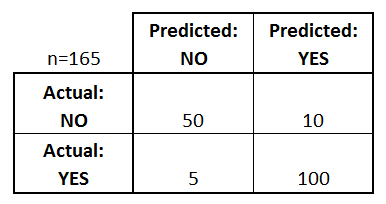
Работи добре единствено ако има равен брой обучаващи примери, принадлежащи към всеки клас. Например, в случай, че в обучаващото множество има 98% обучаващи примери от клас А и 2% от клас В, нашият модел лесно би могъл да постигне 98% тренировъчна точност просто като прогнозира, че всеки пример принадлежи към клас А.

Когато същият модел се тества с обучавао множество, в което 60% от примерите са от клас А и 40% от клас В, то тогава тренировъчната точност би спаднала до 60%. Това показва, че точността на класификация би могла да даде фалшива увереност в постигнатата точност.

Истинският проблем възниква, когато цената на грешното класифициране на обучаващите проби от непреобладаващия клас е висока. Например, ако се обработват данни за рядко, но смъртоносно заболяване, цената на грешното диагностициране на болен човек като здрав е много по-висока, отколкото ако се наложи здрави хора да бъдат подложени на допълнителни тестове.

### Матрица на грешките (confusion matrix)

Матрицата на грешките както името подсказва ни дава матрица като изход, и описва пълното поведение на модела ни. Ако предположим че имаме двуичен класификационен проблем имаме примери принадлежащи към 2 класа – yes или no. Също така имаме и наш класификатор който предсказва клас за подаден пример. При тестването на нашият модел на 165 примера получаваме следният резултат:



Фигура ‑ Пример за матрица на грешките

Тук виждаме 4 важни неша:

**Истински позитивни(TP)** – това са случаите в които ние сме предсказали YES и данните също са имали стойност YES (100 примера)

**Истински негативни(TN)** – Това са случаите в които сме предсказали NO и данните също са имали стойност NO (50 примера)

**Грешно позитивни(FP)** – това са случаите в които ние сме предсказали YES а всъщност е трябвало да се предскаже NO (10 примера)

**Грешно негативни(FN)** – това са случаите в които ние сме предсказали NO а всъщност е трябвало да се предскаже YES (5 примера)

Точността на матрицата ни се изчислява като се вземе стредно аритметичното на всички стойности лежащи на главният диагонал, в този случай = (TP + TN) / Всички примеи. Този тип матрица е базата за оценка на другите видове метрики.

### F1 резултат

F1 резултатът е хармоничното средно между precission и recall. Стойностите на F1 са в интервала [0, 1]. То казва колко прецизен е класификаторът (колко входни данни класифицира коректно) и също така колко устойчив е (не пропуска оглям брой примери).

Висок precision но нисък recall, дава голяма точност, но след това пропуска голям номер от примери които са трудни за класифициране. Колкото по-голям е F1 резултатът, толкова по-добре се справя моделът ни. Тази метрика може да ес изрази по следния начин:



Формула F1 резултат

**Precision –** Наричаме броят на кореткните поситивни резултати, разделени на броят на позитивните резултати предсказани от класификатора.



Формула Precision

**Recall** – Наричаме броят н коретните положителни резултати, разделен на номера на всички релевантни примери (всички които е трябвало да бъдат идентифицирани като позитивни).



Формула Recall

### Средна абсолютна грешка

Средната абсолютна грешка е средното между разликата в оригиналните стойности и предсказаните стойности. Този вид метрика се използва при регресивни проблеми. То ни дава мярка за това колко на дълече са отишли предсказанте стойности от реално желаните. Този тип грешка обаче не ни дава идея за посоката на грешката, т.е. дали сме предсказали по-ниска стойност или по-висока стойност. Математически се представя така:



Формула Средна абсолютна грешка

### Средна квадратична грешка

Този тип грешка е подобна на средно абсолютната грешка, разликата е че квадратичната грешка взема средното на квадратите на разликата между оригиналните и предсказаните стойности. Предимството е че можем по-лесно да изчислим градиента, докато при средната абсолютна грешка са ни нужни някой инструменти от линейното програмиране за да се справим с проблема. Тъй като вземаме квадратът на грешката, ефекта на голямата грешка става по голям, от този на малката грешка, и от тук моделът се фокусира повече върху поправянето на големите грешки.



Формула Средна квадратична грешка

# Използвано множество от данни (Dataset)

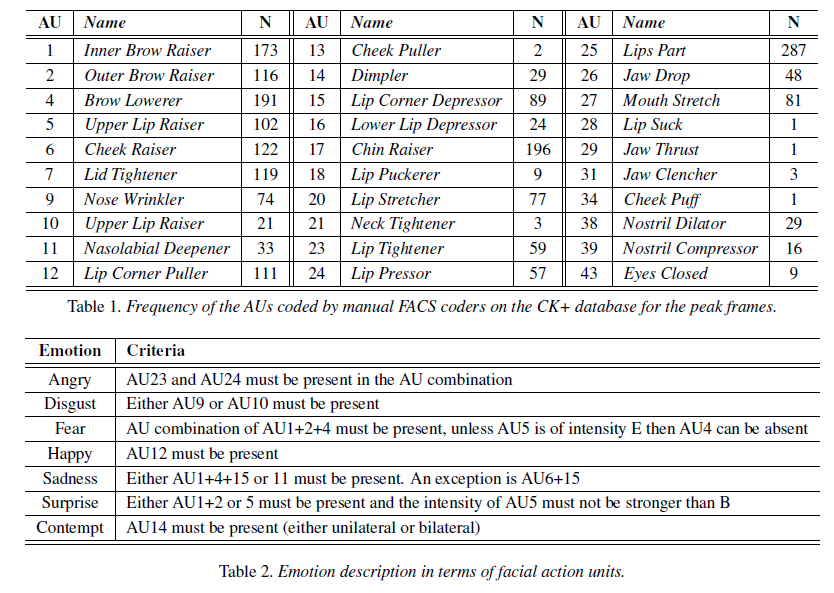
През 2000 година Cohn-Kanade dataset-a е съставен с цел промотирането на изследвания в областта на автоматичното разпознаване на лицеви изображения. От токава CK базата става една от най-широко използваните бази за сравнение за алгоритми в тази сфера.

Автоматичното засичане на лицеви изражения се превръща в сфера с нарастващо значение. Проблемът включва компютърно зрение, машинно само обучение и поведенчески науки, и може да бъде използвано на много места, като например сигурност, интеракция човек-компютър, сигурност на шофьорите, и здраве опазване.

В използваната диструбуция CK базата съдържа 593 сесии на 123 човека. Всяка сесия съдържа снимки, като стартовата снимка е във неутрално изражение, а последната снимка – в пиково изражение. Пиковият кадър е анотиран със FACS (facial action coding system).

Лицевото поведения на 210 възрастни бива записано използвайки синхронизирани камери. Участниците са на възраст от 18 до 50 години, 69% жени, 81% Евро-Американци, 13% Афро-Американци, 6% други етнически групи. Участниците са инструктирани от човекът провеждащ експеримента да извършат серия от 23 лицеви изображения. Тези включват единични FACS маркери, или комбинация от FACS маркери. Снимките са във формат 640х480рх със 8bit черно-бяла скала.

За всичките 593 сесии, има предоставено FACS кодиране, на пиковият кадър. Приблизително 15% от тези сесии са кодирани от втори сертифициран FACS кодер.



Фигура ‑ Примери за Action Unit-и и критерии за емоции

В горе посочената таблица се виждат кодовете на Action Unit-ите, например 1 – вътрешната вежда е повдигната, 2 – външната вежда е повдигната, видовете емоциите (7 на брой) и нужният критерий за да бъде категоризирана дадане емоция във снимка като активна.



