

**ТЕХНОЛОГИЧНО УЧИЛИЩЕ ЕЛЕКТРОННИ СИСТЕМИ  
към ТЕХНИЧЕСКИ УНИВЕРСИТЕТ - СОФИЯ**

**ДИПЛОМНА РАБОТА**

Тема: Система за изготвяне на снимка без хора от статичен видеоклип.

Дипломант:  
*Николай Станишев*

Научен ръководител:  
*Александър Чернаев*

**С О Ф И Я**

**2 0 1 8**



**ТЕХНОЛОГИЧНО УЧИЛИЩЕ  
ЕЛЕКТРОННИ СИСТЕМИ  
към ТЕХНИЧЕСКИ УНИВЕРСИТЕТ - СОФИЯ**

Дата на заданието: 06.11.2017 г.

Утвърждавам:.....

Дата на предаване: 06.02.2018 г.

/проф. д-р инж. Т. Василева/

## **ЗАДАНИЕ**

### **за дипломна работа**

на ученика Николай Георгиев Станишев 12 А клас

1. Система за изготвяне на снимка без хора от статичен видеоклип.

2.Изисквания:

2.1 Локализиране на хората от снимката.

2.2 Премахване на хората от снимката.

2.3 Комбиниране на многото снимки без хора.

3.Съдържание 3.1 Обзор

3.2 Същинска част

3.3 Приложение

Дипломант :.....

Ръководител:.....

/Александър Чернаев/

Директор:.....

/ доц. д-р инж. Ст. Стефанова /

## Отзив на Научният ръководител

Настоящата дипломна работа отговаря на изискванията в заданието и може да бъде допусната до защита.

Дипломант :.....

*Николай Станишев*

Научен ръководител:.....

*Александър Чернаев*

## Увод

Целта на Дипломната работа е да се създаде система, която изработва снимка без хора от статичен видеоклип, като изготвената снимка трябва да бъде само на фона.

Системата би могла да се използва за направата на снимка на много посещавана забележителност, около която непрекъснато има хора.

Проблемът за разрешаване се разделя на две части. Първата част е проблемът с откриването на хората, а втората част е намирането на информация за заместване на хората.

Откриването на хора на снимка е един от важните проблеми с които се сблъсква Компютърното зрение. Компютърното зрение в последните няколко години се е развило много благодарение на развитието на Конволюционните невронни мрежи. Примерни приложения за места в които се използва Компютърно зрение са самоуправляващите се коли, самоуправляващите се роботи, лицевото разпознаване и в много други свери, в които се изисква компютъра да "види" човека.

Намирането на информация да се замести човек може да се намери в някой от другите кадри от статичният видеоклип, да бъде генерирана на база на някой алгоритъм използващ машинно самообучение или да бъде генерирана на база околното съдържание. За целта на Дипломната работа ще бъде използван първия вариант с намирането на липсващата информация на някой от другите кадри.

Задачите на Дипломната работа са системата да предостави интерфейс за потребителя в който той да може да изпраща статичен видео клип, който системата да може да получава и след това да го разделя на кадри като за всеки от кадрите да се открият хората, които присъстват на него. След като имаме тази информация трябва да се премахнат хората от кадрите и последната задача преди да се върне готовата снимка на потребителя е да се комбинират многото кадри без хора до една готова снимка на фона.

