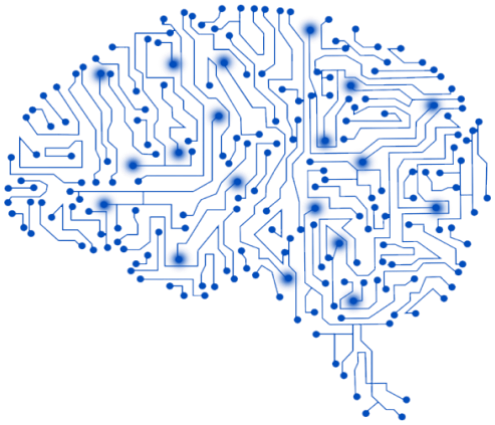
Универзитет „Св. Кирил и Методиј“ во Скопје
ФАКУЛТЕТ ЗА ИНФОРМАТИЧКИ НАУКИ И КОМПЈУТЕРСКО ИНЖЕНЕРСТВО

Детекција на дијабетична ретинопатија врз база на фундус фотографии со помош на конволуциски невронски мрежи

Семинарска работа по предметот Вештачка интелигенција



Ментор:
Проф. Др. Соња Гиевска

Изработила:
Биљана Димитрова, 211098

Содржина

Апстракт

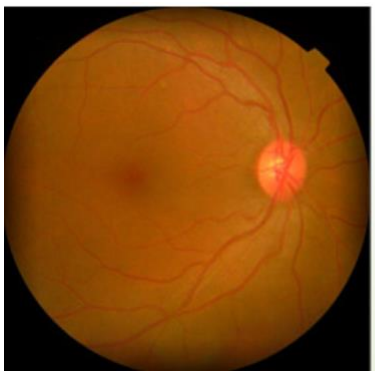
Во светот каде што брзото напредување на технологиите континуирано ги менува динамиките на истражувањето и развојот, како предмет на истражување решив да ја вклучам конволуциски невронски мрежи за разрешување на предизвици во областа на медицината поточно детекцијата на дијабетичната ретинопатија којашто е болест која се јавува како последица кај луѓето коишто страдаат од дијабетес. Зголемениот шеќер ги афектира крвните садови на ретината коишто испуштаат течност во макулата при што резултира со заматен вид во крајни случаеви и слепило. Протечената крв или течност се јавува во форма на дамки наречени лезии преку коишто се дијагностицира стадиумот на болеста. Од исклучителна важност е болеста да се дијагностицира во рана фаза се со цел што порано да почне да се дејствува против нејзино распространување. Човечкото око во одредени случаеви не може да ги забележи лезиите па за таа цел направив експеримент за модел на конволуциска невронска мрежа којшто врши класификација на стадиумот на болеста врз база на фундус фотографии од ретината.

Вовед

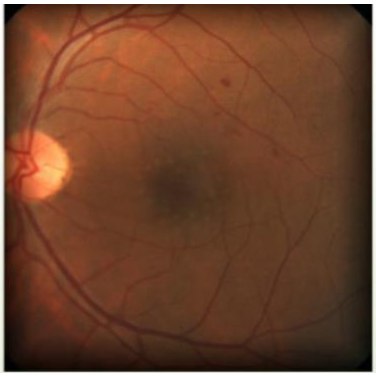
Дијабетичната ретинопатија (ДР) претставува сериозна последица кај луѓето коишто страдаат од дијабетес како резултат на оштетување на крвните садови од превисоко ниво на шеќер во крвта. Се јавуваат два типа на последици од овој тип на болест:

- Дијабетичен макуларен едем - крвните садови испуштаат течност во макулата при што кај пациентот се причинува заматен вид
- Неоваскуларен глауком – енормен раст на крвните садови којшто е причинтел на блокада на течноста којашто ја лачат очите коешто може да доведе до значително оштетување на видот во критични ситуации дури и слепило.

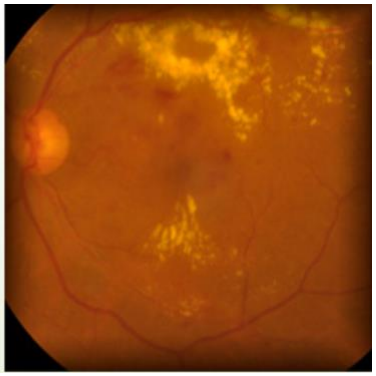
Од исклучителна важност е болеста да се детектира во најраната фаза се со цел да не дојде до некоја посериозна последица како што е целосна загуба на видот. Мануелната дијагноза од страна на офталмолозите бара повеќе време и детална анализа, бидејќи во одредени случаеви лезиите можат да бидат незабележливи што за жал може да доведе до погрешна дијагноза. За таа цел се работи на развој на системи коишто би предвидувале еден од можните пет стадиуми.



