СОФИЙСКИ УНИВЕРСИТЕТ
„СВ. КЛИМЕНТ ОХРИДСКИ“

Факултет по математика и информатика

Катедра „Компютърна информатика“

Дипломна работа

На тема

Разпознаване на емоции на лица

Ръководител

доц. Д-р В. Симеонова

кат. „Информационни технологии“

Дипломант

Щилиян Александров Узунов

ФН: 24676

Съдържание

1 Увод

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

В същността си задачата е класификационен проблем с 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, отвращение, страх, щастие, тъга, изненада и презрение. Тези емоции се установяват на базата освен на входна снимка и на подадени опорни точки (68 на брой, всяка с x, y координати), но и на информация за FACS активации, на база на опорните точки.

1.1 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.

1.2 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 задача ще бъде направен сравнителен анализ между двата модела.

1.3 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.

1.4 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	Action Unit – единица на активация в 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 в невронната мрежа. Използва се за налагане на регуларизация
KERNEL	Ядро, използва се като параметър при SVM моделите
ACCURACY / PRECISION / RECALL / MAE / MSE	Всички тези съкращения съответстват на различни метрики за оценка на даден модел

2 Преглед на използваните научни методи

2.1 Дефиниция за „Разпознаване на емоции“

Разпознаването на емоции е процес по идентифициране на човешката емоция. Хората често определят по различен начин 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 на лицата от професионалисти.

Точността на разпознаването на емоции се увеличава когато се комбинират различни източници на човешкото 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 в 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 с широката наличност на такива ресурси (базирани на знания). От друга страна недостатък на този подход е невъзможността му да се справя с концептуални нюанси и сложни лингвистични 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 разширяване на началният списък от възможни емоции. Корпус базираните подходи започват със стартов списък от думи, които изразяват мнение или емоции, и разширяват базата намирайки други думи с контекстно специфични характеристики в по-голям корпус. Докато подходите базирани на корпус от документи взимат предвид контекста, те все още имат проблеми с точността в различните домейни, тъй като дума в един домейн може да има различна ориентация и значение от това в друг домейн.

Статистическите 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

предсказва 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 използваните алгоритми за решаване на този проблем включват машини на поддържащите вектор, наивен Бейсов класификатор и максимум ентропия. Дълбокото самообучение също се използва широко в разпознаването на емоции. Добре известни архитектури, които решават такива типове проблеми са конволюционните невронни мрежи, машини за екстремно обучение и дълга краткотрайна памет (long short-term memory LSTM). Популярността на подходите за дълбоко самообучение в тази област се дължи от части на успеваемостта на тези подходи в полетата на компютърното зрение, разпознаване на реч и обработка на естествени езици.

Хибридните подходи при разпознаването на емоции са комбинация от техники, базирани на знания и статистически подходи, които експлоатират допълващи се характеристики за двата вида техники.

2.2 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 Хьортсьо. 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 с 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 почти всяко лицево изражение, което е анатомично възможно. Това става като то се декомпозира в специфични единици на действие (Action Units – AU). Тези единици са независими 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 с определяне

на FACS кодове използвайки софтуер. CANDIDE или Artnatomy са примери за такива системи, които позволяват израженията на човешкото лице да бъдат изкуствено създадени, 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 даден мускул на лицето. Въпреки, че мускулната активация не е част от FACS, основните мускули включени в лицевото изражения са добавени.

Оценката на интензитета на FACS се анотира, добавяйки букви A до E към Action Unit-а с цел - да се покаже колко „силно“ е активиран AU. Пример:

A – лека следа
B – умерено загатнато
C – маркирано
D – усилено
E – максимално

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

Списък от AUs и техните описания.

КОД	ИМЕ	АКТИВНИ МУСКУЛИ
0	Neutral face	
1	Inner brow raiser	frontalis (pars medialis)
2	Outer brow raiser	frontalis (pars lateralis)
4	Brow lowerer	depressor glabellae, depressor supercilii, corrugator supercilii
5	Upper lid raiser	levator palpebrae superioris, superior tarsal muscle
6	Cheek raiser	orbicularis oculi (pars orbitalis)
7	Lid tightener	orbicularis oculi (pars palpebralis)
8	Lips toward each other	orbicularis oris
9	Nose wrinkler	levator labii superioris alaeque nasi

10	Upper lip raiser	levator labii superioris, caput infraorbitalis
11	Nasolabial deepener	zygomaticus minor
12	Lip corner puller	zygomaticus major
13	Sharp lip puller	levator anguli oris (also known as caninus)
14	Dimpler	buccinator
15	Lip corner depressor	depressor anguli oris (also known as triangularis)
16	Lower lip depressor	depressor labii inferioris
17	Chin raiser	mentalis
18	Lip pucker	incisivii labii superioris and incisivii labii inferioris
19	Tongue show	
20	Lip stretcher	risorius w/ platysma
21	Neck tightener	platysma
22	Lip funneler	orbicularis oris
23	Lip tightener	orbicularis oris
24	Lip pressor	orbicularis oris
25	Lips part	depressor labii inferioris, or relaxation of mentalis or orbicularis oris
26	Jaw drop	masseter; relaxed temporalis and internal pterygoid
27	Mouth stretch	pterygoids, digastric
28	Lip suck	orbicularis oris

Примери за закодиране на емоции:

Емоция	**Списък от AUs**
Гняв	4+5+7+23
Презрение	R12A+R14A
Отвращение	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 бъдат повдигнати горните клепачи (5), клепачите да бъдат стегнати (7), да бъдат стегнати устните (23).

2.3 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 в класификационни и регресивни проблеми. Шаблоните, които те разпознават са цифрови, съдържащи се във вектори, в които трябва да се преведе всякакъв вид данни, от реалния свят – снимки/звук/текст.

2.3.1 Дефиниция

В дълбокото самообучение, конволюционна невронна мрежа (CNN) е клас дълбока невронна мрежа, най-често използвана в 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 се основава 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.

Конволюционните невронни мрежи са опростени версии на многослойните персептрони. Многослоен персептрон обикновено означава напълно свързана мрежа, т.е. всеки неврон в един слой е свързан с всички неврони в следващият слой. Тази пълна свързаност на мрежите ги прави лесни за нагласяване към данните. Типичните начини за регуларизация (опростяване) включват някаква форма на загуба на теглата, или деактивация на някой от невроните. CNN има различен подход към регуларизацията. Възползват се от йерархичните шаблони в данните и съставят по-сложни шаблони, използвайки по-малки под-шаблони. Затова по отношение на размерност и свързаност, CNN е едно ниво по-ниско от персептроните.

