# Препознавање на емоции од слика

**Ментор:**

**Проф. Д-р Ивица Димитровски**

**Изработиле:**

**Јана Ѓелевска 171084**

**Марта Толевска 171109**

# Содржина

# Библиотеки

## A. PIL

PIL (Python Imaging Library) [1] е бесплатна библиотека со отворен код за програмскиот јазик Python која овозможува отворање, манипулација и зачувување на различни типови и формати на слики и датотеки. Главната функција е брз пристап до податоци кои се зачувани во вид на пиксели со што се овозможува основа за понатамошна употреба на алатки за процесирање на слики.

Во склоп на овој проект ние ја користевме класата Image од PIL библиотката која овозможува читање на слики, процесирање како и креирање на слики од почеток. Класата ја искористивме при креирање на сликите кои потоа се дел од множеството за тренирање и тестирање на моделот. Искористена е за креирање на слики од податоци кои се читаат од готова датотека во која секоја слика е претставена во вид на дводимензионална низа од броеви (вредност за секој пиксел од сликата), која што со помош на функцијата *Image.fromarray(matrix)* ја креира сликата и потоа со методот *save()* сликата се зачувува во соодветниот фолдер.

## B. Numpy

Numpy [2] е библиотека за програмскиот јазик Pyhton, која овозможува работа со големи, мулти-димензионални низи и матрици и обезбедува огромна колекција од напредни математички функции за работа со истите податоци. Голема предност е тоа што овозможува брзо и ефикасно работење, односно оперирање, со податоците.
Имајќи во предвид дека сликите со повеќе канали, се претставуваат како три-димензионални низи. Индексирањето, промената на големината и спојувањето (операции) со други низи, е многу ефикасен начин да се пристапи до некој конкретен пиксел од слика. Numpy низата е универзална податочна структура во OpenCV, за работа со слики, филтри и сл.

## C. Pandas

Pandas [3] е софтверска библиотека напишана за програмскиот јазик Python која овозможува манипулација и анализа на податоците. Имено, овозможува податочни структурии операции за манипулирање со нумерички табели и временски серии. Оваа библиотека ја искористивме во неколку делови од нашиот проект. Најчесто податоците ги интерпретиравме во вид на податочната структура DataFrame, која овозможува побрза и поефикасна манипулација на податоците содржани во ваквата структура.

## D. TensorFlow и Keras

TensorFlow [4] е платформа за машинско учење со отворен код и која работи на принципот end-to-end. Може да се објасни како инфраструктурен слој за диференцирано програмирање.

Платформата комбинира четири клучни способности:

- Ефикасно извршување на ниско ниво на tensor операции на CPU (процесорот), GPU (графичкиот процесор) или TPU(tensor процесор)
- Пресметување на градиентот на произволни диференцијабилни изрази
- Скалирање на пресметките на повеќе уреди
- Изнесување (exporting) на програми/графови, до надворешни извршувачи (runtimes) како сервери, прелистувачи (browsers), мобилни и вгнездени уреди.

Keras [5] е API [6] (Application Programming Interface) за длабоко учење напишан во Python, кој работи на платформата за машинско учење TensorFlow. Развиен е за овозможување на брзи експерименти со што за кус период ќе може за одредена идеа да даде некаков резултат. Основните структури на податоци на Керас се слоеви и модели. Наједноставниот тип на модел е Секвенцијален модел - линеарна група од слоеви. За покомплексни архитектури, треба да се користи Keras функционален API, кој овозможува градење произволни графикони на слоеви или пишување модели целосно од нула преку подкласа.

Keras е исто така високо-флексибилна рамка погодна за итерирање на најсовремените идеи за истражување, која го следи принципот на прогресивно обелоденување на комплексноста. Ова го олеснува започнувањето, но исто така овозможува справување со произволни напредни кориснички сценарија (use cases), побарувајќи само инкрементално учење во секој чекор. На истиот начин на кој може да се тренира и тестира обична невронска мрежа само со неколку линии код така може да се креираат нови тренирачки функции или обемни и богати модели.

### E. Face_recognition

Face_recognition [7] е готова библиотека креирана за програмскиот јазик Python со помош на која се препознаваат лица на дадена слика. Изградена е со помош на модел на длабоко учење кој има 99,38% точност. Освен за препознавање на лица од дадена слика со оваа библиотека може да се врши манипулација со истите како и препознавање на некои личности кои се познати на библиотеката. Во склоп на овој проект оваа библиотека ја искористивме за брзо и ефикасно пронаоѓање на лица од дадена слика од кои потоа со нашиот модел ќе можеме да им ја препознаеме емоцијата.

Во овој проект, ја искористивме функцијата *face_recognition.face_locations(face_image)*, која како влезен аргумент ја зема сликата и го пронаоѓа лицето на сликата.