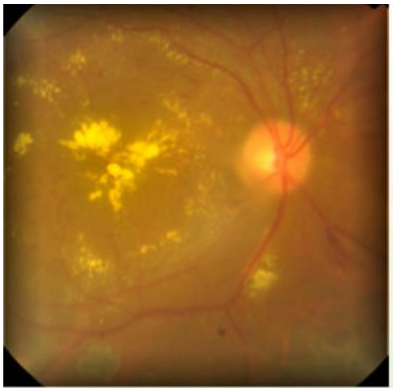
Нема знаци на ДР



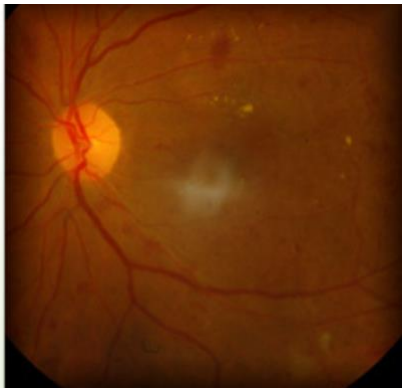
Рана фаза



Умерена фаза



Сериозна фаза



Пролиферативна фаза

Слика 1. Петте стадиуми на дијабетична ретинопатија

Така што главната цел на ова истражување е да се развие и евалуира CNN-базиран систем за детекција на ДР врз база на фундус фотографии. Со користење на анотирани податоци, ќе обучам модел кој автоматски ќе детектира промени и патолошки состојби во ретината на лицата со дијабетес. Ваквиот модел би можел да обезбеди брза и доста прецизна детекција, што би им овозможило на офталмолозите рана интервенција и ефикасно управување со дијабетичната ретинопатија. Во следните секции ќе бидат опфатени техниките коишто се употребени при развојот на моделот како и самата структура на конволуциската невронска мрежа вклучително и евалуацијата.

Техники за претпроцесирање

Пред да се употребат фотографиите за тренирање на моделот потребно е да поминат низ одредено претпроцесирање се со цел да се оптимизира тренирањето и ефикасноста на моделот. Од особена важност е претпроцесирањето особено за медицински тип на фотографии поради тоа што може да се постигне зголемена експресивност на одредени детали при подготовка за понатамошна поефективна анализа. Исто така овозможува подобрување на специфични карактеристики коишто се есенцијални за подготовка на податоците пред вметнување во невронската мрежа. Не сите модели поддржуваат различни големини или пак непропорционални фотографии така што потребно е да се обрне внимание и на големината на податоците. Конкретно за конволуциските мрежи потребно е сите фотографии да имаат иста големина, не е задолжително но е препорачливо да имаат и квадратен сооднос но со одредена претпазливост се со цел да не се предизвика дисторзија на фотографиите бидејќи тоа би резултирало со значително намалување на перформансите на CNN такашто би можеле да се изостават некои детали коишто би имале зголемена важност. Конкретно во мојот случај употребив неколку техники коишто овозможуваат подобрување на квалитетот на податоците и ефикасно тренирање на моделот. Започнав со промена на големината на фотографиите бидејќи првичната големина на поголемиот дел од фотографиите беше 4752 × 3168 но сепак големината не е соодветна за тренирање на CNN модел така што одлучив драстично да ја намалам големината на 256 x 256 при што би помогнало во спречување на формирање на прекумерен број на параметри и креирање на конзистентна податочна структура.

```
def read_images(file_paths):
    results=[]
    for img_path in file_paths:
        img = cv2.imread(img_path)
        if img is not None:
            #Resizing the image
            resized_img = cv2.resize(img,(256,256))
            results.append([resized_img,img_path])

    return results
```

Слика 2. Читање на фотографиите и промена на нивните големини во сооднос 256 x 256

На слика 2 е претставен кодот на функцијата во програмскиот јазик Python чијашто функционалност читање на фотографиите во соодветен формат во вид на матрица како и промена на големината во посакуваниот сооднос 256 x 256. При што на влез се дадени патеките на фотографиите, додека пак како резултат се враќаат прочитаните фотографии со променета големина.
Се со цел зачувување на најважните карактеристики од фотографиите па такашто бидејќи се работи за фундус фотографии доста видлива е црната рамка околу ретината, така што тоа беше мојот следен чекот зачувување само на најважните детали. Сето тоа е остварено преку хардкодирана функција којашто е прикажана во продолжение:

```
DEL_PADDING_RATIO = 0.02
THRESHOLD_LOW = 7
THRESHOLD_HIGH = 180
MAX_TARGET_SIZE = (256, 256)

def del_black(img1):
    if img1.ndim == 2:
        img1 = np.expand_dims(img1, axis=-1)
    width, height = (img1.shape[1], img1.shape[0])

    (left, bottom) = (0, 0)
    (right, top) = (img1.shape[1], img1.shape[0])

    padding = int(min(width, height) * DEL_PADDING_RATIO)

    for i in range(width):
        array1 = img1[:, i, :]
        if np.sum(array1) > THRESHOLD_LOW * array1.shape[0] * array1.shape[1] and \
                np.sum(array1) < THRESHOLD_HIGH * array1.shape[0] * array1.shape[1]:
            left = i
            break
    left = max(0, left - padding)

    for i in range(width - 1, 0 - 1, -1):
        array1 = img1[:, i, :]
        if np.sum(array1) > THRESHOLD_LOW * array1.shape[0] * array1.shape[1] and \
                np.sum(array1) < THRESHOLD_HIGH * array1.shape[0] * array1.shape[1]:
            right = i
            break
    right = min(width, right + padding)
```