Фигура ‑ Примерни изражения от Cohn-Kanade dataset-a

Освен FACS кодирането, всяка снимка е анотирана със 68 на брой опорни точки (landmarks) всяка от които със X и Y координати. За пример – в долната снимка, с червени кръгчета са отбелязани 68те опорни точки.



Фигура ‑ Примери за опорни точки на дадено изражение

# Имплементация и архитектура на системата

Системата е разделена на два модела, всеки от които има задачата да класифицира дадено изображение с емоцията в него. Двата модела са с различна архитектура, и решават проблема по различен начин, но крайният резултат е един и същи. На фигурата отдолу се виждат видовете емоции които психологът проф. Робърт Плутчик описва.



Фигура ‑ Видове емоции, описани от проф. Р. Плутчик

Човешките емоции са много на брой, разделени в 2 основни типа – базови и комплексни, като не всички комплексни емоции могат да бъдат описани чрез FACS кодиране на лицеви изражения.

Настоящата система се фокусира на разпознаването на седемте базови емоции:

* (0) Гняв
* (1) Презрение
* (2) Отвращение
* (3) Страх
* (4) Радост
* (5) Тъга
* (6) Изненада

Числата от 0 до 6 индикират цифровата стойност с която ще бъде маркирана изходната стойност за всеки от моделите.

## Използвани технологии

Имплементацията на системата е със python 3.6 като спомагателни технологии се ползват библиотеките:

* scikit-learn – имплементация на основни модели
* tensorflow – библиотека за дълбоко обучение
* tensorflow-gpu – имплементация на tf алгоритмите върху GPU
* keras – имплементация на различни слоеве за невронни мрежи
* matplotlib – визуализация на данни
* NVIDIA Cuda – framework за изчисления върху GPU
* CudNN – допълнителен модул към CUDA с имплементации на невронни мрежи
* Други

Scikit-learn е избран защото тази библиотека съдържа имплементации на основни алгоритми като машина на поддържащи вектори (SVM) алгоритми за измерване на резултатите от регресивни и класификационни задачи като Mean Absolute Error, Accuracy, F1 резултат, Precision, Recall и други.

Tensorflow е безплатна библиотека с отворен код за диференциално програмиране използвайки подхода на автоматичното диференциране. Този тип диференциране, още известно като алгоритмично диференциране представлява набор от техники за численото изчисление на дадена производна от компютърна програма. Tensorflow съдържа имплементация на основни функции и слоеве на невронни мрежи, на които зависи keras.

Tensorflow-GPU е имплементация на алгоритмите от tensorflow но върху графична карта. Тъй като нужните изчисления са тежки, и отнемат много време, заместването на CPU със GPU ускори процесът на убочение многократно.

Keras предоставя имплементация на невронни мрежи със съпътстващите ги backpropagation алгоритми, целеви функции и метрики. Библиотеката предоставя унифициран програмен интерфейс (API) който е имплементиран от няколко библиотеки – Tensorflow, Theano и CNTK. В текущата система е използвана tensorflow имплементацията.

Matplotlib е библиотека за рисуване на графики използвайки обектно ориентиран програмен интерфейс.

NVIDIA CUDA (Compute Unified Device Architecture) е платформа за паралелни изчисления и програмен интерфейс модел създаден от NVIDIA. Тя позволява на софтуерни инжинери да използват видео карти които са годни за CUDA интеграция за обработка на данни с обща цел – подход който NVIDIA наричат GPGPU (General-Purpose computing on Graphics Processing Units). CUDA платформата е софтуерен слой който дава директен достъп до виртуалния набор от инструкции на видео картата.

CudNN е библиотека със GPU-ускорени примитиви за дълбоки невронни мрежи. CudNN съдържа силно оптимизирани имплементации на стандартни методи като конволюция, пулинг, нормализация и активиране на слоеве.

## Използван хардуер

Две машини бяха използвани по време на разработката.

Dell Precision 5520

CPU: Intel Core i7-7820HQ Quad Core 2.90GHz, 3.90GHz Turbo, 8MB 45W

GPU: NVIDIA Quadro M1200 with 4GB GDDR5

RAM: 32GB

Azure Cloud VM

CPU: unknown

GPU: NVIDIA Tesla k80 with 12GB GDDR5

RAM: 64GB

Използването на втора машина беше нужно, защото изчисленията по време на обучение на Модел 2 на първата машина отнемаха по 15 часа. Видео картата на първата машина нямаше достатъчно RAM и Модел 2 не се побираше в нея, и от тук се появиха 15-те часа изчисления върху CPU-то на 1вата машина. С използването на виртуална машина от Azure с по-мощна видео карта – с 3 пъти повече памет, изчисленията падната от 15 часа на 10минтуи, което направи процесът на разработка много по-бърз, и даде възможност за много повече експерименти с параметрите на модела.

## Предварителна обработка на данните

Предварителната обработка на данните за двата модела е различна. И дваата метода имат за цел да подадат на класификаторите след тях данните в по-удобен формат, който да увеличи максимално успеваемостта.

Във множеството от обучаващи данни за всяка сесия от дадена емоция за субект, има средно по 20 снимки. Първата снимка с неутрално изражение, а последната с най-голям интензитет на изражението. На фигура 4.2 може да се видят данните от 1 сесия в суров формат.

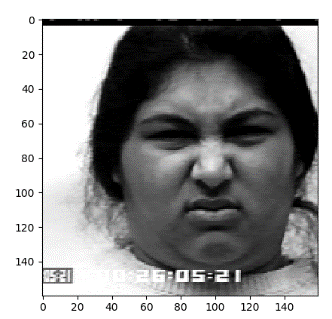


Фигура ‑ Пример за данни в дадена сесия описваща емоция от край до край

Както се вижда данните са сортирани от неутрално, към най-експресивна форма на емоцията. Тъй като Модел 1 иска да прави класификацията от край до край, на него като обучаващи данни са му нужни само най-експресивните примери, от дадена сесия – за това биват взети само последните 3 снимки.

Модел 2 е разделен на 2 под модела - CNN за предсказване на опорните точки, и SVM за извършване на крайната класификация. Данните които се подават на CNN мрежата са 8 случайно избрани снимки от дадена сесия (като вход на модела) със прилежащите им 68 опорни точки (изход). За обучение на SVM модела се използват подадените от обучаващото множество опорни точки - коректно поставени от хората създали Cohn-Kanade dataset-a. За данните на SVM-а предварителна подготовка не е нужна.

След като биват избрани снимки от сесиите, снимките се изрязват до квадратен формат. Входящия формат е 640х480, изрязват се до 480х480. Следващата стъпка е оразмеряване до 160х160 пиксела. С това обработката приключва, и данните се подават на CNN мрежите на Модел 1 и Модел 2.

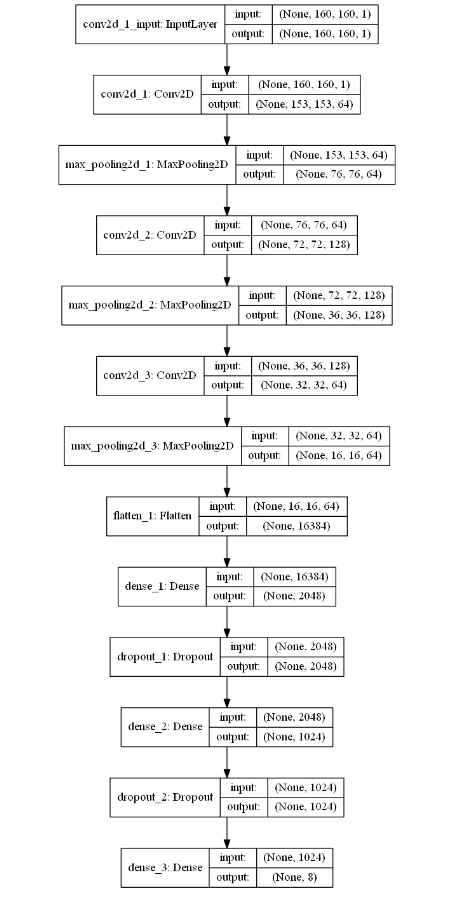
Фигура ‑3 Входен формат Фигура ‑4 Формат след обработка

## Описание на Модел 1

Модел 1 решава задачата по общо приетия наичин, в който класификацията се извършва от край до край от една CNN мрежа.