### F. CV2

CV2 [8] е библиотека со отворен код наменета за програмскиот јазик Python која има голем број на функции за манипулација со слики. Најголема предност е што враќа конзистентни објекти од класата NumPy наместо native Python objects врз кои може да се извршат голем број на функции од истата класа која е постабилна и брзо пресметува операции со низи. Оваа е од голема важност бидејќи сите функции во оваа библиотека на влез примаат слики или низи од пиксели кои во позадина ги претвара во низи од NumPy класата и се извршуваат соодветни операции од истата.

*cv2.cvtColor(src, code, dst, dstCn)* е функција од библиотеката cv2 која се користи за промена на простор на бои на слика. Излезот од функцијата е слика со иста големина, но може да е со различен број на канали и во различен простор на бои. Значењата на аргументите кои ги прима оваа функција се:

- *src*: сликата на која сакаме да и го промениме просторот на бои
- *code*: кодот на каква промена сакаме да извршиме стар_нов простор на бои
- *dst*: опционален параметар за излезната слика
- *dstCn*: бројот на канали во излезната слика, доколку не е инцијализиран се зема бројот на канали на посакуваниот простор на бои назначен во кодот

*cv2.resize(source, dim, fx, fy, interpolation)* е функција од библиотеката cv2 која се користи за промена на големината на сликата. Промената на големината на сликата значи промена на конфигурацијата на сликата, како промена на висината на сликата или промена на ширината на сликата или промена на висината и ширината на сликата. Аргументите на функцијата и нивните значења се:

- *source:* сликата чија големина сакаме да ја промениме
- *dim*: посакуваната нова димензија на сликата во форма на торка од вредности (висина, ширина)
- *fx*: Фактор за скалирање долж хоризонталната оска или X-оската
- *fy:* Фактор за скалирање долж вертикалната оска или Y-оската
- *interpolation*: процес на проценка на непознати вредости на пиксели во сликата, во склоп на функцијата има 3 методи за пополнување на овие пиксели
    - INTER_LINEAR- бинарно пополнување на пиксел со 0 или 1
    - INTER_CUBIC- пополнување со вредност добиена од 4x4 матрица соседни пиксели
    - INTER_LANCZOS4- Lanczos интерполација користејќи 8x8 матрица од соседи

## G. Plotly

Plotly [9] овозможува методи за исцртување на графици, правење на аналитики и статистики, користејќи некој од јазиците - Python, R, MATLAB, Perl, Julia, Arduino, и REST.

# Креирање на множествата од слики за тренирање и тестирање на моделот

Најпрво превземавме готова датотека од интернет во која има слики од човекови лица на кои се изразени различни емоции. Во датотеката сликите се претставени како дводимензионални низи од броеви кои ги претставуваат вредностите на секој пиксел од сликата, во овој случај тоа се вредности од 0-255 бидејќи сликите се дадени во GRAYSCALE color format (со еден канал за секој пиксел). На почеток креиравме фолдери за податоците, фолдер за тренирачкото и фолдер за тестирачкото множество. Потоа со помош на библиотеката NumPy ја креиравме почетната матрица исполнета со нули за при секое ново процесирање на слика да се почне од таа. Потоа, со итерација го земавме секој ред од датотеката и секоја вредност од редот (2304) ја стававме на соодветното место во матрицата. Притоа соодветното место, индексите на полето, ги добивавме така што на бројачот употребувавме floor division  за добивање на x индексот (редицата) додека пак module за добивање на y индексот (колоната) за соодветната вредност во низата. Сликата се креира од матрицата со помош на функција од класата Image од библиотеката PIL. Секоја слика потоа ја додававме на соодветниот фолдер во зависност од која емоција ја има човекот на сликата. Сликите ги поделивме така што 70% од нив се во тест множеството додека пак 30% во множеството за тренирање. Кодот за подготовка на сликите и креирањето на множествата слики се наоѓа во DataPreparation.py во склоп на проектот.

# Креирање на множествата од податоци

## I. Вовед во искористените функци за креирање на множествата од податоци

Со цел да ги искористиме максимално нашите примери за тренинг, ќе ги „зголемиме“ преку бројни случајни трансформации, така што нашиот модел никогаш нема да ја види двојно истата слика. Ова помага да се спречи overfitting и му помага на моделот подобро да генерализира.

Во овој проект ваквата манипулација со слики ни е овозможена со помош на библиотеката Keras, поточно со користење на *keras.preprocessing.image.ImageDataGenerator* класата која ни овозможува да кофигурираме случајни трансформации и функции за нормализација кои ќе се извршат врз секоја слика која ќе биде дел од множеството за тренирање или тестирање.