Слика 3. Отстранување на црната рамка околу сликата во код – прв дел

Најпрво се поставуваат одредени граници коишто треба да ги задоволи левата граница и десната граница на сликата. Првата константа del_padding_ratio означува дека вредноста на пополнувањето на сликата по остранувањето на црната рамка би било 2% од димензијата, додека пак THRESHOLD_LOW и THRESHOLD_HIGH ги означуваат границите до коишто треба да достигнува збирот од вредностите на пикселите. Потоа во функцијата најпрво се проверува дали фотографијата е дводимензионална поточно дали е црно бела доколку се исполни условот се додава дополнителна димензија. Откако се иницијализирани сите потребни променливи се итерира низ колоните при што се поставуваат левата и десната граница на сликата чиишто збир на вредности на пиксели треба да биде измеѓу предефинираните прагови (thresholds). Во продолжение е даден кодниот сегмент во којшто се бараат долните и горните граници на сликата при што во овој случај за детекција за долната граница се итерира од горе кон долу низ секој ред на сликата, поствувајќи го како долна граница првиот ред којшто го задоволува условот за зададениот опсег. Истото се случува и со поставувањето на горна граница разликата е во тоа што се итерира во спротивна насока од долу кон горе низ секој ред на сликата, при што како и за долната слика се додава падинг. Доста важно е да се обрне внимание на центрирањето на регионот којшто преставува важен чекор при проверката дека главната содржина на сликата е позиционирана во средината на резултантната квадратна област. Оваа техника се користи за да се осигура конзистентно позиционирање на

релевантната област во квадратната слика, без разлики во распределбата на пикселите околу неа. Исто така, центрирањето ги олеснува и ја подобрува интерпретацијата на резултатите. Кога ќе се разгледуваат или анализираат резултантните слики, релевантната област е центрирана, што го прави процесот на интерпретација поедноставен. Понатаму се врши скалирање на сликата при што се пресметува факторот за скалирање се со цел да се постигне посакуваната големина. Со оваа процедура гарантираме дека резултантната слика е квадратна и е составена од главната содржина, при што се води сметка за падингот и предефинираната големина.

```
for i in range(height):
    array1 = img1[i, :, :]
    if np.sum(array1) > THRESHOLD_LOW * array1.shape[0] * array1.shape[1] and \
            np.sum(array1) < THRESHOLD_HIGH * array1.shape[0] * array1.shape[1]:
        bottom = i
        break
bottom = max(0, bottom - padding)

for i in range(height - 1, 0 - 1, -1):
    array1 = img1[i, :, :]
    if np.sum(array1) > THRESHOLD_LOW * array1.shape[0] * array1.shape[1] and \
            np.sum(array1) < THRESHOLD_HIGH * array1.shape[0] * array1.shape[1]:
        top = i
        break
top = min(height, top + padding)

center_x = (left + right) // 2
center_y = (bottom + top) // 2

left = max(0, center_x - MAX_TARGET_SIZE[0] // 2)
right = min(width, center_x + MAX_TARGET_SIZE[0] // 2)
bottom = max(0, center_y - MAX_TARGET_SIZE[1] // 2)
top = min(height, center_y + MAX_TARGET_SIZE[1] // 2)

scale_factor = min(MAX_TARGET_SIZE[0] / (right - left), MAX_TARGET_SIZE[1] / (top -
 bottom))

return cv2.resize(img1[bottom:top, left:right, :], None, fx=scale_factor, fy=scale_factor)
```

Слика 3. Отстранување на црната рамка околу сликата во код – прв дел

Како следен чекор во процесот на претпроцесирање на слики е повикување на процедура којашто ги содржи сите претпроцесирачки методи коишто дел од нив се рачно искодирани додека останатите се готови методи коишто се повикуваат од библиотеката open cv (Open Computer Vision library) којашто воедно претставува една од најголемите библиотеки за проблеми од областа на компјутерската визија. Така што со помош на методот cv2.cvtColor() ја конвертирам сликата во grayscale се со цел да се олесни тренирањето на моделот бидејќи сликите во боја имаат три канали додека пак grayscale фотографиите само еден канал така што се намалува бројот на параметри коишто треба да ги научи моделот, згора на се полесни се за обработка тоа би значело помала процесорска моќ. Главните семантички детали се зачувани во фотографијата во овој случај така што не би преставувало проблем промената во grayscale фотографија.

```
img=cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

Сметам дека најглавниот дел од процесот на претпроцесирање претставува употребата на CLAHE (Contrast Limited Adaptive Histogram Equalization) методот којшто врши хистограмска еквализација [1] на фотографијата поради тоа што ги нагласува деталите коишто се важни за класификацијата, поточно лезиите. Целиот тој процес се врши така што најпрво се прави хистограм во којшто се претставува фреквенцијата на интензитетите во сликата при што потоа се нормализира хистограмот се со цел добивање на функција на распределба при што се

[1] Хистограмската еквализација е техника во обработката на слики која се користи за подобрување на контрастот и видливоста на детали во сликата.

креира една кумулативна функција преку којашто се врши пресликувањето на пикселите. На тој начин се добива резултантната слика чии пиксели се добиени како резултат од пресликувањето со помош на кумулативната функција.

```
clahe = cv2.createCLAHE(clipLimit=5.0, tileGridSize=(8, 8))
img=clahe.apply(img)
```

Како едни од најважните параметри кога се применува CLAHE се clipLimit и tileGridSize, clipLimit го ограничува констрастот во суштина се поставува еден праг на контраст којшто не смее да се надмине, додека пак кај вториот параметар главната суштина е поделбата на сликата во квадрати со големина од 8 x 8 коишто се изминуваат додека се применува CLAHE филтерот. Последен чекор при обработката на фотографиите е таканареченото Гаусово замаглување (Gaussian Blur) којшто има функција да го ублажи таканаречениот шум.