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

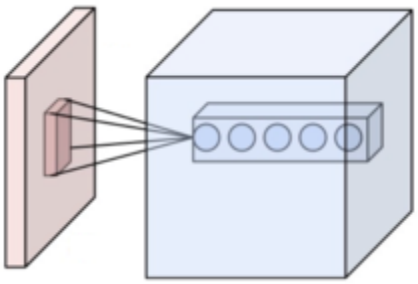
CNN нямат нужда от голяма преработка на данните, за разлика от други алгоритми за класификация на снимки. Това означава, че мрежата сама научава филтри, които при традиционните алгоритми трябва да бъдат написани на ръка. Тази независимост от предишно знание, и ръчно напасване намалява нужната работа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

2.3.2 Градивни блокове

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

2.3.2.1 Конволюционен слой

Конволюционният слой е основният градивен блок на CNN мрежата. Параметрите на слоя се стоят в набор научаеми филтри (или Kernels-ядро), които имат малко поле на рецептивност, но обикалят по целият поток от входни данни. По време 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.

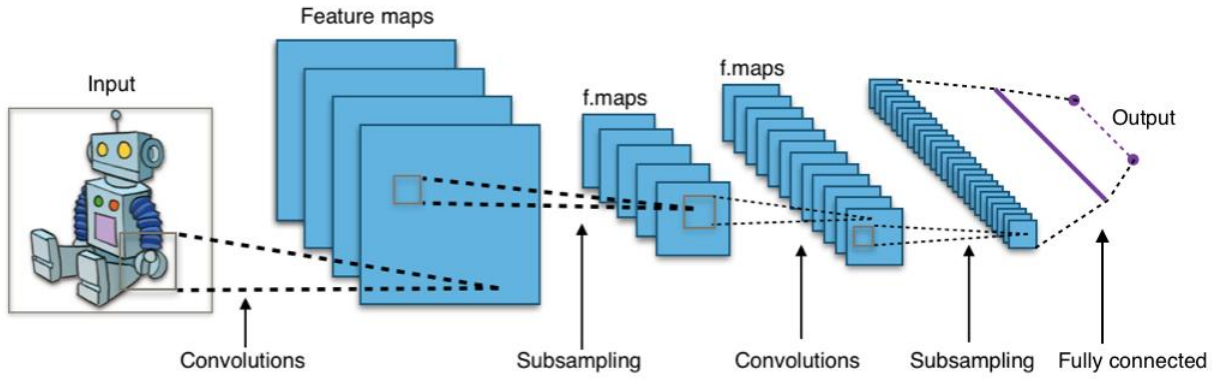


Фигура 2-1 Невроните на конволюционен слой (в синьо), свързани към съответстващото им рецептивно поле(червено)

Когато работим с входни данни с големи измерения, каквито са снимките, не е практично да свързваме всички неврони към тези в предишният слой, защото такава мрежова архитектура не взема под внимание 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 с малък регион от входни данни.

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



Фигура 2-2Типична CNN архитектура

Три хипер-параметъра контролират размера на изходните данни, които преминават през конволюционният слой – stride, depth и zero-padding.

- Stride (крачка) – контролира колко колони да бъдат алокирани в изходният формат. Когато този параметър е със стойност 1, тогава местим филтрите с 1 пиксел на всяка крачка. Това води до силно наслагване на рецептивните полета и също до изходни данни с голям обем. Когато параметърът е 2, тогава филтрите прескачат с 2 пиксела на крачка и т.н. Съответно – когато крачката е S > 0, филтърът 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 в слоя, които са свързани със същият регион във входните 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 в краят на входните данни, ако форматът налага някои пиксели да не бъдат обработени. Например ако имаме входни данни с формат 100x100px и размер на филтъра – 3px, то последният пиксел никога няма да бъде обработен. В такъв случай можем да добавим padding = 1, което 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).

$$\frac{W - K + 2P}{S} + 1$$

Формула 1 Изчисляване на изходния формат от конволюционен слой

2.3.2.2 Пулинг слой(Pooling layer)

Друга важна част от CNN е пулинга или слой за обединяване, което е форма на нелинейно намаляващо селектиране. Има няколко нелинейни функции за имплементиране на пулинг, измежду които максималният пулинг е най-популярен. Той разделя входната снимка в 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 с други входни параметри. Това е идеята за използване на пулинг в конволюционните невронни мрежи. Пулинг слоят служи за прогресивното намаляване на размерността на данните и по този начин да намали броят на параметрите, нужната памет и изчислителна сила за работата на модела. От тук следва, че той също ограничава до известна степен мрежата от прекомерно обучение. Честа практика е да се постави пулинг слой между последователни конволюционни слоеве в 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 2x2 с крачка 2. Такъв тип операция премахва 75% от активациите на предният слой.

$$f_{X,Y}(S) = \max_{a,b=0}^{1} S_{2X+a,2Y+b}$$

Формула 2: 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“, т.е. други функции, които могат да бъдат използвани са средно аритметичен пулинг или L2 нормиран пулинг.

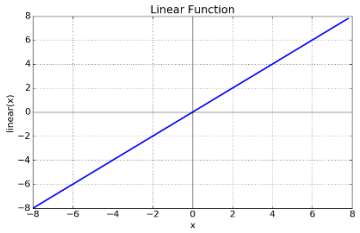
Средно аритметичният обикаля по всички входни числа, и взема средно аритметичното им, L2 прави евклидова нормализация на входните числа.

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

2.3.2.3 Напълно свързан слой и видове активиращи функции

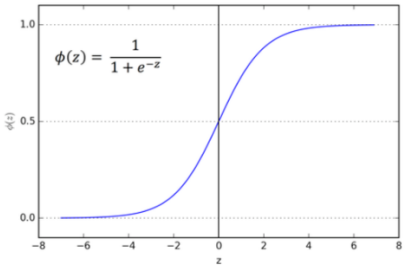
Накрая, след няколко конволюционни и пулинг слоя, логиката, която научава как комбинацията от филтри разпознава даден образ се прави чрез напълно свързан слой. Невроните в напълно свързания слой имат връзки към всеки неврон от предишния слой, точно както е в стандартните (не конволюционни) невронни мрежи. В напълно 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.



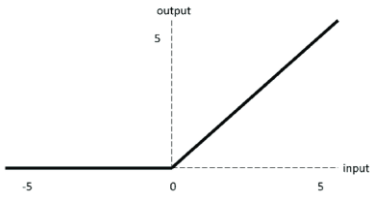
Фигура 2-3 Линейна активация

- 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 вероятността на каквото и да е число между 0 и 1, сигмоидната активация е правилният избор за такъв тип проблем.