Архитектура на мрежата:

model = Sequential()  
  
model.add(Conv2D(64, kernel\_size=8, activation=**'relu'**, input\_shape=(DATA\_RESOLUTION, DATA\_RESOLUTION, 1)))  
model.add(MaxPooling2D(pool\_size=2))  
model.add(Conv2D(128, kernel\_size=5, activation=**'relu'**))  
model.add(MaxPooling2D(pool\_size=2))  
model.add(Conv2D(64, kernel\_size=5, activation=**'relu'**))  
model.add(MaxPooling2D(pool\_size=2))  
model.add(Flatten())  
  
model.add(Dense(units=2048, activation=**'relu'**))  
model.add(Dropout(rate=0.25))  
model.add(Dense(units=1024, activation=**'relu'**))  
model.add(Dropout(rate=0.25))  
model.add(Dense(units=8, activation=**'softmax'**))

Нагледно това може да бъде визуализирано със следната диаграма:

Полетата Input и Output показват трансформациите през които преминават входните данни след обработка от всеки от слоевете.

Пример: след прилагане на MaxPooling2d входните данни от размер 153х153 преминават към размерност 76х76.

Conv2d слоевете научават различни видове филтри които да прилагат върху данните. Използвани са 3 на брой conv слоя, по подобие на Alex Net Архитектурата.

MaxPooling2d слоевете изпълняват 2 функции – избират максималната стойност от подадено 2х2 поле, като по този начин мрежата не се интересува от точното разположение на дадена характеристика и така позволяват по-лесна генерализация, и също намаляват двойно размера на полето което се обработва, което спомага за по-бързата обработка на данните.

Dense слоят е стандартен пълносвързан слой, типичен за всяка невронна мрежа.

Flatten слоят сменя размерността на данните като ги прервъща от 2d в 1d.

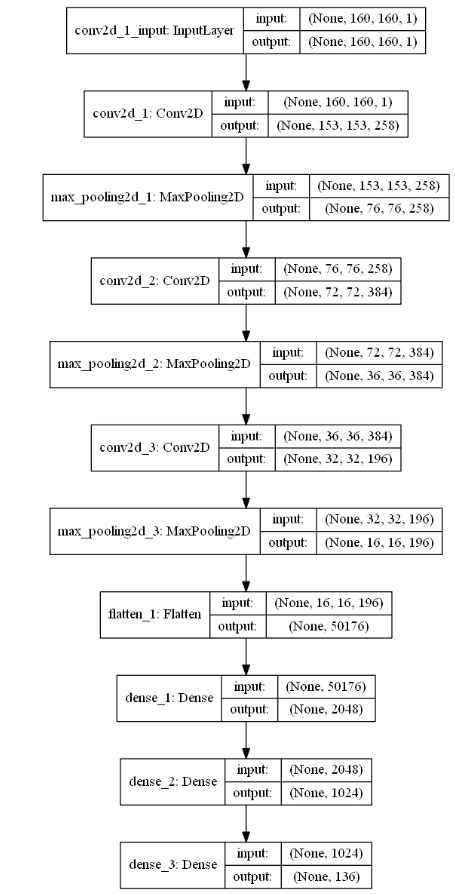
Dropout беше добавен, защото беше наблюдавано повишение на точността от около 2%. Функцията му е на случаен принцип (равен на стойността на dropout параметъра) по време на обучение да бъдат изключвани дадени неврони, за да може мрежата да генерализира по-добре, и да не се пренагажда към данните.

Фигура ‑ Архитектура на CNN 1

Изходът на мрежата са 8 неврона, всеки от които отговаря за 1 от класовете.

## Описание на Модел 2

Модел 2 решава задачата на 2 нива. Първо ниво намира опорните точки по подадена входна снимка използвайки CNN. Второ ниво намира емоцията, по подадени входни опорни точки използвайки SVM класификатор.

Архитектура на невронната мрежа:

Фигура ‑ Архитектура на CNN 2

model = Sequential()  
  
model.add(Conv2D(258, kernel\_size=8, activation=**'relu'**, input\_shape=(DATA\_RESOLUTION,DATA\_RESOLUTION, 1)))  
model.add(MaxPooling2D(pool\_size=2))  
model.add(Conv2D(384, kernel\_size=5, activation=**'relu'**))  
model.add(MaxPooling2D(pool\_size=2))  
model.add(Conv2D(196, kernel\_size=5, activation=**'relu'**))  
model.add(MaxPooling2D(pool\_size=2))  
model.add(Flatten())  
  
model.add(Dense(units=2048, activation=**'relu'**))  
model.add(Dropout(rate=0.2))  
model.add(Dense(units=1024, activation=**'relu'**))  
model.add(Dropout(rate=0.2))  
model.add(Dense(units=136, activation=**'linear'**))

Мотивацията за избор на слоеве е същата както и при Модел 1. Основна разлика между двата модела е във броят на конволюционните филтри, и в изходният слой.

Модел 2 използва 258 броя филтри на 1 ниво, 384 на 2ро и 196 на 3то ниво (сравнено с модел 1, където филтрите бяха съответно 64/128/64).

Бяха правени експрименти с по-малък брой филтри, но точността беше ниска.

Друга разлика е изходния слой – тук той има 136 на брой неврона (по 2 за всяка точка за х и у координатата) и активационната му функция е сменена – от softmax (която връща изход в интервала 0-1, към linear, което превръща проблема в регресивен).

SVM класификаторът използва следните параметри:

SVC(C=1000, kernel=**'linear'**)

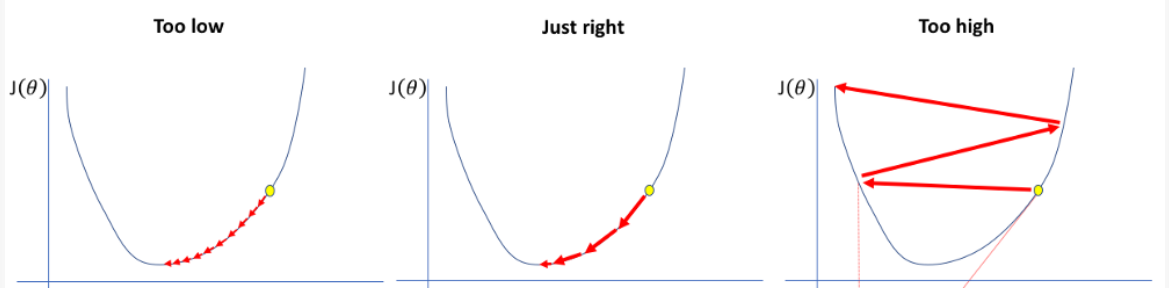
Използват се линейно ядро, и регуларизационен параметър С=1000. Точността на SVM класификатора спрямо обучаващите данни от множеството е 98%. Тази висока точност означава че до голяма степен крайната точност на Модел 2 ще зависи от това колко добре CNN мрежата която предвижда позицията на опорните точки е се справи със задачата си.

## Обучение на модел 1

За обучението на Модел 1 е използван Адам оптимизатор, със параметър на обучение 0.00001 и целева фунцкия – categorical\_crossentropy. Използваната метрика е accuracy – стандартна за класификационни задачи. Входните данни са 784 на брой снимки. Епохите на обучение са 15 на брой.

model.compile(  
 optimizer=Adam(learning\_rate=0.00001),  
 metrics=[**'accuracy'**],  
 loss=**'categorical\_crossentropy'**)

По подразбиране learning\_rate параметъра във keras библиотеката е със стойност 0.001. Изборът на правилна стойност се оказа ключов, защото стойността по подразбиране е прекалено голяма, и със нея моделът не може да намери локалните минимуми на целевата функция и ги прескача. Пример за този тип поведение може да се види на следната фигура:



Фигура ‑ Намиране на локален минимум на целева функция, в зависимост от стойността на обучаващия параметър.