```
img= cv2.GaussianBlur(img, (3,3), 0)
```

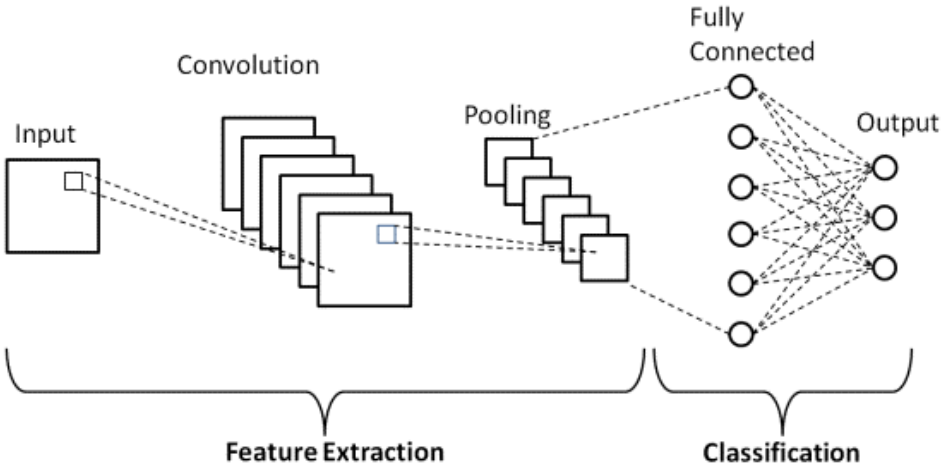
Како параметри се поставуваат фотографијата врз која е потребно да се изврши гаусовиот филтер, како втор параметар се поставува големината на јадрото (кернел) којшто ќе итерира низ матрицата со големина од (3,3) при што ќе бидат вклучени соседни пиксели во радиус од 1 пиксел од секој пиксел. Како последен параметар е поставена стандардната девијација на гаусовиот филтер којашто во овој случај е поставена на 0, што означува автоматски избор на стандардната девијација базирана на големината на јадрото. Исходот од целосното претпроцесирање е прикажано во продолжение:

Слика 4. Споредба помеѓу оригиналната и обработената фотографија

Како што може да се забележи на обработената слика успешно е елиминирана црната обвивка околу ретината со исклучок на аглите секако тоа е направено со цел да не се изгуби формата. Деталите се доста поистакнати отколку на оригиналната фотографија што придонесува до поефикасна класификација. Целиот код којшто беше напишан при процесот на обработка на сликите се наоѓа во фајлот preprocessing.ipynb, додека претпроцесираните фотографии се во формат на зипуван фајл под името preprocessed.zip.

Конволуциски невронски мрежи

Конволуциските невронски мрежи (CNNs) претставуваат доста моќен модел во областа на длабокото учење, дизајниран специфично за обработка и анализа на визуелни податоци. Со изразена способност за распознавање на шаблони во слики, CNNs најчесто се користат во задачи како класификација на слики, детекција на објекти и семантичко сегментирање. Токму поради тие причини сметав дека е најсоодветно да се користат CNN како модел во проблемот на истражување, пред се поради својата архитектура којашто е доста погодна кога станува збор за проблеми од компјутерската визија. Во продолжение е претставена архитектурата на CNN:



Слика 5. Архитектура на конволуциска невронска мрежа

Како што може да се согледа, станува збор за поделба на слоевите коишто се одговорни да извршат екстракција на карактеристиките од сликите и слоеви кадешто сите неврони се меѓусебно поврзани чијашто одговорност е класификацијата или типот на проблемот за којшто станува збор, но во конкретниот случај станува збор за класификација. При што слоевите воглавно се поделени на влезен слој, скриени слоеви, излезен слој кадешто секој слој си има своја одговорност. Влезниот слој има за задача само да ги пренесе податоците кон следниот слој, единствено на што треба да се внимава е да се посочи структурата на податоците поточно за колку димензионални податоци се работи и слично. Влезниот слој потоа ги пренесува податоците на првиот скриен слој којшто не мора да значи дека е и последен. Во било кој тип на невронска мрежа може да имаме произволен број на скриени слоеви се додека имаме подобрување во евалуацијата на моделот, секако тоа може да варира во зависност од типот на моделот. Во скриените слоеви воглавно се вршат главните пресметки како и тежинските операции, тука спаѓаат: Конволуциските слоеви, Активациски функции и Pooling слоевите коишто се задолжени да ги извлечат најважните карактеристики.Во суштина главната цел на CNN е да се намали големината на параметрите така што ќе се зачуваат најважните карактеристики коишто придонесуваат кон зголемување на прецизноста па носители на тоа својство се конволуциските и pooling слоевите.