За множествата за тренирање и тестирање најпрво инастанциравме објект од оваа класа во кој го иницијализиравме параметарот rescale. *Rescale* е вредност која ќе биде помножена со секој влезен пиксел со што ќе се изврши нормализација на вредностите. Имено доколку имаме слики кои се во RGB color format секој нивни пиксел има вредности 0-255, но тие вредности се многу големи за нашиот модел да може брзо да ги процесира и во овој случај бојата не ни е важна туку имаме за цел да ги извлечеме емоциите кои се во вид на форми и

линии. Затоа потребно е вредностите да ги скалираме со фактор 1/255 за истите да добијат вредности помеѓу 0-1, на тој начин сите слики ќе бидат во GRAYSCALE color format.

```
train_datagen = ImageDataGenerator(rescale=1./255)
val_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(48,48),
    batch_size=batch_size,
    color_mode="grayscale",
    class_mode='categorical')

validation_generator = val_datagen.flow_from_directory(
    val_dir,
    target_size=(48,48),
    batch_size=batch_size,
    color_mode="grayscale",
    class_mode='categorical')
```

Потоа конструираме множества на слики со помош на генератори кои ќе го конструираат множеството од одреден фолдер од кој пикселите на сликите ќе се скалираат со претходно иницијализираниот фактор на скалирање. Ова е овозможено со функцијата *.flow_from_directory()* употребена врз претходо иницијализираните генератори. Во оваа функција освен аргументот за спецификација на патеката на фолдерот со слики од кои ќе биде конструирано множеството, ги иницијализиравме и следните аргументи

- *Target_size:* специфицира големина на секоја слика (48x48)
- *Batch_size:* специфицира број на слики кои ќе се земат одеднаш, во една итерација
- *Color_mode:* специфицира color format на сликите (grayscale)
- *Class_mode:* контролира подредување на класите (доколку не се постави, подредени се лексикографски)

Множествата за тренинг и тест ги конструираме во вид на генератори бидејќи како такви потоа ни служат како влезни аргументи во методите на моделот кој ќе го изградвме, тренираме и тестираме.

## II. Методологија за креирање на множествата со податоци

На почеток ги инцијализиравме променливите кои ни се потребни во продолжение на кодот, иницијализацијата во променливи е најпогодна бидејќи вредностите на истите често ги менувавме за да добиеме подобар модел. Потоа ги иницијализиравме генераторите со помош на кои подоцна ги креиравме потребните множества за тренинг и тест на нашиот модел. Иницијализацијата ја направивме со *ImageDataGenerator(rescale=1./255)*, каде параметарот е поставен ма вредност 1./255 со што секој пиксел ќе се скалира со овој фактор за да добиеме црно бели слики. Потоа со помош на функцијата *train_datagen.flow_from_directory()* ги креиравме множествата, влезните аргументи кои ги употребивме се патеката каде што се наоѓаат сликите, *target_size=(48,48)* големината на сликите, *batch_size=batch_size* количество на податоци

при секоја итерција, *color_mode="grayscale"* простор на бои на сликите, *class_mode='categorical'* сликите спаѓаат во повеќе класи. Истите аргументи, само со различна патека на податоци, ги користевме при креирање на тренинг и тест множеството бидејќи потребна е конзистентност на сликите (да се во ист формат) за успешно тренирање и тестирање на моделот.

# Креирање на моделот

## I. Вовед во искористените операции

Моделот го иницијализиравме како секвенцијален со намера да додаваме повеќе слоеви притоа креирајќи еден вид конволуциска невронска мрежа. Во продолжение додававме слоеви низ кои се процесираат податоците од сликите и се прави успешна предикција за нејзината класа. Најпрво ќе следи објаснување на наведените поими, а подоцна, подетално ќе ги објасниме чекорите при градењето на моделот.

### A. Sequential - градење на невронска мрежа

Секвенцен модел [10] е соодветен за обична група од слоеви, каде што секој слој има точно еден влез tensor и еден излез tensor. Овој модел не е соодветен во случаи кога нашиот модел или некој од слоевите, имаат повеќе влезови или повеќе излези, или пак кога имаме потреба од споделување на слоеви.

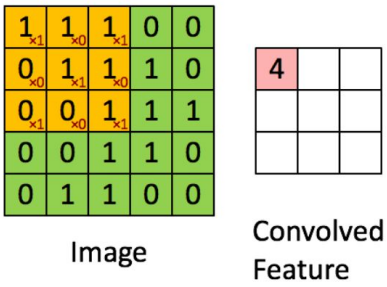
### B. Конволуција

Конволуција [11] од математичка гледна точка е математичка операција врз две функции, притоа креирајќи трета која го изразува влијанието на едната врз другата. Во нашиот случај таа се користи за да можеме да ја трансформираме оригиналната функција во друга форма преку која ќе можеме да добиеме повеќе информации за некоја слика. Конволуциите се користат веќе подолго време при процеси на обработка на слики, за истите да се заматат и изострат, но и да се извршуваат други операции, како што се подобрување на рабовите и релјефот, извлекување на карактеристики за сликата.