Изход от процеса на обучение:

768/784 [============================>.] - ETA: 1s - loss: 3.7423 - accuracy: 0.3125  
784/784 [==============================] - 58s 74ms/step - loss: 3.7464 - accuracy: 0.3112  
Epoch 3/15  
  
 32/784 [>.............................] - ETA: 41s - loss: 2.3791 - accuracy: 0.3125  
 64/784 [=>............................] - ETA: 40s - loss: 2.3785 - accuracy: 0.3438  
  
768/784 [============================>.] - ETA: 0s - loss: 1.0945 - accuracy: 0.6276  
784/784 [==============================] - 45s 57ms/step - loss: 1.0824 - accuracy: 0.6327  
Epoch 6/15  
  
 32/784 [>.............................] - ETA: 44s - loss: 0.8730 - accuracy: 0.6250  
 64/784 [=>............................] - ETA: 42s - loss: 0.9845 - accuracy: 0.6719  
 96/784 [==>...........................] - ETA: 41s - loss: 0.8096 - accuracy: 0.7604  
  
Epoch 12/15  
  
 32/784 [>.............................] - ETA: 1:11 - loss: 0.2627 - accuracy: 0.8750  
 64/784 [=>............................] - ETA: 1:08 - loss: 0.2161 - accuracy: 0.9062  
  
768/784 [============================>.] - ETA: 1s - loss: 0.1242 - accuracy: 0.9688  
784/784 [==============================] - 53s 67ms/step - loss: 0.1256 - accuracy: 0.9681  
Epoch 15/15  
  
 32/784 [>.............................] - ETA: 51s - loss: 0.0910 - accuracy: 0.9688  
 64/784 [=>............................] - ETA: 49s - loss: 0.1029 - accuracy: 0.9688  
.....  
736/784 [===========================>..] - ETA: 3s - loss: 0.1171 - accuracy: 0.9660  
768/784 [============================>.] - ETA: 1s - loss: 0.1163 - accuracy: 0.9648  
784/784 [==============================] - 62s 79ms/step - loss: 0.1150 - accuracy: 0.9656

От изхода се вижда как на всяка от епохите точността се получава. Бяха правени и опити със повече от 15 епохи, но се оказа ненужно, защото след 13-14 епоха, точността се покачваше със 0.001%.

## Обучение на модел 2

Подобно на Модел 1 при Модел 2 се използва Adam оптимизатор, но метриката е за регресивни проблеми – средна абсолютна грешка. Основната разлика е във целевата функция. Бяха правени експерименти със 2 целеви функции – средна квадратична грешка, и собствена функция специфична за Модел 2. Оказа се че използването на собствена функция дава по-добри резултати.

model.compile(  
 optimizer=Adam(learning\_rate=0.00001),  
 metrics=[**'mae'**],  
 *#loss='mean\_squared\_error'* loss=\_custom\_loss  
)

Параметърът learning\_rate по същата мотивация както при Модел 1 беше занижен спрямо стойността по подразбиране.

Целевата функция с която беше експериментирано е следната:

low\_prio\_indecies = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16]  
normal\_indecies = [18, 19, 20, 23, 24, 25, 27,28,29,30,31,32,33,34,35,  
 37,38,40,41,43,44,46,47,50,51,52,55,56,57,58,59,60,61,62,63,64,65,66,67]  
high\_prio\_indecies = [17, 21, 22, 26, 36, 39, 42, 45, 48, 49, 53, 54]  
  
weight\_lp = 1  
weight\_normal = 2  
weight\_hp = 4  
total\_weight = len(low\_prio\_indecies) \* weight\_lp + \  
 len(normal\_indecies) \* weight\_normal + \  
 len(high\_prio\_indecies) \* weight\_hp

**def** \_custom\_loss(yTrue,yPred):  
 \_apply\_weight\_to\_tensor(yTrue, low\_prio\_indecies, weight\_lp)  
 \_apply\_weight\_to\_tensor(yPred, low\_prio\_indecies, weight\_lp)  
  
 \_apply\_weight\_to\_tensor(yTrue, normal\_indecies, weight\_normal)  
 \_apply\_weight\_to\_tensor(yPred, normal\_indecies, weight\_normal)  
  
 \_apply\_weight\_to\_tensor(yTrue, high\_prio\_indecies, weight\_hp)  
 \_apply\_weight\_to\_tensor(yPred, high\_prio\_indecies, weight\_hp)  
  
 yTrue = yTrue / total\_weight  
 yPred = yPred / total\_weight  
  
 **return** K.sum(K.square(yPred - yTrue))

**def** \_apply\_weight\_to\_tensor(input, indecies, weight):  
 **for** i **in** indecies:  
 *#x point* tf.multiply(input[i \* 2], weight)  
 *#y point* tf.multiply(input[(i \* 2) + 1], weight)

Целта на този тип целева функция е да приложи различни тегла спрямо различните точки, тъй като някои точки имат по-голямо значение при крайната класификация на емоцията. Такива точки на пример са ъгълчетата на устата, позицията на точките на веждите и ъглите на очите. Стандартната целева функция взема средно аритметично на всички 68 точки, и по този начин – тя смята всяка точка за еднакво важна, докато тази функция се опитва да придаде важност на няколко конкретни точки.

В горе описаният сегмент код индексите на точките са разделени на 3 вида – такива с нисък приоритет (low\_prio\_indecies), такива със среден приоритет (normal\_prio\_indecies) и такива със висок приоритет (high\_prio\_indecies). Съответните тегла които се прилагат за всеки вид приоритет са 1(нисък) 2(среден) и 4(висок).

С това целевата ни функция се превръща от средно квадратична към средна квадратична претеглена оценка.

**return** K.sum(K.square(yPred - yTrue))

След обучение от 25 епохи беше достигната точност от 1.79 отклонение за пиксел.

Epoch 1/25

1569/1569 [==============================] - 39s 25ms/step - loss: 393.8409 - mae: 31.7700

Epoch 2/25

1569/1569 [==============================] - 30s 19ms/step - loss: 45.9740 - mae: 11.3462

...

Epoch 8/25

1569/1569 [==============================] - 30s 19ms/step - loss: 6.5243 - mae: 4.0293

Epoch 9/25

1569/1569 [==============================] - 30s 19ms/step - loss: 5.0803 - mae: 3.5559

...

Epoch 14/25

1569/1569 [==============================] - 30s 19ms/step - loss: 2.3870 - mae: 2.4504

Epoch 15/25

...

Epoch 18/25

1569/1569 [==============================] - 31s 19ms/step - loss: 1.5594 - mae: 2.0024

Epoch 19/25

1569/1569 [==============================] - 31s 19ms/step - loss: 1.4417 - mae: 1.9316

Epoch 20/25

...