Фигура 2-4 Сигмоидна активация

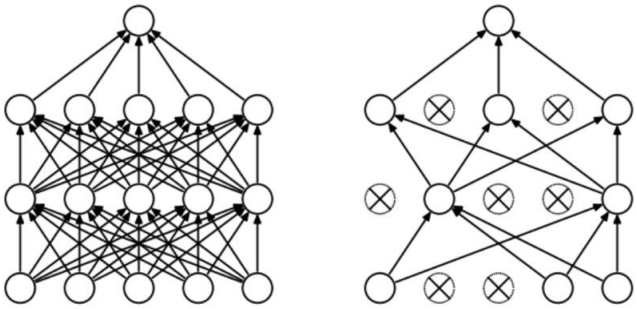
- ReLU активация – Това е най-често използваната активация за скрити слоеве в невронните мрежи. Формулата за ReLU е: f(x) = max(0, x) като предимствата й са няколко, първо има рядко разпределена активация при случайно инициализиране на невронната мрежа – около 50% от невроните се активират, второ: по-добро предаване на грешката в задна посока по време на backpropagation – тази функция няма проблемът с изчезващият градиент, както при сигмоидната активация например, трето – лесна за изчисление, необходимите 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 2-5 ReLU активация

2.3.2.4 Dropout слой

Честа практика е напълно свързаните слоеве да бъдат следвани от dropout слой. Този вид слой има за цел да адресира проблема с прекомерното обучение на мрежата спрямо обучаващите данни в невронните мрежи. Той работи като на всеки неврон от предишният слой (напълно свързаният) се добавя вероятност да бъде изпуснат в следващия стадий 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 нагледно да се види в следващата фигура.



Фигура 2-6: Ляво - невронна мрежа без Dropout. Десно - с Dropout

2.4 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 изчислява 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 отношение 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 стриктно само за алгоритъма за изчисляване на градиента, а не за това как градиента се използва, но терминът често се използва и за целият обучаващ алгоритъм, включително за това как градиента се използва. Backpropagation генерализира изчислението на градиента по делта правилото, което е еднослойната версия на backpropagation.

Преглед на параметрите и алгоритъма.

Нека X е входен вектор от атрибути, а Y е 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). Тогава:

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

L – брой слоеве

W^l – Теглата между слоеве l-1 и l

f^l – Активационни функции на слой l. За класификационни задачи последният слой обикновено използва логистична функция за двоична класификация и софтмакс за многокласова класификация, докато за скритите слоеве използваме ReLU

По време на изчислението на производните на алгоритъма се използват и други междинни стойности. Тези стойности са описани по-долу. За целите на backpropagation-a, 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:

$$g(x) := f^L(W^L f^{L-1}(W^{L-1} \cdots f^1(W^1 x) \cdots))$$

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

$C\ (y_i, g(x_i))$

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

Backpropagation изчислява градиента за фиксираните входно-изходни двойки (x_i, y_i), където теглата W могат да варират. Всеки индивидуален компонент на градиента $\partial C/\partial w^l_{jk}$ може да бъде изчислен по верижното правило за диференциране, но изчислението по този начин за всяко отделно тегло е неефективно. Ефективното изчисление на градиентите избягвайки дублиращи се изчисления и изчисляването на междинните стойности се прави, като се изчислява градиента на всеки слой – по-точно, градиента на претегленият вход за всеки слой – отбелязан с d^l отзад напред.

Неформално, начинът по който дадено тегло W влияе на целевата функция е чрез ефектът си на следващият слой, и това става линейно, d^l е единствената информация, която е нужна за изчисляване на градиентите на

теглата на слой l, и след това можем да изчислим предишният слой d^{l-1} и да повторим рекурсивно. Това решава проблема с неефективността по 2 начина. Първо, избягва се дублирането, 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 $\partial a^{l'}_{j'} / \partial w^{l}_{jk}$.

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 с 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.

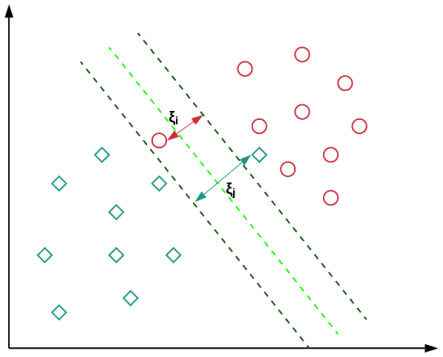
Математическите свойства, които трябва да притежава целевата функция са 2, за да може да бъде използвана в backpropagation. Първо е, че трябва да бъде записана като средното $E = \frac{1}{n} \sum_x E_x$ върху дадени функции, които изчисляват грешки E_x за n на брой обучаващи примера x. Нуждата от това 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 с минаването през „плата“ на функцията – това ще рече продължителни плоски участъци.

2.5 Машини на поддържащите вектори (SVM)

В машинното самообучение, машина на поддържащите вектори(SVM) представлява метод за обучение 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 невероятностен двоичен линеен класификатор.

SVM моделът представя примерите като точки в n-мерното пространство, така разпределени, че примерите от отделни категории са разделени от права, и разстоянието от 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 нелинейна класификация, използвайки различни видове ядра (kernels).

По-формално, машините на поддържащи вектори конструират хипер-равнина, или множество хипер равнини в многомерното пространство, които се използват за класификация или регресия. Добро разделяне на данните се постига когато хипер-равнината, която има най-голямо отстояние от най-близките обучаващи точки, на който и да е от класовете. От тук следва и, че колкото 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 $(x_1, y_1),\ldots,(x_n, y_n)$ където y_i е или 1 или -1, всеки индикиращ класът към, който точката x_i принадлежи. Всеки x_i е p-измерен вектор от реални числа. Искаме да намерим хипер-равнина с максимално отстояние, която разделя групата точки x_i за, които $y_i = 1$ от групата от точки, за които $y_i = -1$. Всяка хипер-равнина може да бъде изразена като множество от точки x удовлетворяващо следното уравнение:

$$\vec{w} \cdot \vec{x} - b = 0$$

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

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

$\vec{w} \cdot \vec{x} - b = 1$ всичко на или над тази равнина е от клас 1

$\vec{w} \cdot \vec{x} - b = -1$ всичко на или под тази равнина е от клас -1

Геометрично, разстоянието между тези 2 хипер-равнини е 2/||w|| така, че за да максимизираме 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 добавяме следните ограничения за всяко i:

$\vec{w} \cdot \vec{x}_i - b \geq 1$, if $y_i = 1$

$\vec{w} \cdot \vec{x}_i - b \leq -1$, if $y_i = -1$

Тези ограничения могат да бъдат записани като

$$y_i(\vec{w} \cdot \vec{x}_i - b) \geq 1, \quad \text{for all } 1 \leq i \leq n. \qquad (1)$$

Тези ограничения налагат всяка точка да лежи на правилната страна на хипер-равнината. С това оптимизационният ни проблем се превръща в минимизиране на вектора w по отношение на $y_i(\vec{w} \cdot \vec{x}_i - b) \geq 1$ for $i = 1, \ldots, n$.