Конволуциската операција која тука претставува матрично множење се извршува за секој елемент (пиксел) на сликата. Каде што едната матрица е сликата, а другата е филтерот или јадрото (*kernel* - функција која ја дава сличноста помеѓу два податоци, врши операција која често се користи) што ја обработува сликата. Излезот од ова е конечната слика со применет филтер.
Доколку сликата е поголема од големината на филтерот, филтерот се лизга до различните делови на сликата, па конволуциската операција се извршува на секој од нив. Секој пат кога ќе го сториме тоа, генерираме нов пиксел во излезната слика.
Бројот на пиксели со кои го лизгаме јадрото е познат како чекор. Чекорот обично се зема да е 1, но може да се зголеми по потреба. Кога ќе се зголеми, можеби ќе треба да ја зголемиме големината на сликата за неколку пиксели за да се вклопи во јадрото на рабовите на сликата. Ова зголемување се нарекува полнење.

### C. Conv2D

Најчестиот вид на конволуција што се користи е 2D конволуцискиот слој, кој обично се нарекува conv2D [12]. Филтер или јадро во conv2D слој има висина и ширина. Тие се генерално помали од влезната слика и затоа ги преместуваме низ целата слика. Делот од сликата каде што се наоѓа филтерот се нарекува активно поле.
Conv2D филтрите се протегаат низ сите канали на сликата, и може да бидат различни за секој канал. Откако вртењата ќе се извршат индивидуално за секој канал, тие се додаваат за да се добие конечната слика со применета конволуција. Излезот од филтерот по конволуциската операција се нарекува мапа на карактеристики (feature map).
Секој филтер во конволуцискиот слој е иницијализиран со одредена дистрибуција по случаен избор (Нормална, Гаусва, итн.). Имајќи различни критериуми за иницијализација, секој филтер се тренира различно. Со ова се овозможува секој филтер да идентификува различни карактеристики. Доколку сите се иницијализирани слично, тогаш многу е веројатно дека двата филтри ќе научат да препознаваат исти карактеристики. Случајната иницијализација гарантира дека секој филтер учи да идентификува различни карактеристики, затоа секој од случајно генерираните филтри ќе препознава различна карактеристика со што секој слој кои ќе ги содржи овие филтри ќе може да ги распознава поголемиот дел од карактеристиките.
Секој од овие филтри се користи како влез во следниот слој во невронската мрежа. Ако има 8 филтри во првиот слој и 32 во вториот, тогаш секој филтер во вториот слој прима 8 влезови. Значи добиваме мапа на карактеристики со вкупно 32X8 карактеристики во вториот слој, при што секоја од 8 мапи на карактеристики, од еден филтер, се додаваат за да се добие по еден излез од секој слој. Всушност, секој филтер во слојот conv2D е матрица на броеви. Матрицата одговара на шема или карактеристика/својство што филтерот ја бара.
Conv2D слоевите се користат во првите неколку конволуциски слоеви на CNN, за да се извлечат едноставни features од сликата. Во минатото тие биле главните филтри кои се користеле и од нив биле направени повеќето од CNN. Денес, со напредувањето во конволуциските слоеви и филтрите, дизајнирани се пософистицирани филтри кои можат да служат за различни намени. Иако conv2D слоевите работат прилично импресивно, сепак имаат одредени ограничувања. Нивното најголемо ограничување е фактот дека нивното извршување чини многу. Голем conv2D филтер одзема многу време за пресметка, но и самата поделба во слоеви го зголемува количеството на пресметки. Најлесното решение за ова е да се намали големината на филтрите и да се зголемат чекорите. Но тоа тоа исто така го намалува ефективното рецептивно поле на филтерот и ја намалува количината на информации што може да ги фати.

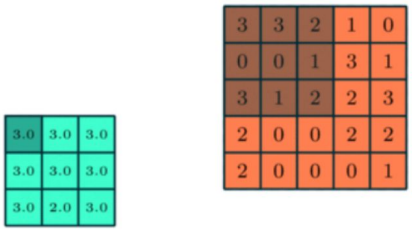


### D. Max Pooling Layer

Максималното здружување [13] (Max Pooling) е вид на операција што обично се додава на CNN следејќи ги индивидуалните конволуциски слоеви. Кога се додава на модел, максималното здружување ја намалува димензионалноста на сликите со намалување на бројот на пиксели во излезот од претходниот конволуционен слој. Односно, максималното здружување е операција која ја пресметува максималната или најголемата вредност во секој дел од мапата на карактеристики (матрицата), при што големината на делот се задава како параметар на функцијата и обично е многу помал од влезната слика (најчесто секој пиксел со неколку соседи во околина). Резултатите се намалени примероци или здружени мапи на карактеристики кои ја истакнуваат најприсутната карактеристика во секој дел, а не просечното присуство на одликата во случај на average pooling. Овој вид е подобар за проблемот на класификација на слики отколку average pooling.