Конволуциските слоеви вршат екстракција на карактеристики со примена на конволуцивни операции на влезните податоци, под конволуциски операции се подразбира специјалниот тип на множење на матрици. Конволуциските операции се одвиваат на тој начин што најпрво се дефинира кернелот/филтерот којшто се пополнува со рандом тежини коишто не се менуваат во текот на изминувањето на сликата при што се одвива таканаречената операција конволуција којашто всушност означува множење на матрици но првиот елемент од првата

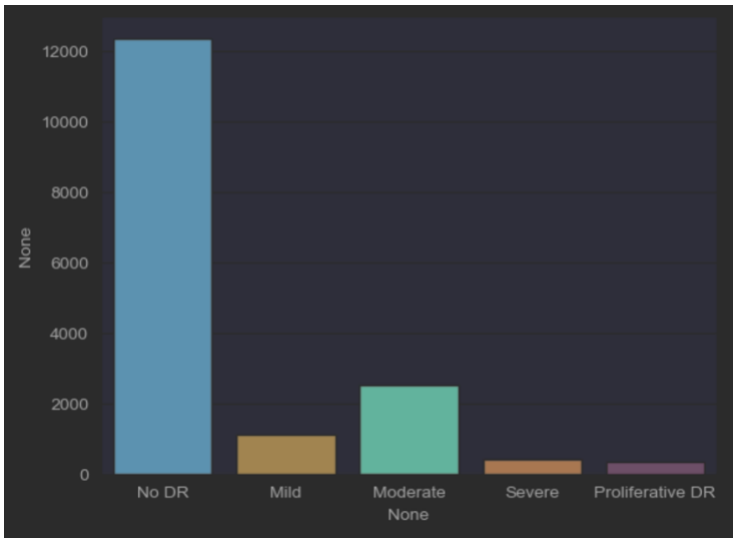
матрица се множи со последниот елемент од втората матрица, се додека не се измине целата, резултатот од множењето се запишува во таканаречена мапа на карактеристики (feature map). Притоа за намалување на димензијата на feature map се користат Pooling слоевите коишто ја намалуваат комплексноста и ги извлекуваат карактеристиките со најголема тежина.Најчесто се користи MaxPooling при што исто така се дефинира големината на филтерот којшто се применува најчесто на излезот од конволуциските операции и се зема најголемата вредност од регионот којшто моментално се изминува со тој прозорец, сето тоа значително ја намалува комплексноста и димензијата на feature map.

Fully connected слојот има за цел да изврши финална обработка на влезните карактеристики и да донесе краен излез на моделот. Во пракса пред да се пратат излезните податоци коишто произлегле како резултат на MaxPooling слојот потребно е да поминат низ таканаречено израмнување(Flatten) при што податоците се трансформираат во еднодимензионален вектор. Притоа секој неврон во fully connected слојот е поврзан со секоја вредност во векторот, ова значи дека постојат тежински параметри помеѓу секоја можна комбинација на неврони. Излезот се добива со примена на тежините на влезните податоци и додавање на биас (наклонетост), при што за проблеми на класификација најчесто се користи активациската функција softmax бидејќи на излез се добиваат веројатности за секоја од можните класи (вредности коишто се наоѓаат во опсегот помеѓу 0 и 1).

Имплементација

Небалансираност на податочното множество

Во првичното множество со податоци, кое вклучува вкупно 16803 анотирани фундус фотографии, се констатира небалансираност. Оваа небалансираност е детектирана преку анализа на податоците, како што се прикажува во следниот barplot:



Слика 6. Дистрибуција на класите

Поради тоа што се работи за детекција на заболување сепак потешко оди анотацијата и самиот процес на колекција на податоци, поради тоа нормално е во такви случаеви да се соочуваме

со ваков тип на податочни множества коишто се прилично предизвикувачки поради својата небалансирана природа, поради тоа беше клучно да се употреби некоја од техниките за небалансираност. По поделбата на податочното множество на тренирачко и на тестирачко подмножество во сооднос 80-20% промените се рефлектираат на тренирачкото множество како и техниките со небалансираноста се со цел да се сочуваат оригиналните податоци коишто не треба да бидат вклучени во процесот на тренирање. Најпрво сметав дека е ефикасно да се употреби undersampling на податоците анотирани со класата No DR поточно да се отстранат дел од нив од причина што разликата со останатите класи по бројот на податоци е прилично драстична така што би можело да се случи фаворизирање на единствена класа.