Epoch 23/25

1569/1569 [==============================] - 30s 19ms/step - loss: 1.0103 - mae: 1.6227

Epoch 24/25

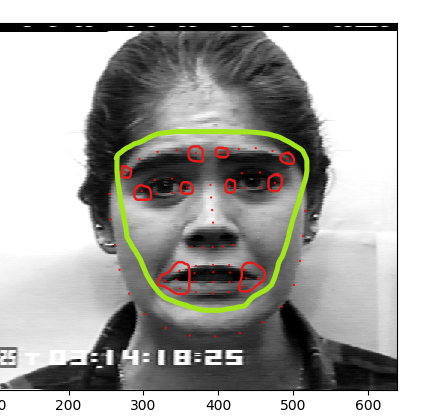
1569/1569 [==============================] - 31s 19ms/step - loss: 1.0941 - mae: 1.7174

Epoch 25/25

1569/1569 [==============================] - 30s 19ms/step - loss: 0.9343 - mae: 1.5724

393/393 [==============================] - 4s 9ms/step

Evaluation: [1.2477316626610646, 1.7943311929702759]



Фигура ‑ Примерно отбелязване на теглата по опорните точки

На фигура 4.6 се виждат кои точки към коя теглова група спадат. Всички оградени с червено спадат към групата на най-приоритетните точки. Всички които попадат във зелената зона са със нормален приоритет, а всички извен нея са с нисък приоритет.

Оценката на това коя от двете целеви функции дава по-добър резултат беше нарпавена на базата на успеваемостта на цялостната класификация.

Успеваемост целева функция MSE: 77%

Успеваемост custom функция: 81.725%

На база на тези експерименти беше нарпавен извода че функцията която прилага тегла на отделните точки е по-добра.

Втората част от обучението на Модел 2 се състои в обучаване на SVM класификатора. Обучението там е със входни данни не от CNN-а за разпознаване на опорни точки, а със опорните точки дадени като „правилни“ от обучаващото множество.

## Сравнителен анализ между двата модела

Структура на моделите:

|  |  |
| --- | --- |
| Модел 1 | Модел 2 |
| \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Layer (type) Output Shape Param #  =================================================================  conv2d\_1 (Conv2D) (None, 153, 153, 64) 4160  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  max\_pooling2d\_1 (MaxPooling2 (None, 76, 76, 64) 0  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  conv2d\_2 (Conv2D) (None, 72, 72, 128) 204928  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  max\_pooling2d\_2 (MaxPooling2 (None, 36, 36, 128) 0  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  conv2d\_3 (Conv2D) (None, 32, 32, 64) 204864  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  max\_pooling2d\_3 (MaxPooling2 (None, 16, 16, 64) 0  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  flatten\_1 (Flatten) (None, 16384) 0  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  dense\_1 (Dense) (None, 2048) 33556480  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  dropout\_1 (Dropout) (None, 2048) 0  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  dense\_2 (Dense) (None, 1024) 2098176  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  dropout\_2 (Dropout) (None, 1024) 0  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  dense\_3 (Dense) (None, 8) 8200  =================================================================  Total params: 36,076,808  Trainable params: 36,076,808  Non-trainable params: 0 | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Layer (type) Output Shape Param #  =================================================================  conv2d\_1 (Conv2D) (None, 153, 153, 258) 16770  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  max\_pooling2d\_1 (MaxPooling2 (None, 76, 76, 258) 0  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  conv2d\_2 (Conv2D) (None, 72, 72, 384) 2477184  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  max\_pooling2d\_2 (MaxPooling2 (None, 36, 36, 384) 0  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  conv2d\_3 (Conv2D) (None, 32, 32, 196) 1881796  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  max\_pooling2d\_3 (MaxPooling2 (None, 16, 16, 196) 0  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  flatten\_1 (Flatten) (None, 50176) 0  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  dense\_1 (Dense) (None, 2048) 102762496  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  dropout\_1 (Dropout) (None, 2048) 0  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  dense\_2 (Dense) (None, 1024) 2098176  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  dropout\_2 (Dropout) (None, 1024) 0  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  dense\_3 (Dense) (None, 136) 139400  =================================================================  Total params: 109,375,822  Trainable params: 109,375,822  Non-trainable params: 0 |

Като архитектура двата модела са идентични, използват се един и същи видове слоеве, подредени в една и съща последователност. Основните разлики са във броят параметри които се използват за изграждането им. Както се вижда от горната таблица Модел 2 използва 109,375,822 параметъра, докато Модел 1 използва 36,076,808 което е почети 3 пъти по-малко.

От тази разлика биха следвали различни нужди за RAM памет на модела, и по-голямо процесорно време за обработка на дадена заявка. Такива експерименти не бяха направени.

Основна разлика между двата модела е точността с която те дават за краен резултат:

Модел 1: 98.98%

Модел 2: 81.725%

Това се дължи на факта че Модел 2 решава по трудна задача – първо да намери опорни точки и след това да ги използва за класификация. В представата на Модел 1 никъде не присъстват опорните точки, и това улеснява работата му.

Освен това Модел 2 има нужда от допълнителен Модел – SVM класификаторът който да дава крайният резултат. Това допълнително би забавило действието му и повишава сложността на модела.

Очевидно от двата модела Модел 1 се справя с проблемът на класификация на изражения по-точно и с по-малко ресурси.

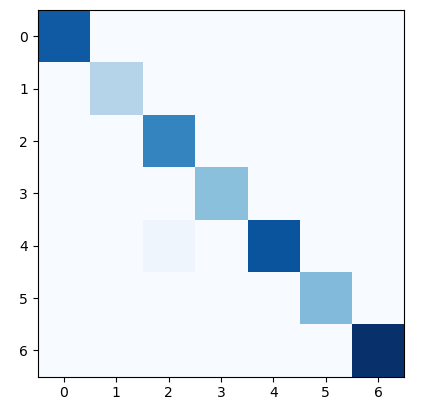
Като предимство на Модел 2 можем да кажем че намерените опорни точки ни дават допълнителни знания с които ние разполагаме, когато видим дадена снимка. В случай на нужда, тези допълнителни знания биха ни били полезни за допълнителна обработка на снимките, например 3D ротации на лицето и извършване на деформации.

# Резултати от работата на системата

## Резултати на Модел 1

Тъй като модел 1 извършва класификаця от край до край (за разлика от модел 2) за измерването на неговата точност е нужна само 1 вид класификационна метрика.

Долната фигура, е визуализация на матрицата в ляво от нея. Матрицата е с размерност 7х7 – какъвто е броят на емоциите които се опитваме да разпознаваме, по X координатата на матрицата са резултатите които нашият модел предсказва, а по У са реалните стойности от тестовото множество. Матрицата се чете по следният примерен начин – за ред 1, имаме стойности (1,0) = 0 (1,1) = 13, (1,2) = 0 и т.н. (1,6) = 0, това означава че за емоция с код 1 – гняв имаме пълно разпознаване на емоциите без грешка. Налична грешка има на ред номер 4, където (4,2) = 2, (4,4) = 37 и всички други стойности са 0. Това означава че общо всички тестови примери е имало 39 примера със код 4 – страх, като 37 от тях са били познати правилно, и 2 са били грешно класифицирани от нашият модел като 2 – презрение.



Фигура ‑ Confusion Matrix за модел 1

0:[36 0 0 0 0 0 0]

1:[ 0 13 0 0 0 0 0]

2:[ 0 0 29 0 0 0 0]

3:[ 0 0 0 18 0 0 0]

4:[ 0 0 2 0 37 0 0]

5:[ 0 0 0 0 0 19 0]

6:[ 0 0 0 0 0 0 43]

На база на горната матрица точността на нашият модел се определя по следния начин, използвайки Accuracy като:

Общ брой тестови данни: 197

Брой правилно класифицирани: 195

Точност на работа: (195/197) \* 100 = 98.98%

F1 резултатът, чиято формула може да бъде видяна в точка 2.6.3 има следната стойност:

F1 = 99.30%

На база на измерените резултати можем да заключим че Модел 1 решава класификационният проблем почети перфектно.