Стойностите на w и b, които решават този проблем представляват нашият класификатор. Важна последица от това геометрично описание е, че хипер-равнината с максимално разстояние е изцяло определена от тези точки x, които лежат най-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.

$$\max(0, 1 - y_i(\vec{w} \cdot \vec{x}_i - b))$$

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

$$\left[\frac{1}{n}\sum_{i=1}^{n}\max(0, 1 - y_i(\vec{w} \cdot \vec{x}_i - b))\right] + \lambda\|\vec{w}\|^2$$

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

2.6 Метрики за валидация

2.6.1 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.

$$Accuracy = \frac{Number\ of\ Correct\ predictions}{Total\ number\ of\ predictions\ made}$$

Формула 3 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 A и 2% от клас B, нашият модел лесно би могъл да постигне 98% тренировъчна точност просто като прогнозира, че всеки пример принадлежи към клас A.

Когато същият модел се тества с обучаващо множество, в което 60% от примерите са от клас A и 40% от клас B, 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.

2.6.2 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 двоичен 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:

n=165	**Predicted: NO**	**Predicted: YES**
Actual: NO	50	10
Actual: YES	5	100

Фигура 2-8 Пример за матрица на грешките

Тук виждаме 4 важни неща:

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 средно аритметичното на всички стойности, лежащи на главният диагонал, в този случай = (TP + TN) / Всички примери. Този тип матрица е базата за оценка на другите видове метрики.

2.6.3 F1 резултат

F1 резултатът е хармоничното средно между precision и recall. Стойностите на F1 са в интервала [0, 1]. То показва колко прецизен е класификаторът (колко входни данни класифицира коректно) и също така колко устойчив е (не пропуска голям 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 се изрази по следния начин:

$$F1 = 2 * \frac{1}{\frac{1}{precision} + \frac{1}{recall}}$$

Формула 4 F1 резултат

Precision – Наричаме броят на коректните позитивни резултати, разделени на броя на позитивните резултати предсказани от класификатора.

$$Precision = \frac{TruePositives}{TruePositives + FalsePositives}$$

Формула 5 Precision

Recall – Наричаме броят на коректните положителни резултати, разделен на броя на всички релевантни примери (всички, които е трябвало да бъдат идентифицирани като позитивни).

$$\text{Recall} = \frac{TruePositives}{TruePositives + FalseNegatives}$$

Формула 6 Recall

2.6.4 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 далече са отишли предсказаните 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:

$$MeanAbsoluteError = \frac{1}{N}\sum_{j=1}^{N}|y_j - \hat{y}_j|$$

Формула 7 Средна абсолютна грешка

2.6.5 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.

$$MeanSquaredError = \frac{1}{N}\sum_{j=1}^{N}(y_j - \hat{y}_j)^2$$

Формула 8 Средна квадратична грешка

3 Използвано множество от данни (Dataset)

През 2000 година Cohn-Kanade dataset-а е съставен с цел промотирането на изследвания в областта на автоматичното разпознаване на лицеви изображения. От тогава 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 самообучение и поведенчески науки, и може да бъде използвано на много места, като например сигурност, интеракция човек-компютър, сигурност на шофьорите, и здравеопазване.

В използваната дистрибуция CK, базата съдържа 593 сесии на 123 човека. Всяка сесия съдържа снимки, като стартовата снимка е в неутрално изражение, а последната снимка – в пиково изражение. Пиковият кадър е анотиран с 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 човека, провеждащ експеримента, да извършат серия от 23 лицеви изображения. Те включват единични FACS маркери, или комбинация от FACS маркери. Снимките са във формат 640x480px с 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.

AU	*Name*	N	AU	*Name*	N	AU	*Name*	N
1	*Inner Brow Raiser*	173	13	*Cheek Puller*	2	25	*Lips Part*	287
2	*Outer Brow Raiser*	116	14	*Dimpler*	29	26	*Jaw Drop*	48
4	*Brow Lowerer*	191	15	*Lip Corner Depressor*	89	27	*Mouth Stretch*	81
5	*Upper Lip Raiser*	102	16	*Lower Lip Depressor*	24	28	*Lip Suck*	1
6	*Cheek Raiser*	122	17	*Chin Raiser*	196	29	*Jaw Thrust*	1
7	*Lid Tightener*	119	18	*Lip Puckerer*	9	31	*Jaw Clencher*	3
9	*Nose Wrinkler*	74	20	*Lip Stretcher*	77	34	*Cheek Puff*	1
10	*Upper Lip Raiser*	21	21	*Neck Tightener*	3	38	*Nostril Dilator*	29
11	*Nasolabial Deepener*	33	23	*Lip Tightener*	59	39	*Nostril Compressor*	16
12	*Lip Corner Puller*	111	24	*Lip Pressor*	57	43	*Eyes Closed*	9

Table 1. *Frequency of the AUs coded by manual FACS coders on the CK+ database for the peak frames.*