```
condition = (Train_Y['No DR'] == 1)
rows_to_remove = Train_Y[condition].sample(frac=0
 .4, random_state=42).index
Executed at 2024.02.08 21:26:17 in 23ms

Train_Y=Train_Y.drop(rows_to_remove)
Executed at 2024.02.08 21:26:17 in 38ms
```

Слика 7. Процесот на undersampling

Процесот во код се одвива така што најпрво се преземаат оние редици коишто го исполнуваат условот поточно фотографиите кај коишто не е дијагностицирана ДР притоа се избираат рандом 40% од примероците коишто треба да се избришат се со цел да се постигне поголема балансираност помеѓу податоците. Следно решив да употребам oversampling на класите коишто се анотирани со 1,3,4 се со цел подобрување на процесот на тренирање со тоа што на овој начин и класите коишто се помалку застапени малку повеќе ќе дојдат до израз.
Целиот процес се одвива во долунаведениот код, кадешто варијаблите I,j,k всушност означуваат колку фотографии се додадени, ограничени се на одреден број за да не биде цело податочно множество само дуплирано, потоа стандарно се додаваат редици во тренирачкото множество.

```
for index in range(0,len(train_images)):
  if i<300 and Train_Y['Proliferative DR'][index]==1:
    train_images_oversampling.append(train_images[index])
    new_row={
             'No DR':Train_Y['No DR'][index],'Mild':Train_Y['Mild'][index],
             'Moderate':Train_Y['Moderate'][index],
             'Severe':Train_Y['Severe'][index],
             'Proliferative DR':Train_Y['Proliferative DR'][index]}
    train_y_oversampling=train_y_oversampling._append(new_row,ignore_index=True)
    i=i+1
  if j<500 and Train_Y['Severe'][index]==1:
    train_images_oversampling.append(train_images[index])
    new_row={
             'No DR':Train_Y['No DR'][index],'Mild':Train_Y['Mild'][index],
             'Moderate':Train_Y['Moderate'][index],
             'Severe':Train_Y['Severe'][index],
             'Proliferative DR':Train_Y['Proliferative DR'][index]}
    train_y_oversampling=train_y_oversampling._append(new_row,ignore_index=True)
    j=j+1
  if k<700 and Train_Y['Mild'][index]==1:
    train_images_oversampling.append(train_images[index])
    new_row={
             'No DR':Train_Y['No DR'][index],'Mild':Train_Y['Mild'][index],
             'Moderate':Train_Y['Moderate'][index],
             'Severe':Train_Y['Severe'][index],
             'Proliferative DR':Train_Y['Proliferative DR'][index]}
    train_y_oversampling=train_y_oversampling._append(new_row,ignore_index=True)
    k=k+1
  if k>700 and i>300 and j>500:
    break
```

Слика 8. Процесот на oversampling

Потоа како следна техника решив да употребам data augmentation поточно креирање на вештачки примероци кај коишто е додаден одреден шум се со цел подобра генерализација. Во следниот коден сегмент е прикажан процесот на податочна аугментација каде што фокусот паѓа на ротацијата на фотографијата. Се генерира случаен агол за ротација во опсегот од -20 до 20 степени, потоа со користење на OpenCV, се создава ротациона матрица (rotation_matrix) со соодветниот агол и центарот на ротација при што сликата се ротира со користење на функцијата cv2.warpAffine.
Следно, се креира нов ред во податочното множество (new_row), каде што информациите за оригиналната слика се копираат, но со додавање на "_augmented" кон името на новата слика. Овој нов ред се додава на крајот на податочното множество.
На крај, функцијата враќа новиот ред што претставува аугментирана слика.

```python
def data_augmentation(index):
    img = train_images[index]
    angle = np.random.uniform(-20, 20)
    rotation_matrix = cv2.getRotationMatrix2D((img.shape[1] // 3, img.shape[0] // 3), angle, 1.0)
    img = cv2.warpAffine(img, rotation_matrix, (img.shape[1], img.shape[0]))
    new_row={'image':Train_Y['image'][index]+"_augmented",'No DR':Train_Y['No DR'][index],
     'Mild':Train_Y['Mild'][index], 'Moderate':Train_Y['Moderate'][index],
     'Severe':Train_Y['Severe'][index], 'Proliferative DR':Train_Y['Proliferative DR'][index]}
    train_images.append(img)
    return new_row
```

Executed at 2024.02.08 21:26:57 in 105ms

```python
for subset in [index_augmentation_1,index_augmentation_2,index_augmentation_3,
 index_augmentation_4]:
    for index in subset:
        Train_Y=Train_Y._append(data_augmentation(index),ignore_index=True)
```

Executed at 2024.02.08 21:27:05 in 4s 656ms

Слика 9. Процес на податочна аугментација

На следната фотографија е прикажан крајниот резултат од процесот со користење на дел од техниките за справување со небалансирани податочни множества.

	<unnamed>
No DR	5957
Mild	3192
Moderate	3385
Severe	1348
Proliferative DR	992

Моделирање на конволуциска невронска мрежа

Во оваа секција ќе биде детално објаснет процесот на моделирање на конволуциската невронска мрежа и образложен изборот на хиперпараметрите коишто се од круцијално значење. Првично одбрав TensorFlow како основна рамка за машинско учење, искористувајќи го за неговата голема флексибилност и способност да работи со големи и комплексни модели. Притоа одбирајќи Keras како библиотека која значително го олеснува процесот на изградба на модел, давајќи интуитивна контрола над дизајнирањето на невронските мрежи. Во продолжение е дадена структурата на моделот:

```
model=Sequential()
model.add(Conv2D(96, (3, 3), padding='valid', input_shape=(256,256,3)))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(64, (3, 3), padding='valid', input_shape=(256, 256, 3)))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(64, (3, 3), padding='valid', input_shape=(256, 256, 3)))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(32, (3, 3), padding='valid', input_shape=(256, 256, 3)))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dense(units=5, activation='softmax'))
model.summary()
```

При градењето најпрво се започнува со конволуциски слој којшто е конфигуриран со бројка од 96 неврони се со цел подобра екстракција на детали од влезниот слој при што во понатамошните делови сето тоа ќе биде филтрирано и на тој начин само најважните карактеристики би останале, исто така се дефинира и големината на јадрото/филтерот којшто во случајов е поставен на стандардна големина од (3,3) поради тоа што фотографиите се со големина од 256 x 256 така што сметам дека е сосема доволно голем филтерот за податоци со таа големина. Исто така се наоѓа и параметарот padding='valid' што всушност означува "без дополнување", поточно операцијата за конволуција ќе се примени без дополнување со нули околу влезната слика. По секој конволуциски слој преферирам да користам BatchNormalization() поради тоа што го оптимизира процесот на тренирање, спречува до некој размер overfitting и што е најважно помага во внатрешното ковариjантно поместување така што без да изрши дисторзија на сликата се грижи за тоа сите фотографии да имаат идентична распределба така што сите воглавно сите влезни податоци за секој слој би биле дистрибуирани околу иста средна вредност и стандардна девијација[2]. Така што од тие причини сметав дека би било погодно да се користи слој кадешто ќе се врши нормализација но не на ниво на податок туку на ниво на batch (пакет) чијашто големина се дефинира при процесот на тренирање. Потоа се додава одредена нелинеарност на моделот со помош на активациската функција relu така што сите вредности коишто имаат вредност помали од 0 ги елиминира додека останатите се предаваат во понатамошните слоеви, исто така ова е доста корисно кога станува збор за справување со проблемот за vanishing gradient којшто се појавува во случаеви кога градиентот станува толку мал што не придонесува воопшто кон подобрување на перформансите на моделот. Следниот слој е MaxPooling којшто придонесува до редуцирање на просторните димензии (ширина и висина) на сликите во излезниот тензор од претходниот конволуциски слој. Начинот на функционирање на слојот е така што најпрво се дефинира параметарот pool_size=(2, 2) при што се дефинира големината на филтерот така што за секој 2x2 предел, MaxPooling операцијата ја зема максималната вредност од 4 вредности во пределот и ја враќа како излез. Ова значи дека за секои 2x2 пиксели, само една вредност ќе се задржи, што ја редуцира просторната резолуција на сликата. Моделот даваше подобри резултати при постепено намалување на невроните во конволуциските слоеви тоа би можело да биде како резултат на подобра генерализација и поефикасно тренирање поточно моделите

[2] Колку варираат податоците од средната вредност

коишто имаат преголем број на параметри премногу се навикнуваат на тренирачкото множество и ја губат способноста за генерализација. По завршените конволуциски операции, од каде што добив резултантни податоци, пред да се пренесат податоците во Dense слојот, сметав дека е соодветно да се додаде Dropout слој којшто е конфигуриран да случајно деактивира (постави на 0) околу 20% од излезите од предходниот слој се со цел да се спречи overfitting. Потоа, излезот од Dropout слојот се пренесува на Flatten слојот којшто има за цел да ги претвори податоците во еднодимензионална структура, што е неопходно пред да ги предадеме на Dense слојот овозможувајќи правилно поврзување. Во Dense слојот се конфигурирани 128 неврони при што секој од нив е поврзан со секој излез од претходниот слој при што секоја врска (тежина) има соодветен параметар кој се учи за време на тренирањето на моделот. Воглавно, Dense слојот се користи за учење на тежини (weights) и биаси (biases) кои се користат за адаптирање и трансформирање на влезните податоци. Потоа пред предавање на излезниот Dense слој се врши BatchNormalization како и додавање на нелинеарност поточно активациска функција (relu). На крајот од невронската мрежа се наоѓа излезниот слој којшто има 5 неврони или бројот на класи бидејќи со активациската функција softmax се добиваат вредности помеѓу 0 и 1, при што секоја вредност ја застапува веројатноста за излезот да биде една од петте класи.

Следен чекор е компајлирањето на моделот кадешто се користи функцијата model.compile од Keras библиотеката пред да започне процесот на тренирање. Оваа функција дефинира начинот на кој моделот треба да се ажурира и учи со користење на оптимизатор, функција на загуба и метрики за следење на перформансите на моделот. Како оптимизатор којшто има функција да ги ажурира тежините на моделот го одбрав Adam поради тоа што даваше најдобри резултати во споредба со другите оптимизатори и доста поефикасно тренирање. Притоа многу важен параметар сметам дека е функцијата на загуба бидејќи го дефинира критериумот што моделот ќе го минимизира при тренирањето. Оваа функција е математички израз што оценува колку добро или лошо моделот прави предвидувања во однос на реалните вредности. Најдобри резултати даваше CategoricalCrossentropy но само минимално подобри резултати во споредба со CategoricalFocalCrossentropy, но сепак глобално подобри резултати се добиваа со првичната функција на загуба.