## Резултати на Модел 2

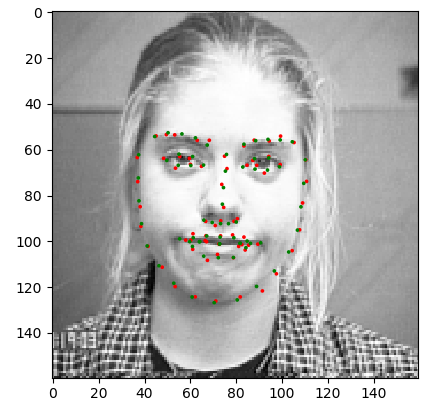
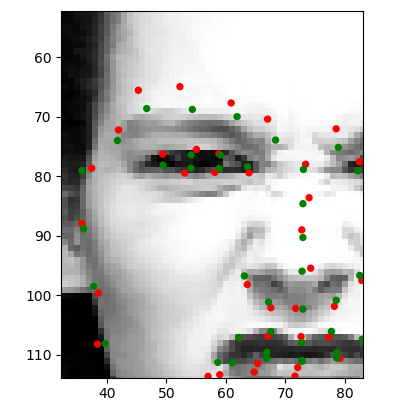
Поради архитектурата на модел 2 за измерването на работата му са нужни няколко вида метрики – такива за регресивни задачи, и такива за класификационни задачи.

За измерването на точноста на невронната мрежа която продуцира 68-те опорни точки използваме средна абсолютна грешка (описана в точка 2.6.4) която измерва средното абсолютно отстояние на всяка координата (х или у, от целевата и стойност).

Измерената средна абсолютна стойност на всеки от координатите за 197 тестови примера е:

**Mean absolute error: 1.2314**

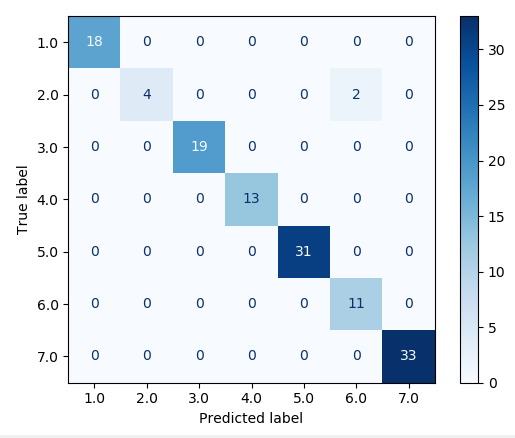
Което при размер на входните данни от 160х160px представлява 1.2314/160 = 0.007% от цялото пространство на снимката. Пример за предсказани и целеви точки може да се види на долната фигура. Със зелено са отбелязани правилните положения на точките, а със червено – тези които са били предсказани.

Фигура ‑ Резултат от предсказване на опорни точки Фигура ‑ Доближен резултат

На фигура 5.2 се вижда резултатът от предсказване на опорните точки – видимо доста близко до целевият резултат, на фигура 5.3 се виждат малките отстояния (от които идва и грешката от 1.2314) в които точките се различават.

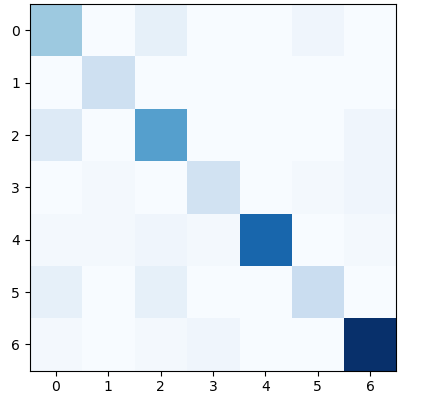
Следващата част от модела която измерваме е точността на машината на поддържащите вектори. Там при подадени входни данни – 68 на брой опорни точки (взети от обучаващото множество) се получава като резултат предсказание за емоция. Използваните метрики e точност (Accuracy). На фигура 5.4 се вижда confusion matrix на база на която са направени изчисленията.



Фигура ‑ Confusion Matrix за SVM на Модел 2

SVM Accuracy за Модел 2 = **98.47%**

Последната част която измерваме е точността на целия модел от край докрай, т.е. даден тестови пример минава през невронната мрежа за измерване на опорни точки, и след това същите тези опорни точки биват подавани на SVM модела.



Фигура ‑ Confusion Matrix за целият Модел 2

0: [18 0 4 0 0 2 0]

1: [ 0 10 0 0 0 0 0]

2: [ 6 0 27 0 0 0 2]

3: [ 0 1 0 9 0 1 2]

4: [ 1 1 2 1 38 0 1]

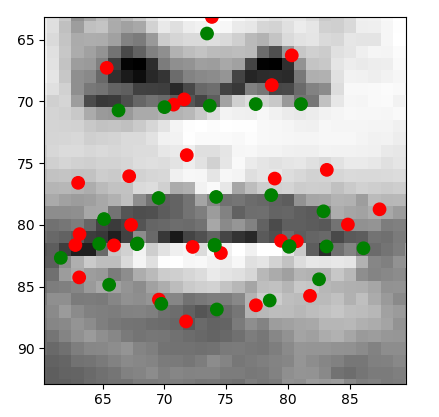
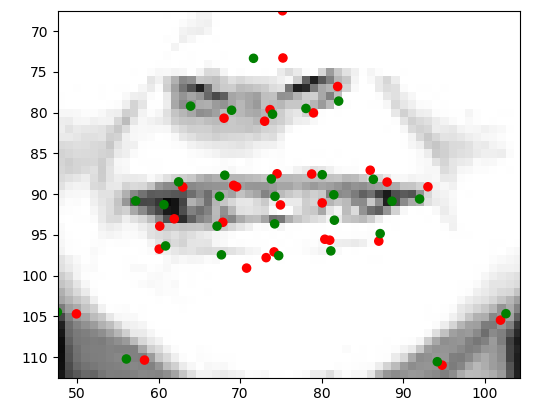
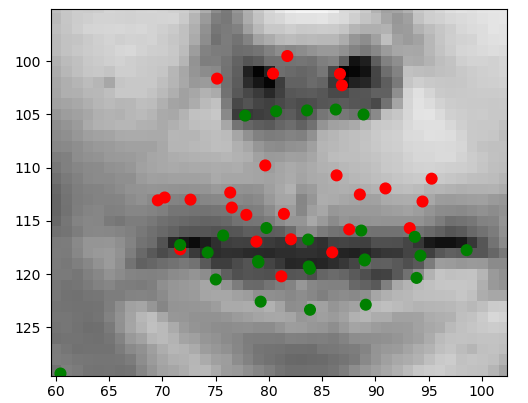
5: [ 4 0 4 0 0 11 0]

6: [ 1 0 1 2 0 0 48]

Тук както виждаме от матрицата резултатите са по-ниски, с доста повече стойности които не лежат на главният диагонал. Това индикира и повече грешки, изчислената стойност е:

Accuracy: **81.725%**

Най-вероятната причина за този резултат е не достатъчно добрата успеваемост при определянето на опорните точки. Измерената там средна грешка от 1.2314 все пак е голяма, и най-вероятно е нужна по-голяма прецизност. Това може да бъде разбрано по-интуитивно от долните фигури, където виждаме доближен резултат от предсказването на опорни точки в областта на устата.



Фигура ‑ Примери за разминаващи се опорни точки в областта на устата.

За да бъде затворена тази грешка е нужно по-продължително обучение, което в рамките на тази дипломна работа не беше направено, и ще бъде оставено като вариант за бъдещо развитие.

## Резултати от използването на двата модела с камера

И двата модела бяха тествани на разпознаването на изражения използвайки камера, като всеки кадър от камерата беше подаван за разпознаване на всеки от моделите. Измерими резултати на точността не бяха напавени, но цялостното впечетление беше че точността е много по-ниска от колкото при dataset-овете с които беше правено обучението на моделите. Най-вероятната причина за това е че във всички снимки в обучаващото множество има налични едни и същи елементи на заден фон – чисто черно или чисто бяло перде, всички снимки са заснети под един и същи ъгъл – фронтален, и е използвано едно и също осветление.