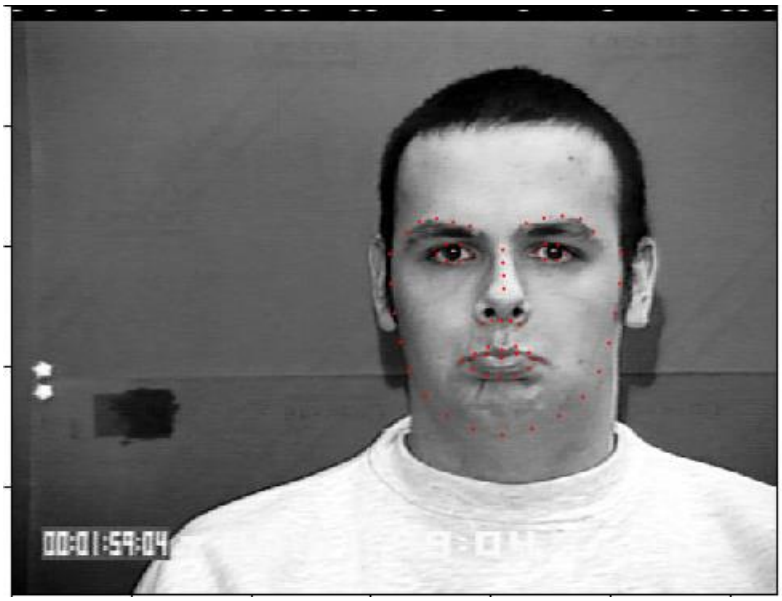
Emotion	Criteria
Angry	AU23 and AU24 must be present in the AU combination
Disgust	Either AU9 or AU10 must be present
Fear	AU combination of AU1+2+4 must be present, unless AU5 is of intensity E then AU4 can be absent
Happy	AU12 must be present
Sadness	Either AU1+4+15 or 11 must be present. An exception is AU6+15
Surprise	Either AU1+2 or 5 must be present and the intensity of AU5 must not be stronger than B
Contempt	AU14 must be present (either unilateral or bilateral)

Фигура 3-1 Примери за Action Unit-и и критерии за емоции

В горепосочената 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 в снимка като активна.

Фигура 3-2 Примерни изражения от Cohn-Kanade dataset-a

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



Фигура 3-3 Примери за опорни точки на дадено изражение

4 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 същ. На фигурата отдолу се виждат видовете емоции, които психологът проф. Робърт Плутчик описва.

Фигура 4-1 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.

4.1 Използвани технологии

Имплементацията на системата е с 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 с GPU ускори процесът на обучение 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 инженери 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 с GPU-ускорени примитиви за дълбоки невронни мрежи. CudNN съдържа силно оптимизирани имплементации на стандартни методи като конволюция, пулинг, нормализация и активиране на слоеве.

4.2 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 спаднаха от 15 часа на 10 минути, което направи процесът на разработка много по-бърз, и даде възможност за много повече експерименти с параметрите на модела.

4.3 Предварителна обработка на данните

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

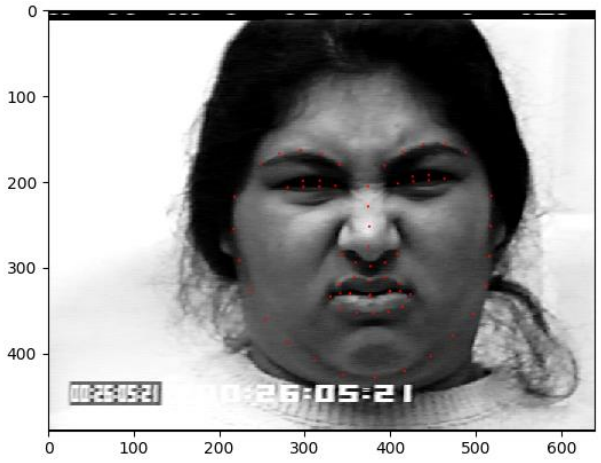
В множеството от обучаващи данни за всяка сесия от дадена емоция за субект, има средно по 20 снимки. Първата снимка с неутрално изражение, а последната с най-голям интензитет на изражението. На фигура 4.2 може да се видят данните от 1 сесия в суров формат.

Фигура 4-2 Пример за данни в дадена сесия описваща емоция от край до край

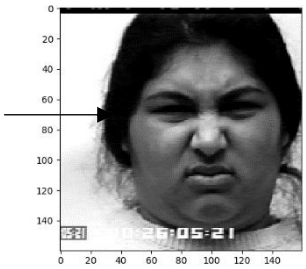
Както се вижда, данните са сортирани от неутрално, към най-експресивна форма на емоцията. Тъй като Модел 1 иска да прави класификацията от край до край, на него като обучаващи данни са му нужни само най-експресивните примери, от дадена сесия – за това биват взети само последните 3 снимки.

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) с 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-а. 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 640x480, изрязват се до 480x480. Следващата стъпка е оразмеряване до 160x160 пиксела. С това обработката приключва, и данните се подават на CNN мрежите на Модел 1 и Модел 2.



Фигура 4-3 Входен формат



Фигура 4-4 Формат след обработка

4.4 Описание на Модел 1

Модел 1 решава задачата по общоприетия начин, в който класификацията се извършва от край до край от една CNN мрежа.