Постојат неколку причини зошто додавањето на максимално здружување во нашата мрежа може да биде корисно.

1. Намалување на пресметковниот товар - Бидејќи максималното здружување ја намалува резолуцијата на дадениот излез на конволуционен слој, мрежата ќе разгледува поголеми области на сликата во исто време, што ја намалува количината на параметрите во мрежата, но и го намалува пресметковниот товар.
2. Намалување на прекумерно вклопување - Дополнително, максималното здружување може да помогне и во креирањето на по генерализиран модел. Интуицијата зошто работи максималното здружување е дека, за одредена слика, нашата мрежа ќе бара да извлече некои посебни карактеристики.




### E. Dropout

Кај невронските мрежи за длабоко учење веројатно брзо ќе дојде до overfitting доколку имаме тренирачко множество со малку примери.
Познато е дека збир од невронски мрежи со различни конфигурации на моделите, го намалуваат overfitting-от, но бараат поголеми ресурси за пресметување и одржување на повеќе модели.
Еден модел може да се искористи за симулирање на голем број различни мрежни архитектури, преку случајно испуштање на јазли за време на тренирањето. Ова се нарекува dropout и е многу пресметковно евтин и неверојатно ефикасен метод за регуларизација, намалување и спречување на overfitting, како и за подобрување на грешките при генерализацијата во длабоките невронски мрежи од сите видови.

Dropout [14] се имплементира на секој слој во невронската мрежа. Може да се користи со повеќето типови слоеви, како што се густи целосно поврзани слоеви, конволуциски слоеви и рекурентни слоеви како што е долгиот краткорочен мемориски мрежен слој. Може да се спроведе на кој било или на сите скриени слоеви во мрежата, како и на видливиот или влезниот слој, но не се користи на излезниот слој.
Воведен е и нов хиперпараметар кој што ја специфицира веројатноста со која се испуштаат излезите на слојот, или обратно, веројатноста со која се задржуваат излезите на слојот. Заедничка вредност е веројатност од 0,5 за задржување на излезот на секој јазол во скриениот слој и вредност близу до 1,0, како што е 0,8, за задржување на влезовите од видливиот слој. Dropout не се користи по тренинг, кога се прави предвидување со истренираната мрежа.

# II. Функции од библиотеката keras за моделот

### A. *Sequental()*

Секвентниот модел е соодветен за обична група од слоеви, каде што секој слој има точно еден влез tensor и еден излез tensor.

### B. *Conv2D(filters, kernel_size, strides, padding, data_format, dilation_rate, groups, activation, use_bias, kernel_initializer, bias_initializer, kernel_regularizer, bies_regularizer, activity_regularizer, kernel_constraint, bias_constraint)*

Овој слој креира конволуциско јадро кое е convolved со влезниот слој, со цел да креира tensor of outputs. Аргументи кои ги предадовме на оваа функција во склоп на проектот се:

- *filters*: дименионалноста на излезниот простор, односно бројот на излезни филтри во конволуцијата
- *kernel_size*: е цел број, или торка/листа од 2 цели броеви, кои ја специфицираат висината и ширината на 2Д конволуцискиот прозорец.
- *activation*: Активациска функција која ќе се користи. Ако не се прати вредност за неа, нема да има активациска функција

### C. *MaxPooling2D( pool_size, strides, padding, data_format)*

MaxPooling2D е функција која ги намалува влезовите, со тоа што ја зема максималната вредност од прозорецот, која е зададена како вредност на *pool_size*, за сите димензии долж оската на карактеристиката. Овој прозорец се поместува за одреден број чекори, внесени како вредност за аргументот *strides*, за секоја димензија. Излезот при користење на *padding=”valid”,* има *shape: output_shape = (input_shape - pool_size + 1) / strides)*. Додека пак, доколку вредноста на аргументот *padding* е *“same”* тогаш *output_shape = input_shape / strides.*

- *pool_size:* вредности во вид на цел број, или торка од 2 цели броеви, која што дефинира големина на прозорецот со кој ќе биде земен максимумот. Односно, (2, 2) ќе ја земе максималната вредност од 2x2 pooling прозорец. Доколку вредноста е еден цел број, тогаш прозорецот ќе има форма на квадрат.
- *strides*: вредности во вид на цел број, или торка од 2 цели броеви, или *None*. Специфицира колку далеку ќе се движи pooling прозорецот при секој чекор. Ако вредноста е None, тогаш ќе се движи за pool_size вредност.
- *padding*: дозволени вредности се или “valid” или “same”. “Valid” значи дека нема padding, а “same” значи дека padding-от ќе биде еднаков од сите страни, така што излезот ќе биде со истите димензии како влезот.
- *data_format*: вредноста е од тип стринг, или “channels_last” или “channels_first”. Channels_last соодветствува на влезовите со shape (batch, height, width, channels), додека пак channels_first соодветствува со влезовите со shape (batch, channels, height, width).