Бяха направени експерименти с 2 вида среда – такъв със случаен заден фон, при който моделите много рядко различаваха каква е реалната емоция на човека пред камерата, и такъв със черен фон, при който се забелязваше някакво подобрение, но пак по-лошо от това при обучаващото множество – най-вероятно защото позицията на главата и светлината пак остават вариращ фактор, спрямо обучаващото множество.

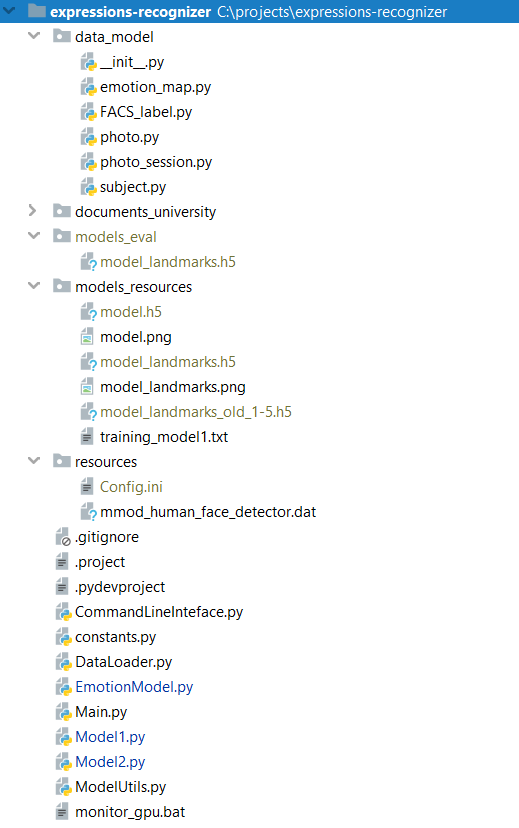
Като евентуално решение на проблема със задния фон бихме могли да проложим алгиритъм за засичане на лице, и като входна информация за моделите да се подава само изрязана част на лицето на субекта на снимката.

За проблемите със осветлението и позицията на главата ще са ни нужни по-вече на брой обучаващи данни, с различна светлина и поза. Множеството от данни Cohn-Kanade е с ограничен обхват на случаите, и е направено с експериментални цели, от тук е логично че за да бъде решен този проблем в реална среда са нужни още обучаващи данни.

# Инструкции за инсталация и употреба на приложението

Source code на проекта може да бъде изтеглен от github на адрес:  
https://github.com/ShtiliyanUzunov/expressions-recognizer.git

## Описание на файловата структура

**data\_model**

Пакетът съдържа имплементациите на всички Python класове използвани от приложението.

**documents\_university**

Пакетът съдържа всички документи приложени към текущата дипломна работа.

**models\_eval**

Това е изходна директория за всички модели след като бъдат тренирани.

**models\_resources**

Тук се съдържат готовите модели които се използват за предсказване на данните + снимки на архитектурата на всеки модел. След като бъде обучен в models\_eval даден модел, е нужно ръчно да бъде преместен във models\_resources, ако потребителя има желание да го ползва.

**resources**

Съдържа Config.ini файлът, който оказва къде на файловата система се намира Cohn-Kanade dataset-a.

**Главна директория**

Тук се намират имплементациите на моделите, командният интерфейс и методи за зареждане на данните. Файлът monitor.gpu може да се използва на Windows-ки системи (при наличие на NVIDIA видео карта) за наблюдение на работата на видео картата.

## Нужен софтуер и хардуер

Нужна е инсталация на python3.5+ версия, със инсталирани библиотеки scikit-learn, matplotlib, tensorflow, tensorflow-gpu, numpy и всички техни библиотеки от които зависят. Освен python и неговите библиотеки са нужни и NVIDIA CUDA Runtime 10.1 и CudNN64\_7.dll

Желателно е потребителят да разполага с машина с видео карта. Обучението на двата модела може да бъде стартирано от Model1.py и Model2.py, като за обучението и на двете невронни мрежи се използва tensorflow-gpu модула, който разчита че има налична видео карта. Ако няма да се пуска функционалността за обучение, разпознаването на емоции може да се изпълни и сам върху CPU-то на машината.

## Описание на потребителският интерфейс

Интерфейсът е конзолно базиран, поддържа команди за разглеждане на обучаващото множество, класифициране на данни и преглеждане на точността на всеки от моделите.

Списък с поддържани команди:

- printScores – Изкарва резултатите на emotion model-a след обучение

- exit – Изход от приложението

- listSubjects – Изкарва списък на всички субекти в обучаващото множество

- listSessions – Изкарва списък на всички сесии на подаден субект

- help – Изкарва всички възможни команди

- getEmotion – Изкарва емоцията за дадена сесия вземайки я от обучаващото множество

- predictEmotion – Изпълнява предсказване на емоция за дадена сесия

- showSession – Показва всички снимки от дадена сесия

- showPeak – Показва пиковото изражение от дадена сесия

- listFalsePredictions – Показва грешно позитивните класификации за дадена сесия

# Възможности за бъдещо развитие

Системата решава с висока успеваемост проблемите за които е предназначена, но има някои насоки в които могат да бъдат направени подобрения за вбъдеще. Това са проблемите със ниската успеваемост при разпознаване на изображения от камера, и повишаване на успеваемостта на модел 2 (която както беше упоменато във точка 2 е почети 82%).

Ниската успеваемост при разпознаване на изображения от камера се дължи най-вероятно на средата в която са заснети обучаващите примери. При почети всички картинки има наличен черен заден фон, без елементи по него и изражение на човек на преден план. При заснемане от камера във жилищна среда, назаден фон илизат всякакви предмети от бита – които внасят шум, за който мрежата не е обучена. Нормално е резултатите да бъдат по-ниски и един начин по който този проблем може да бъде преудолян е като се добавят още обучителни примери със сними от жилищна среда.

Вторни начин по който може да бъде преудолян този проблем е като се използва facial detection, или метод за засичане на позицията на главата, и бъдат изрязани всички детайли от задният фон. Това би намалило шумът и също би подобрило работата на модела при данни идващи от камера.

Що се отнася до успеваемостта на модел 2, класификацията там зависи на 2 под модела – този на невронната мрежа която намира опорните точки, и този на SVM модела, който на база на точките прави крайната класификация, е много важно първият модел да върши работат си близо до перфектното. Тя може да бъде повишена при по продължително обчение, с повече на брой епохи и повече на брой примери. По време на обучението бяха поради липса на достатъчно ресурси, обучението на първият под модел от модел 2 беше сведено до 1 час. При удвояване или утрояване на убочаващото време, цялостната успеваемост на модел 2 би била по-висока.