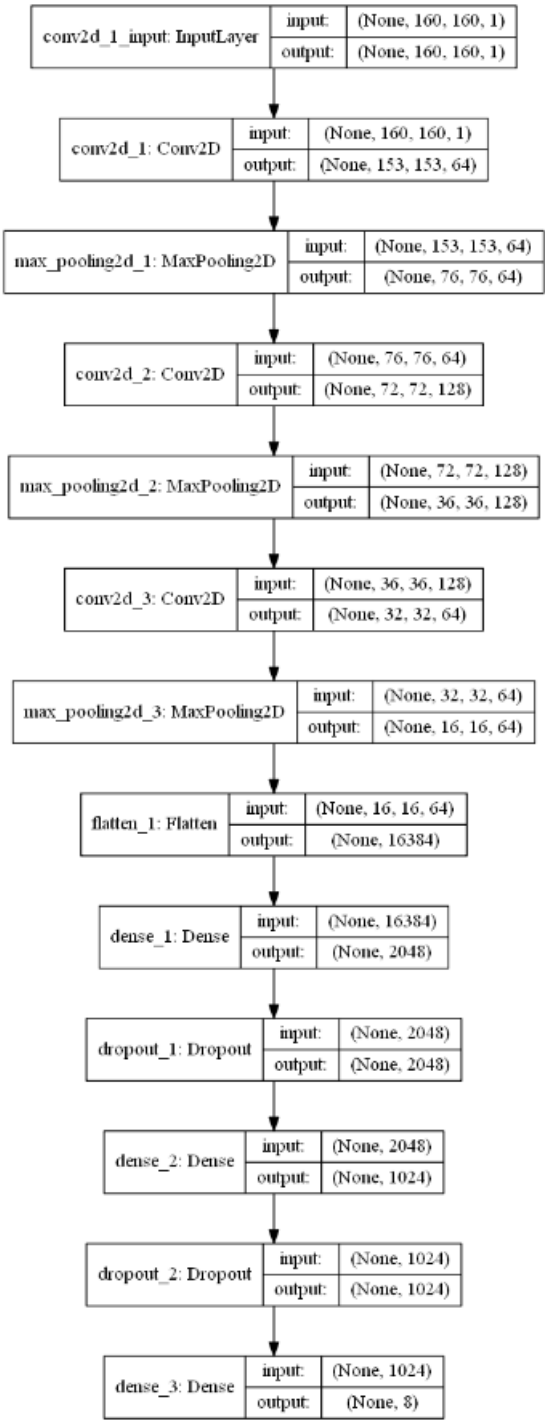
Архитектура на мрежата:

```
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:



Фигура 4-3 Архитектура на CNN 1

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 153x153 преминават към размерност 76x76.

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 2x2 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 напълно свързан слой, типичен за всяка невронна мрежа.

Flatten слоят сменя размерността на данните като ги превръща 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 претренира.

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

4.5 Описание на Модел 2

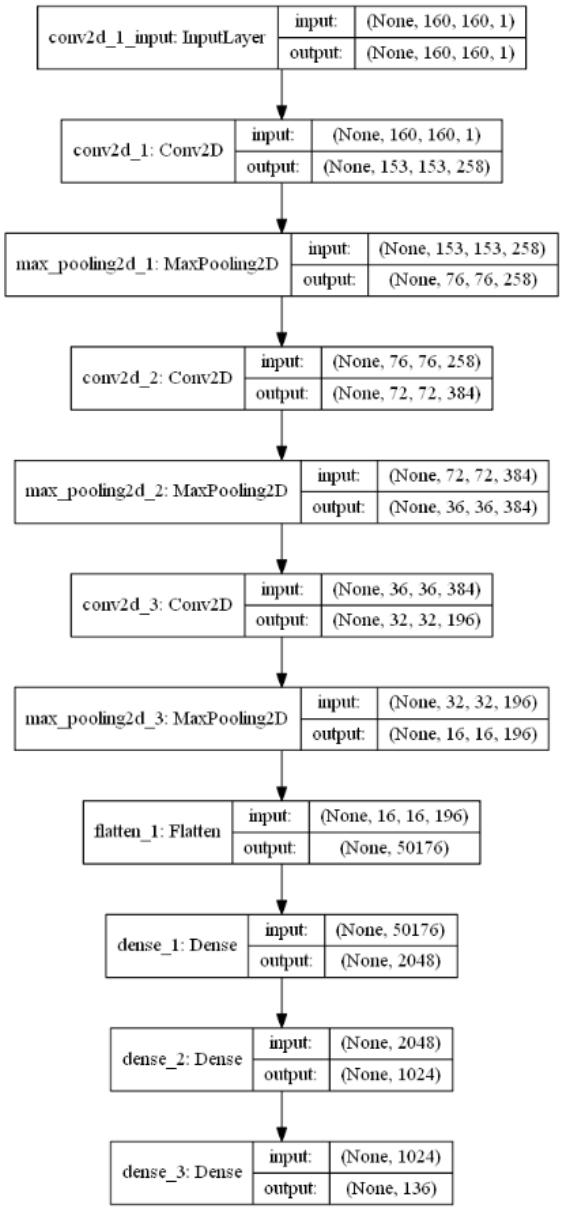
Модел 2 решава задачата на 2 нива. Първо ниво намира опорните точки по подадена входна снимка използвайки CNN. Второ ниво намира емоцията, по подадени входни опорни точки използвайки SVM класификатор.

Архитектура на невронната мрежа:

```
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'))
```



Фигура 4-4 Архитектура на CNN 2

Мотивацията за избор на слоеве е същата както и при Модел 1. Основна разлика между двата модела е в 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 направени експерименти 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 x и y 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')
```

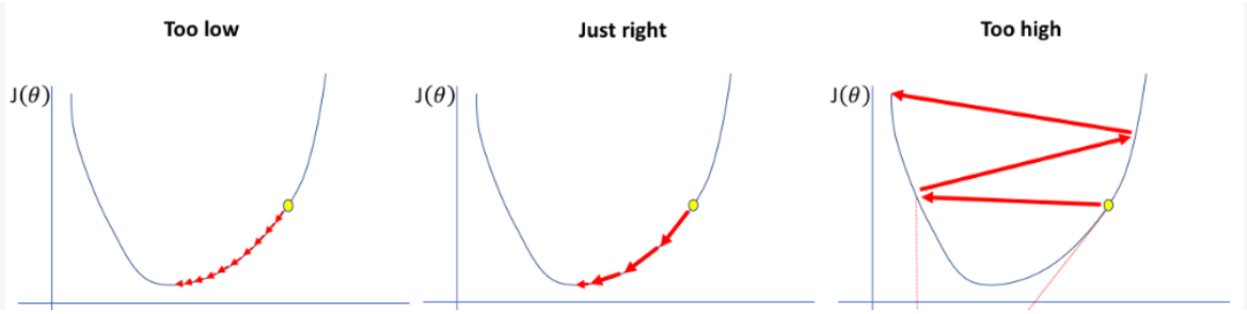
Използват се линейно ядро, и регуларизационен параметър C=1000. Точността на SVM класификатора спрямо обучаващите данни от множеството е 98%. Тази висока точност означава, че до голяма степен крайната точност на Модел 2 ще зависи от това колко добре CNN мрежата, която предвижда позицията на опорните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

4.6 Обучение на модел 1

За обучението на Модел 1 е използван Адам оптимизатор, с 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 в keras библиотеката е със стойност 0.001. Изборът на правилна стойност се оказа ключов, защото стойността по подразбиране е прекалено голяма, и с 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 4-5 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 повишава. Бяха правени и опити с повече от 15 епохи, но се оказа ненужно, защото след 13-14 епоха, точността се покачваше с 0.001%.

4.7 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 в целевата функция. Бяха правени експерименти с 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 например 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 гореописаният 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 с 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.