```
model.compile(optimizer=keras.optimizers.Adam(),
              loss=keras.losses.CategoricalCrossentropy(),
              metrics=[keras.metrics.Precision(),
              keras.metrics.Recall()])
```

Како метрики ги одбрав precision и recall поради тоа што се работи за небалансирано множество сметав дека е потребно да се разгледа процесот на тренирање и од двете страни.

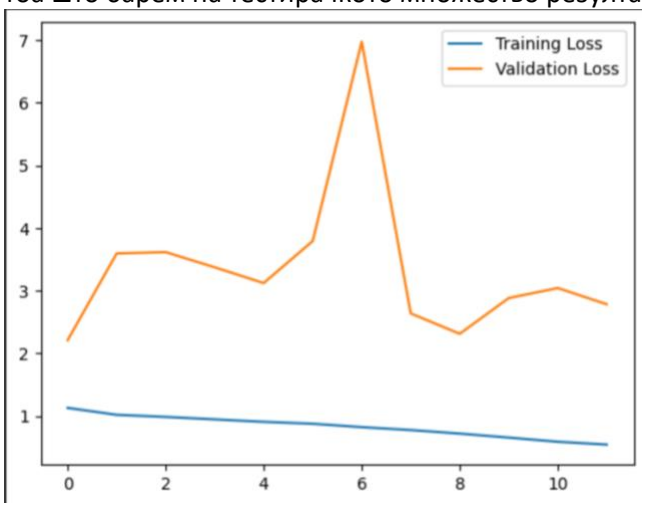
```
history = model.fit(Train_X, Train_Y, validation_split=0.2, epochs=12,
batch_size=32)
```

Во функција model.fit() потребно е да се дефинираат фотографиите со коишто ќе се тренира моделот и секако да се наведат лабелите коишто претходно се one hot енкодирани. Во параметарот validation_split се дефинира процентот од тренирачкото множество коешто ќе се додели на валидациското множество се со цел да се проценат перформансите на моделот на податоци коишто не се видени. Исто така како едни од поважните параметри се бројот на епохи и batch size. При што во мојот случај со оглед на небалансираното множество и малиот број на податоци поголем број на епохи предизвикува overfitting и зголемување на вредноста

добиена од функцијата на загуба така што на тој начин само ги влошуваше перформансите при што како идеална вредност за број на епохи е 12, додека за batch_size вредноста 32 бидејќи со намалување на вредноста доаѓаше до проблем на недоволна генерализација и фаворизирање на само класата со најголем број на примероци, можеби тоа е резултат на недоволно примероци од останатите класи во таканаречените пакети, такашто една класа е доминантна и останатите класи не се исперцепирани. Целиот код од овој процес се наоѓа во фајлот model.ipynb.

Резултати

Во овој дел, ќе бидат детално објаснети резултатите од моделот кадешто главна цел е детекција на стадиумот на болеста. Како метрики во процесот на ревидирање на резултатите добиени од моделот ги искористив: прецизност, точност, AUC, одвиз, F1 мерка. Секоја од метриките си дава посебно значење на резултатите при што неопходно е да се земе во предвид небалансираното податочно множество. Исто така извршив и визуелизација во однос на функцијата на загуба којашто прикажува знаци на overfitting но не толку драстично поради тоа што барем на тестирачкото множество резултатите се во ред.



Според графикот грешката постепено се стреми кон нулата на тренирачкото множество, но на валидациското множество не се добиваат толку добри резултати.

Во продолжение е прикажан класификацискиот извештај добиен преку функцијата classification_report() од библиотеката sckit.learn којашто дава доста прегледни резултати.

	precision	recall	f1-score	support
0	0.74	0.96	0.83	2426
1	0.14	0.01	0.02	230
2	0.27	0.05	0.08	534
3	0.17	0.05	0.07	85
4	0.23	0.16	0.19	76
accuracy			0.71	3351
macro avg	0.31	0.25	0.24	3351
weighted avg	0.60	0.71	0.63	3351

Метриките покажуваат доста добри резултати за податоците анотирани со класа 0 поточно фундус фотографии коишто не укажуваат на знаци на ДР, резултатите за останатите класи се значително полоши во однос на нултата класа, иако вредноста за точност е во ред. Метриката

за прецизност ни укажува на бројот на вистински позитивни класи поделен со збирот на вистински позитивни и лажни позитивни, поради ниските резултати на останатите класи тоа значи дека моделот има тенденција да предвидува лажно позитивни инстанци. Мерката за одзив укажува на способноста на моделот да ги идентификува сите вистински позитивни инстанци. Пресметана е како бројот на вистински позитивни делен со збирот на вистински позитивни и лажни негативни. Моделот успешно предвидува кога пациентот не страда од ДР но кога станува збор за детекција на стадиумот на болеста не се покажува баш најдобро. Ф1 мерката всушност претставува просек помеѓу прецизноста и одзивот. Пресметана е како 2 * (precision * recall) / (precision + recall). Точноста дава генерално во ред резултати но сепак не е крајно релевантен фактор кога станува збор за ваков тип на податочно множество. Како метрика којашто ги евалуира перформансите на моделот ја вклучив и метриката AUC (Area Under The Curve) чијашто вредност ја добив со помош на функцијата roc_auc() од библиотеката sckit.learn и изнесува *0.6636967144459668.* Тоа значи дека класификаторот во скоро 66% од случаевите би ја предвидел точната позитивна класа наместо негативната. Доколку вредноста беше околу 0.5 во тој случај тоа означува дека моделот го рандомизира излезот, во случај кога вредноста се наоѓа близу 0.7 во тој случај во ред се предвидувањата но сепак има простор за подобрување. Кодот за добивање на вредноста од AUC метриката е даден во продолжение:

```
from sklearn.metrics import roc_auc_score
auc_roc = roc_auc_score(np.argmax(Test_Y.values, axis=1), preds2, average='macro',multi_class='ovr

[ ] print("Multiclass AUC-ROC:", auc_roc)

    Multiclass AUC-ROC: 0.6636967144459668
```

Врз база на добиените резултати дефинитивно сметам дека има простор за подобрување особено во техниките за справување со небалансираноста на множеството како и во делот за претпроцесирање кадешто ќе можат лезиите на окото да бидат поистакнати се со цел моделот подобро да ги разликува фотографиите бидејќи во првите три фази разликите помеѓу фундус фотографиите се тешки за согледување така што сметам дека е потребно посебно во тој дел да се подобри во смисла на зголемување на бројот на анотирани фотографии обележани како фундус фотографии коишто покажуваат рани знаци на ДР.

Заклучок

Моделот постигна задоволителна точност, особено при детекцијата кај пациенти коишто не страдаат од дијабетична ретинопатија. Сепак, постојат предизвици во точноста и одзивот на другите класи, што може да биде предмет за идни истражувања и подобрувања.

Дисбалансот помеѓу класите беше идентификуван како потенцијален предизвик, и можеби би било корисно истражување на техники за справување со небалансираните податоци. Дополнително, анализата на грешките и потребите за подобрувања во перформансите на моделот може да биде корисна за идни итерации. Но генерално резултатите од моделот се задоволителни кога би се земале сите фактори во предвид и може да послужи како добра основа за понатамошни надградби и истражувања на оваа тема.

Референци

- Diabetic Retinopathy Fundus Image Classification and Lesions Localization System Using Deep Learning
- Automatic detection of 39 fundus diseases and conditions in retinal photographs using deep neural networks
- eyewiki-Diabetic Retinopathy
- Deep learning in ophthalmology: The technical and clinical considerations
- A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way
- Class Imbalance Strategies