**D.** *Dense(units, activation, use_bias, kernel_initializer, bias_initializer, kernel_regularizer, bies_regularizer, activity_regularizer, kernel_constraint, bias_constraint)*

*Dense* ја имплементира операцијата *output = activation(dot(input, kernel) + bias),* каде што *activation* е активациската функција за елементот, *kernel* е тежинска матрица креирана од слојот, а *bias* е вектор креиран од слојот. Аргументи кои ги предадовме на оваа функција, кога ја искористивме во нашиот проект се:

- *Units*: позитивен цел број кој ја означува димензионалноста на излезниот простор
- *Activation*: активациска функција која ќе се употребува. Ако не е специфицирана, нема да се примени никаква активациска функција.

**E.** *model.compile()* - го конфигурира моделот за тренирање преку специфицирање на неговите карактеристики кои се земаат преку влезните аргументи на функцијата.

*model.compile(loss='categorical_crossentropy', optimizer=Adam(lr=0.0001, decay=1e-6), metrics=['accuracy'])*

Аргументи кои може да ги прими оваа функција се *optimizer*, *loss*, *metrics*, *loss_weights*, *weighted_metrics*, *run_eagrly*, *steps_per_execution*. Од кои при имплементација на оваа функција во проектот ние ги специфициравме

- *optimizer*: функција за оптимизација на моделот
- *loss*: функција која пресметува загуба на податоци на трениранот модел, може да биде една од веќе готовите или ние да креираме наша
- *metrics*: листа на метрики кои моделот ќе ги пресметува додека е во тренинг или тест фазата

**F.** *keras.optimizers.Adam()* или како аргумент *optimizer=Adam()* во функцијата *model.compile()*

*Adam(lr=0.0001, decay=1e-6)* - повикот на оваа готова функција креира оприимизатор на моделот кој користи алгоритам за оптимизација основан на градиент од прв ред, истовремено со прилагодливи проценки во зависност од дадениот момент. Имено функцијата прима повеќе аргументи *lr, beta_!, beta_2, epsilon, decay,* за потребите на нашиот проект овој оптимизатор го имплементиравме со следните параметри

- *lr*: рата на учење, големина на чекор во секоја итерација со кој моделот моделот сака да дојде до минимална загуба
- *decay*: намалување на ратата на учење со зголемување на итерациите, овој параметар е погоден за да се избегне промашување на минимумот на функцијата на чинење, сакаме да земаме помали чекори како што се приближуваме до минимумот

**G.** *Categorical_crossentropy* - како функција на загуба која ја пресметува crossentropy помеѓу вистинските класи и предикциите на множество на податоци. Имено во структура на невронска мрежа при тренирање се користи функција на загуба за секој предвиден примерок притоа се прави сума на ентропијата за секоја класа содржана во моделот. Оваа функција на загуба се пресметува со формулата:

$$C(h_\theta(x), y) = \begin{cases} -\log(h_\theta(x)) & ако\ y = 1 \\ -\log(1 - h_\theta(x)) & ако\ y = 0 \end{cases}$$

$$-\frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K}[y_k^i\ log h_\theta(x^i)_k + (1 - y_k^i)\ log(1 - h_\theta(x^i)_k)]$$

Притоа во нашиот пример ние за последниот слој користиме *softmax* активациска функција во невронската мрежа која прво се пресметува и на крај се пресметува вкупната загуба со *categorical cross entropy*. Ја користиме categorical бидејќи нашиот модел има повеќекласен излез.

**H.** *model.load_weights('model.h5')*- на крај трениранииот модел го зачувавме со помош на оваа функција која всушност ги чува тежините меѓу секој од слоевите во невронската мрежа кои ќе ни служат при предикција со истиот.

# III. Методологија за креирање на моделот

Моделот го иницијализиравме како секвенцијален со намера да додаваме повеќе слоеви притоа креирајќи еден вид конволуциска невронска мрежа. Во продолжение додававме слоеви низ кои се процесираат податоците од сликите и се прави успешна предикција за нејзината класа. Вкупно нашиот модел има 4 конволуциски слоеви кои се иницијализирани со функцијата *Conv2D()*. Сите конволуциски слоеви имаат исти димензии за јадрената функција (3,3) како и иста активациска функција *relu(max(x,0))*, но со различна димензионалност на излезот. Во првиот слој диензионалноста е најмала со 32, потоа во вториот е 63 и во последните два 128, ваквата конструкција на димензионалноста е со цел во секој слој да има филтри со ист влез но може на крај да препознае различна карактеристика. Колку повеќе филтри толку поголема шанса за распознавање на повеќе карактеристики.