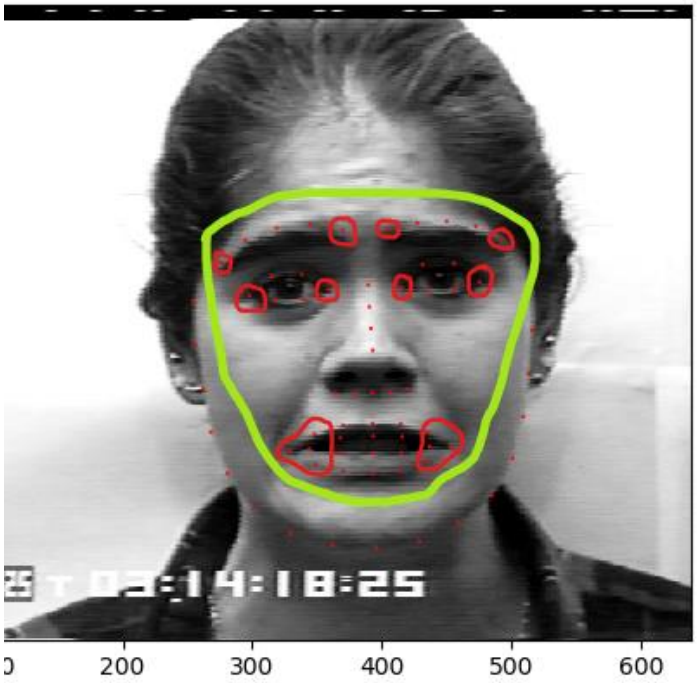
```
 return K.sum(K.square(yPred - yTrue))
```

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 4-6 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 в зелената зона са с нормален приоритет, а всички извън нея са с нисък приоритет.

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

Успеваемост целева функция MSE: 77%

Успеваемост custom функция: 81.725%

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

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

4.8 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	_____ 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

```
_________________________________________________
_____________________
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
```

```
_________________________________________________
_____________________
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 в 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 почти 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 проблема 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.

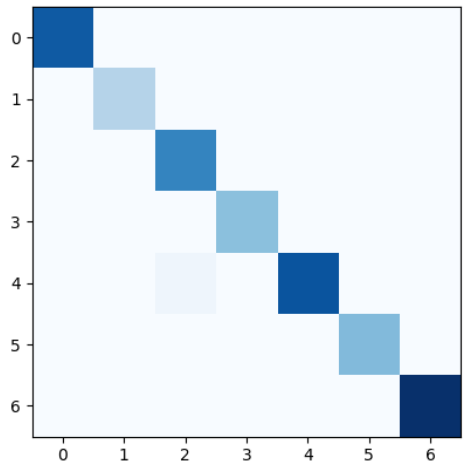
5 Резултати от работата на системата

5.1 Резултати на Модел 1

Тъй като модел 1 извършва класификация от край до край (за разлика от модел 2), то 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 7x7 – 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 с код 4 – страх, като 37 от тях са били познати правилно, и 2 са били грешно класифицирани от нашият модел като 2 – презрение.

```
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]
```



Фигура 5-1 Confusion Matrix за модел 1

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 с много голяма точност.

5.2 Резултати на Модел 2

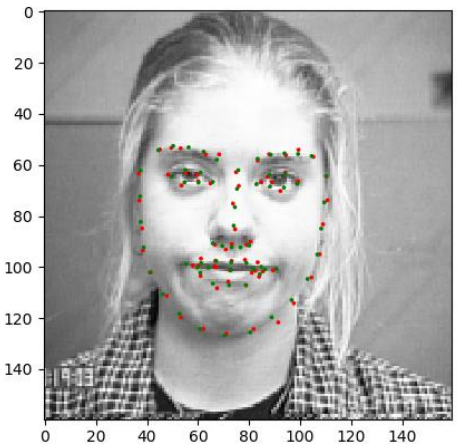
Поради архитектурата на модел 2, за измерването на работата му са нужни няколко вида метрики – такива за регресивни задачи, и такива за класификационни задачи.

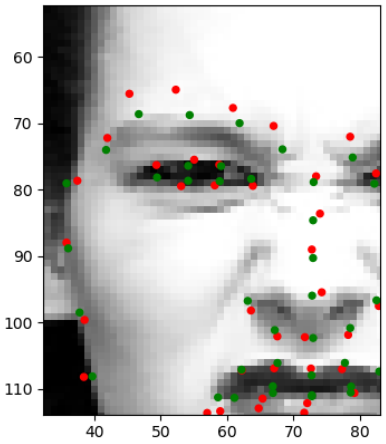
За измерването на точността на невронната мрежа, която продуцира 68-те опорни точки, използваме средна абсолютна грешка (описана в точка 2.6.4), която измерва средното абсолютно отстояние на всяка координата (x или y, от целевата и стойност).

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 160x160px представлява 1.2314/160 = 0.007% от цялото пространство на снимката. Пример за предсказани и целеви точки може да се види на долната фигура. С зелено са отбелязани правилните положения на точките, а с червено – тези, които са били предсказани.

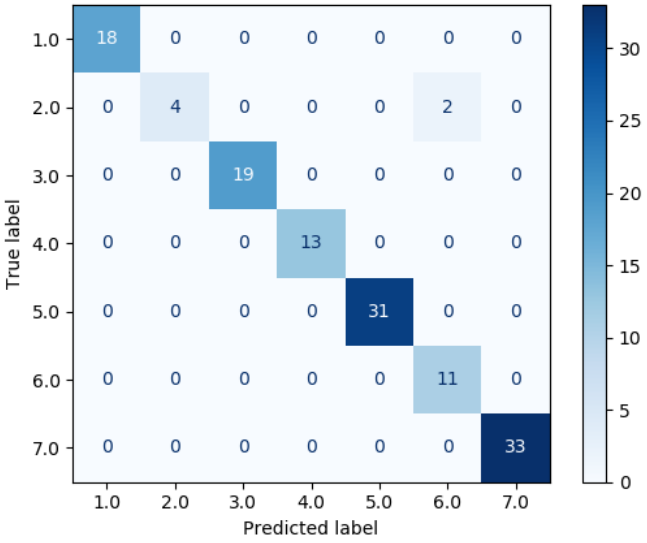




Фигура 5-2 Резултат от предсказване на опорни точки Фигура 5-3 Доближен резултат

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. Използваната метрика е точност (Accuracy). На фигура 5.4 се вижда confusion matrix на база на, която са направени изчисленията.

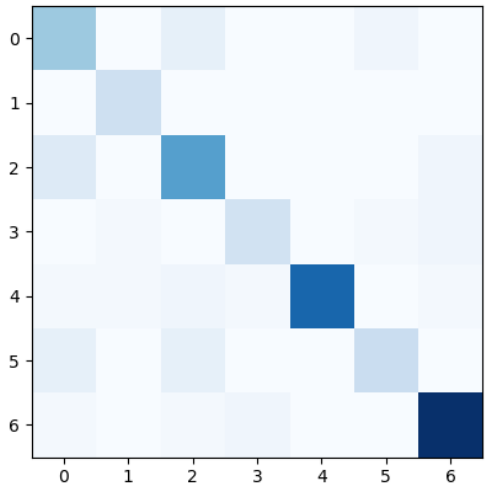


Фигура 5-4 Confusion Matrix за SVM на Модел 2

SVM Accuracy за Модел 2 = **98.47%**

Последната част, която измерваме е точността на целия модел от край до край, 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.