# Използвана литература

1. The Extended Cohn-Kanade Dataset (CK+): A complete dataset for action unit
2. Ian Goodfellow and Yoshua Bengio and Aaron Courville (2016). [Deep Learning](http://www.deeplearningbook.org/). MIT Press. p. 326.
3. ["Convolutional Neural Networks (LeNet) – DeepLearning 0.1 documentation"](http://deeplearning.net/tutorial/lenet.html). DeepLearning 0.1. LISA Lab. Retrieved 31 August 2013.
4. Krizhevsky, Alex; Sutskever, Ilya; Hinton, Geoffrey E. (2017-05-24). ["ImageNet classification with deep convolutional neural networks"](https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf) (PDF). Communications of the ACM. 60 (6): 84–90. [doi](https://en.wikipedia.org/wiki/Doi_(identifier)):[10.1145/3065386](https://doi.org/10.1145%2F3065386). [ISSN](https://en.wikipedia.org/wiki/ISSN_(identifier)) [0001-0782](https://www.worldcat.org/issn/0001-0782).
5. [Bishop, C. M.](https://en.wikipedia.org/wiki/Christopher_M._Bishop) (2006), Pattern Recognition and Machine Learning, Springer
6. Machine learning and pattern recognition "can be viewed as two facets of the same field.
7. Powers, David M W (2011). ["Evaluation: From Precision, Recall and F-Measure to ROC, Informedness, Markedness & Correlation"](http://www.flinders.edu.au/science_engineering/fms/School-CSEM/publications/tech_reps-research_artfcts/TRRA_2007.pdf) (PDF). *Journal of Machine Learning Technologies*. **2**(1): 37–63.
8. Hopfield, J. J. (1982). [*"Neural networks and physical systems with emergent collective computational abilities"*](https://www.ncbi.nlm.nih.gov/pmc/articles/PMC346238). Proc. Natl. Acad. Sci. U.S.A. **79** (8): 2554–2558. [*Bibcode*](https://en.wikipedia.org/wiki/Bibcode_(identifier)):[*1982PNAS...79.2554H*](https://ui.adsabs.harvard.edu/abs/1982PNAS...79.2554H).
9. Russell, Ingrid. [*"Neural Networks Module"*](https://web.archive.org/web/20140529155320/http:/uhaweb.hartford.edu/compsci/neural-networks-definition.html). Archived from [*the original*](http://uhaweb.hartford.edu/compsci/neural-networks-definition.html) on 29 May 2014*. Retrieved 2012*.
10. "Simulation of Self-Organizing Systems by Digital Computer". *IRE Transactions on Information Theory*. **4** (4): 76–84. [doi](https://en.wikipedia.org/wiki/Doi_(identifier)):[10.1109/TIT.1954.1057468](https://doi.org/10.1109%2FTIT.1954.1057468).
11. Miyakoshi, Yoshihiro, and Shohei Kato. ["Facial Emotion Detection Considering Partial Occlusion Of Face Using Baysian Network"](https://ieeexplore.ieee.org/document/5958891). Computers and Informatics (2011): 96–101.
12. Hari Krishna Vydana, P. Phani Kumar, K. Sri Rama Krishna and Anil Kumar Vuppala. ["Improved emotion recognition using GMM-UBMs"](https://ieeexplore.ieee.org/document/7058214/references). 2015 International Conference on Signal Processing and Communication Engineering Systems
13. Poria, Soujanya; Cambria, Erik; Bajpai, Rajiv; Hussain, Amir (September 2017). "A review of affective computing: From unimodal analysis to multimodal fusion". Information Fusion. **37**: 98–125.
14. Hjortsjö CH (1969). [*Man's face and mimic language*](https://books.google.com/books/about/Man_s_Face_and_Mimic_Language.html?id=BakQAQAAIAAJ). free download: [Carl-Herman Hjortsjö, Man's face and mimic language"](http://diglib.uibk.ac.at/ulbtirol/content/titleinfo/782346)
15. Ekman P, Friesen W (1978). Facial Action Coding System: A Technique for the Measurement of Facial Movement. Palo Alto: Consulting Psychologists Press.
16. Ekman P, Friesen WV, Hager JC (2002). *Facial Action Coding System: The Manual on CD ROM*. Salt Lake City: A Human Face.
17. [*"TensorFlow Release"*](https://github.com/tensorflow/tensorflow/releases/latest)*. Retrieved May 16, 2020*
18. Fabian Pedregosa; Gaël Varoquaux; Alexandre Gramfort; Vincent Michel; Bertrand Thirion; Olivier Grisel; Mathieu Blondel; Peter Prettenhofer; Ron Weiss; Vincent Dubourg; Jake Vanderplas; Alexandre Passos; David Cournapeau; Matthieu Perrot; Édouard Duchesnay (2011). [*"Scikit-learn: Machine Learning in Python"*](http://jmlr.org/papers/v12/pedregosa11a.html). Journal of Machine Learning Research. **12**: 2825–2830
19. Srivastava, Nitish; C. Geoffrey Hinton; Alex Krizhevsky; Ilya Sutskever; Ruslan Salakhutdinov (2014). ["Dropout: A Simple Way to Prevent Neural Networks from overfitting"](http://www.cs.toronto.edu/~rsalakhu/papers/srivastava14a.pdf) (PDF). *Journal of Machine Learning Research*. **15** (1): 1929–1958.
20. LeCun, Yann. ["LeNet-5, convolutional neural networks"](http://yann.lecun.com/exdb/lenet/). Retrieved 16 November 2013
21. LeCun, Yann; Bengio, Yoshua (1995). ["Convolutional networks for images, speech, and time series"](https://www.researchgate.net/publication/2453996). In Arbib, Michael A. (ed.). *The handbook of brain theory and neural networks* (Second ed.)
22. Lau, Suki (10 July 2017). ["A Walkthrough of Convolutional Neural Network — Hyperparameter Tuning"](https://towardsdatascience.com/a-walkthrough-of-convolutional-neural-network-7f474f91d7bd). *Medium*. Retrieved 23 August 2019
23. *[Schmidhuber, Jürgen](https://en.wikipedia.org/wiki/J%C3%BCrgen_Schmidhuber" \o "Jürgen Schmidhuber)* (2015). "Deep Learning". Scholarpedia. **10** (11): 85–117
24. Dreyfus, Stuart E. (1 September 1990). "Artificial neural networks, back propagation, and the Kelley-Bryson gradient procedure". *Journal of Guidance, Control, and Dynamics*. **13** (5): 926–928

# Списък с всички фигури

[Фигура 2‑1Невроните на конволюционен слой(в синьо), свързани към съответстващоот им рецептивно поле(червено) 11](#_Toc44117634)

[Фигура 2‑2Типична CNN архитектура 12](#_Toc44117635)

[Фигура 2‑3 Линейна активация 14](#_Toc44117636)

[Фигура 2‑4 Сигмоидна активация 15](#_Toc44117637)

[Фигура 2‑5 ReLU активация 15](#_Toc44117638)

[Фигура 2‑6: Ляво - невронна мрежа без Dropout. Десно - със Dropout 16](#_Toc44117639)

[Фигура 2‑7: SVM разделяне на множества 19](#_Toc44117640)

[Фигура 2‑8 Пример за матрица на грешките 22](#_Toc44117641)

[Фигура 3‑1 Примери за Action Unit-и и критерии за емоции 25](#_Toc44117642)

[Фигура 3‑2 Примерни изражения от Cohn-Kanade dataset-a 25](#_Toc44117643)

[Фигура 3‑3 Примери за опорни точки на дадено изражение 26](#_Toc44117644)

[Фигура 4‑1 Видове емоции, описани от проф. Р. Плутчик 27](#_Toc44117645)

[Фигура 4‑2 Пример за данни в дадена сесия описваща емоция от край до край 30](#_Toc44117646)

[Фигура 4‑3 Архитектура на CNN 1 32](file:///C:\projects\expressions-recognizer\documents_university\TODO_Documentation.docx#_Toc44117647)

[Фигура 4‑4 Архитектура на CNN 2 33](file:///C:\projects\expressions-recognizer\documents_university\TODO_Documentation.docx#_Toc44117648)

[Фигура 4‑5 Намиране на локален минимум на целева функция, в зависимост от стойността на обучаващия параметър. 34](#_Toc44117649)

[Фигура 4‑6 Примерно отбелязване на теглата по опорните точки 38](#_Toc44117650)

[Фигура 5‑1 Confusion Matrix за модел 1 42](file:///C:\projects\expressions-recognizer\documents_university\TODO_Documentation.docx#_Toc44117651)

[Фигура 5‑2 Резултат от предсказване на опорни точки Фигура 5‑3 Доближен резултат 43](#_Toc44117652)

[Фигура 5‑4 Confusion Matrix за SVM на Модел 2 44](#_Toc44117653)

[Фигура 5‑5 Confusion Matrix за целият Модел 2 45](file:///C:\projects\expressions-recognizer\documents_university\TODO_Documentation.docx#_Toc44117654)

[Фигура 5‑6 Примери за разминаващи се опорни точки в областта на устата. 45](#_Toc44117655)