После секој конволуциски слој, со исклучок помеѓу првиот и вториот, додадовме по еден *pooling* слој со кој ја намаливме големината на податоците со што добивме поголема процесирачка брзина и поголема робусност на карактеристиките кои треба да се распознаваат. Овие слоеви избравме да користат *MaxPooling* со аргумент (2,2) што значи за секои 4 соседни пиксели ќе има еден кој ќе биде оној со најголемата вредност од тие 4. За да избегнеме overfitting користевме L2 регуларизазација со што ќе се намалат оние вредности на параметри кои не ни се од значење. Исто така одбравме да правиме *dropout* што значи дека имавме два слоја во кои вредностите се сетираат на нула со фрекфенција 0.25. Потоа за на моделот да му додадеме *Dense* слој ,иницијализиран со 1024 димензионалност на излезот и relu активациска функција, мора претходно да направиме *flatten* на податоците со што на влез на слојот *Dense* ќе му предадеме еднодимензионална низа. Последниот слој исто така е *Dense* но со излез 7 бидејќи тоа е бројот на класи и со активациска функција *softmax*. Овие *Dense* слоеви ни помагаат за да се асоцираат карактеристиките меѓу себе, имено одредени карактеристики се корелирани и со овие слоеви дозволуваме дени на други да имаат влијание кое ќе помогне во учењето на нивното однесување. Исто така во последнниот слој користиме softmax активациска функција, која се користи во полиномна логистичка регресија. *Softmax* често се користи како последна активациска функција во невронската мрежа, со цел да го нормализира излезот од мрежата.

По креирање на сите слоеви, моделот го завршивме со иницијализирање на одредени функционалности кои тој ќе ги користи со помош на функцијата *model.compile()*. Параметрите кои ги дадовме на влез се *loss='categorical_crossentropy'* функцијата на губење, *optimizer=Adam(lr=0.0001, decay=1e-6)* оптимизатор со рата на учење и нејзино намалување, *metrics=['accuracy']* и за време на тренирањето на моделот ја следевме неговата точност.

На крај моделот го тренираме со помош на функцијата *model.fit_generator()* чии влезни аргументи се тренинг множеството, *steps_per_epoch=num_train // batch_size* колку чекори ќе има во секоја епоха, *epochs=num_epoch* број на епохи, тест (валидациско множество), *validation_steps=num_val // batch_size* чекори при тестирање на моделот.

```
model = Sequential()

model.add(Conv2D(32, kernel_size=(3, 3), activation='relu',kernel_regularizer=regularizers.l2(0.0001),input_shape=(48,48,1)))
model.add(Conv2D(64, kernel_size=(3, 3), activation='relu',kernel_regularizer=regularizers.l2(0.0001)))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(128, kernel_size=(3, 3), activation='relu', kernel_regularizer=regularizers.l2(0.0001)))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(128, kernel_size=(3, 3), activation='relu', kernel_regularizer=regularizers.l2(0.0001)))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Conv2D(7, kernel_size=(1, 1), activation='relu', kernel_regularizer=regularizers.l2(0.0001)))
model.add(Conv2D(7, kernel_size=(4, 4), activation='relu', kernel_regularizer=regularizers.l2(0.0001)))
model.add(Flatten())
model.add(Activation("softmax"))
model.summary()
```

```
filepath = os.path.join("./emotion_detector_models/model_v6_{epoch}.hdf5")

checkpoint = keras.callbacks.ModelCheckpoint(filepath,
                                             monitor='val_acc',
                                             verbose=1,
                                             save_best_only=True,
                                             mode='max')
callbacks = [checkpoint]

model.compile(loss='categorical_crossentropy',optimizer=Adam(lr=0.0001, decay=1e-6),metrics=['accuracy'])
nb_train_samples = 28709
nb_validation_samples = 3589
epochs = 100
model_info = model.fit_generator(
    train_generator,
    steps_per_epoch=nb_train_samples // batch_size,
    epochs=epochs,
    callbacks = callbacks,
    validation_data=validation_generator,
    validation_steps=nb_validation_samples // batch_size)
plot_model_history1(model_info)
model.save_weights('model1.h5')
```

# Тренирање и предикција

## I. Методологија за на моделот

*model.predict(image, batch_size, verbose, steps, callbacks, max_queue_size, workers, use_multiprocessing)*