```
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]
```

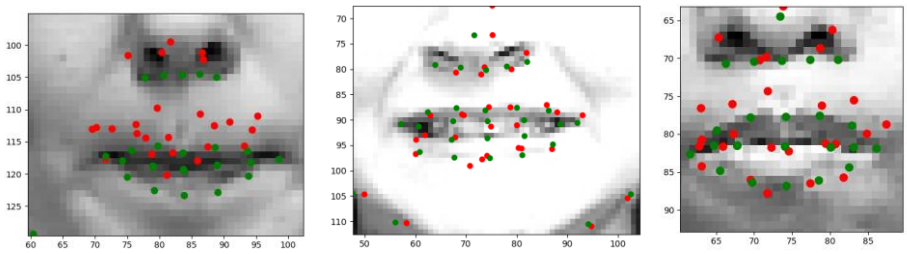


Фигура 5-5 Confusion Matrix за целият Модел 2

Тук както виждаме от матрицата резултатите са по-ниски, с доста повече стойности, които не лежат на главният диагонал. Това показва и повече грешки, като 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 5-6 Примери за разминаващи се опорни точки в областта на устата.

За да бъде преодоляна 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.

5.3 Резултати от използването на двата модела с камера

И двата модела бяха тествани за разпознаването на изражения използвайки камера, като всеки кадър от камерата беше подаван за разпознаване на всеки от моделите. Измерими резултати на точността не бяха направени, но цялостното впечатление беше, че точността е много по-ниска от колкото при данните, 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 с 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.

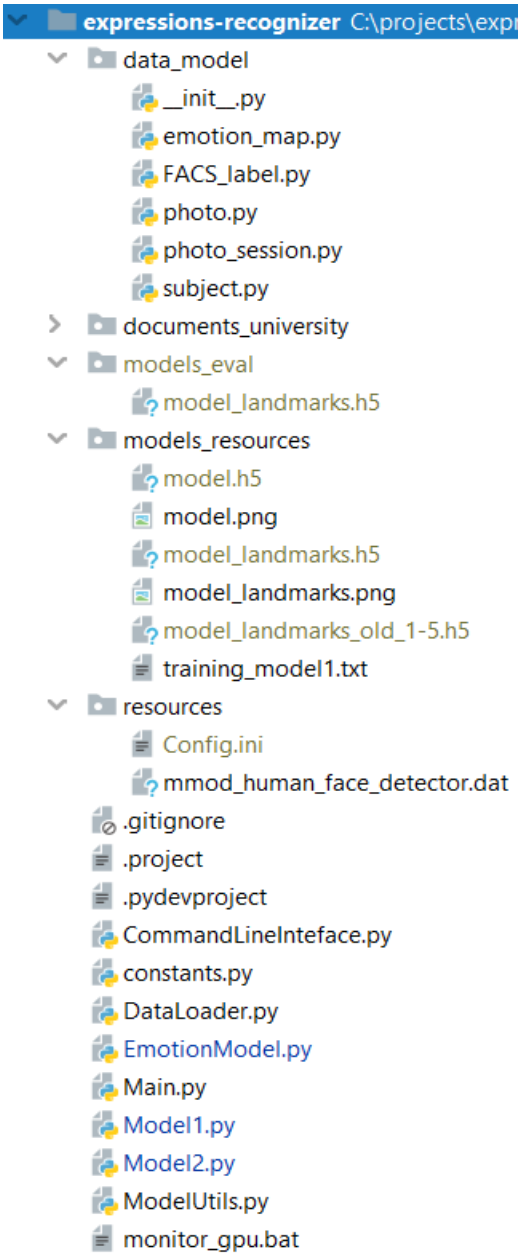
Като евентуално решение на проблема с задния фон бихме могли да приложим алгоритъм за засичане на лице, и като входна информация за моделите да се подава само изрязана част на лицето на субекта на снимката.

За проблемите с осветлението и позицията на главата ще са ни нужни повече 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.

6 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

6.1 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 в 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-а.

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.

6.2 Необходим софтуер и хардуер

Нужна е инсталация на python3.5+ версия, с 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 само върху CPU-то на машината.

6.3 Описание на потребителския 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 – Показва резултатите на emotion model-а след обучение
- exit – Изход от приложението
- listSubjects – Показва списък на всички субекти в обучаващото множество
- listSessions – Показва списък на всички сесии на подаден субект
- help – Показва всички възможни команди
- getEmotion – Показва 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

7 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 в бъдеще. Това са проблемите с ниската успеваемост при разпознаване на изображения от камера, и повишаване на успеваемостта на модел 2 (която както беше упоменато в точка 2 е почти 82%).

Ниската успеваемост при разпознаване на изображения от камера се дължи най-вероятно на средата, в която са заснети обучаващите примери. При почти всички картинки има наличен черен заден фон, без елементи по него и изражение на човек на преден план. При заснемане от камера в жилищна среда, на заден фон излизат 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 преодолян е като се добавят още обучителни примери със снимки от жилищна среда.

Втори начин, по който може да бъде преодолян 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 шума и също би подобрило работата на модела при данни идващи от камера.

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

8 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. MIT Press. p. 326.
3. "Convolutional Neural Networks (LeNet) – DeepLearning 0.1 documentation". 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" (PDF). Communications of the ACM. 60 (6): 84–90. doi:10.1145/3065386. ISSN 0001-0782.
5. Bishop, C. M. (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" (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". Proc. Natl. Acad. Sci. U.S.A.* ***79*** *(8): 2554–2558. Bibcode:1982PNAS...79.2554H.*
9. *Russell, Ingrid. "Neural Networks Module". Archived from the original 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:10.1109/TIT.1954.1057468.
11. Miyakoshi, Yoshihiro, and Shohei Kato. "Facial Emotion Detection Considering Partial Occlusion Of Face Using Baysian Network". 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". 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*. free download: Carl-Herman Hjortsjö, Man's face and mimic language"
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". 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". 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" (PDF). *Journal of Machine Learning Research*. **15** (1): 1929–1958.
20. LeCun, Yann. "LeNet-5, convolutional neural networks". Retrieved 16 November 2013
21. LeCun, Yann; Bengio, Yoshua (1995). "Convolutional networks for images, speech, and time series". 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". *Medium*. Retrieved 23 August 2019
23. *Schmidhuber, Jürgen (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

9 Списък с всички фигури