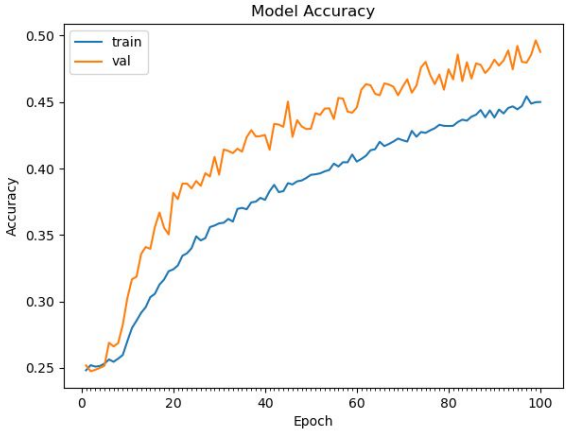
*model.predict()* генерираме излез (предвидување) за дадените (влезни) примероци. Оваа пресметка, т.е. предвидување, се извршува во batches, при што методот е дизајниран да има одлични перформанси дури и кога влезните податоци се огромни. Овој метод го повикуваме со еден влезен параметар кој всушност претставува сликата за која сакаме да го извршиме предвидувањето, притоа вредноста на истата е преставена како numpy низа. Во нашиот проект оваа функција беше имплементирана само со слика како влезен аргумент.

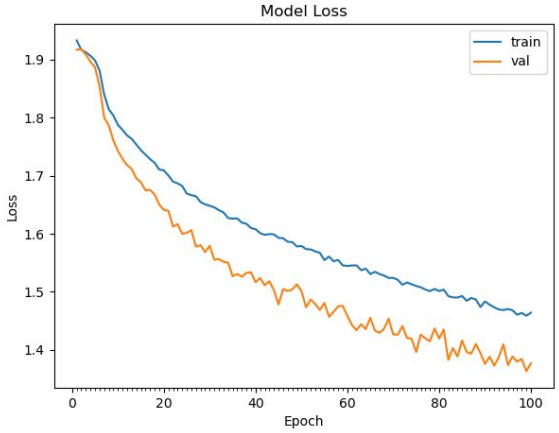
## II. Методологија за предикција

По успешно тренирање и тестирање на моделот направивме многубројни тестирања со истиот со цел да видиме дали тој прави добри предикиции за нови слики. По вчитување на сликата со помош на *face_recognition.face_locations(face_image)* го лоциравме делот на сликата со човеково лице и креиравме нова слика само од лицето со димензии 48,48 и во gray простор на бои со помош на cv2 библиотеката. На крај сликата ја пуштивме низ нашиот модел од конволуциска невронска мрежа преку оваа функција *model.predict(face_image)* и ја добивме лабелата на предвидената класа за истата.

```
model.load_weights('model1.h5')
cv2.ocl.setUseOpenCL(False)
emotion_dict = {0: "Angry", 1: "Disgusted", 2: "Fearful", 3: "Happy", 4: "Neutral", 5: "Sad", 6: "Surprised"}
face_image = cv2.imread("TestImg/fear.jpg")

try:
    face_locations = face_recognition.face_locations(face_image)
    top, right, bottom, left = face_locations[0]
    face_image = face_image[top:bottom, left:right]
except:
    pass

face_image = cv2.resize(face_image, (48, 48), cv2.INTER_AREA)
face_image = cv2.cvtColor(face_image, cv2.COLOR_BGR2GRAY)

plt.imshow(face_image)
print(face_image.shape)
face_image = np.reshape(face_image, [1, face_image.shape[0], face_image.shape[1], 1])

prediction = model.predict(face_image)
maxindex = int(np.argmax(prediction))


print(emotion_dict[maxindex])
```

За време на тренирње и тестирање на моделот, ја следевме неговата точност и загубата на податоци. Потоа со помош на библиотеката `matplotlib.pyplot` ги визуелизиравме резултататите, што ни овозможи да видиме дали моделот успешно класифицира и дали дошло до преголемо прилагодување кон тренинг множеството. Во продолжение е прикажан график при тренирање на моделот во 100 епохи, при што се забележува дека истиот има прилично исти перформанси за двете множества со што може да заклуциме дека е добро генерализиран.



# Референци

[1] - *https://pillow.readthedocs.io/en/stable/*
[2] - *https://numpy.org/*
[3] - *https://pandas.pydata.org*
[4] - *https://www.tensorflow.org/*
[5] - *https://keras.io/*
[6] - *https://www.mulesoft.com/resources/api/what-is-an-api*
[7] - *https://pypi.org/project/face-recognition/*
[8] - *https://docs.opencv.org/master/index.html*
[9] - *https://plotly.com/*
[10] - *https://keras.io/guides/sequential_model/*
[11] - *https://www.saama.com/different-kinds-convolutional-filters*
[12] - *https://www.tensorflow.org/api_docs/python/tf/keras/layers/Conv2D*
[13] - *https://deeplizard.com/learn/video/ZjM_XQa5s6s*
[14] - *https://machinelearningmastery.com/dropout-for-regularizing-deep-neural-networks/*