# Глава Първа

## 1 Проучване

### 1.1 Съществуващи решения

#### 1.1.1 Откриване на обект на снимка

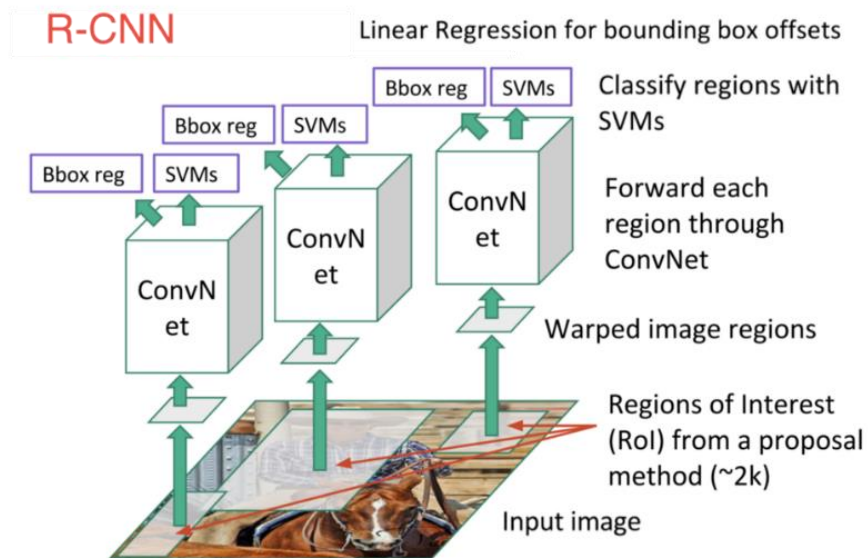
##### 1.1.1.1 Хистограми от Ориентирани Градиенти (ХОГ) за откриване на хора (HOG Person Detector)

ХОГ е трениран с помощта на Подпомагащи Векторни Машини(ПВМ). Чрез ПВМ се създава функционален вектор, който репрезентира целият човек. Той е с ширина 64 пиксела и дължина 128 пиксела. След като е обучен функционалният вектор ХОГ използва плъзгащ се детектор. Чрез плъзгащият се детектор функционалният вектор обикаля снимката. За всяка позиция на прозореца се прави предположение дали там присъства човек. За откриването на хора с различни размери снимката бива намалявана и увеличавана.<sup>[1]</sup>

##### 1.1.1.2 Регионална Конволюционна невронна мрежа (Р-КНМ) (R-CNN)

Р-КНМ е съставена от три стъпки:

1. Сканиране на входната снимка за възможни обекти използвайки алгоритъм наречен Избираемо Търсене. Този алгоритъм разделя входната снимка на много региони, като за всеки от тях генерирайки приблизително 2000 предположения.
2. Пускане на Конволюционната невронна мрежа за всеки от предложените региони.
3. Вземане на изхода от Конволюционната невронна мрежа, който минава през:
  - 3.1. ПВМ за да класифицира региона.
  - 3.2. Линейна регресия да определи Ограничителната кутия.<sup>[2]</sup>

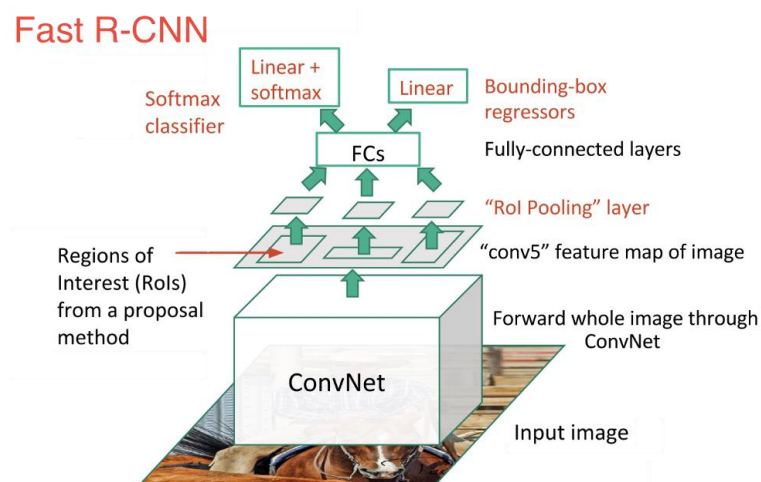


Фигура 1.1 Р-КНМ

### 1.1.1.3 Бърза Р-КНМ (Fast R-CNN)

Бързата Р-КНМ е наследника на Р-КНМ. Бързата Р-КНМ прилича много на предшественика си, но успява да подобри скоростта на правене на предположения благодарение на:

1. Извършване на извличане на свойствата на снимката преди правенето на Избираемо търсене. Благодарение на това Конволюционната невронна мрежа се пуска веднъж вместо 2000 пъти.
2. Заместване на ПБМ със слоя softmax. Това е продължава Невонната мрежа вместо да е отделен модел.<sup>[2]</sup>

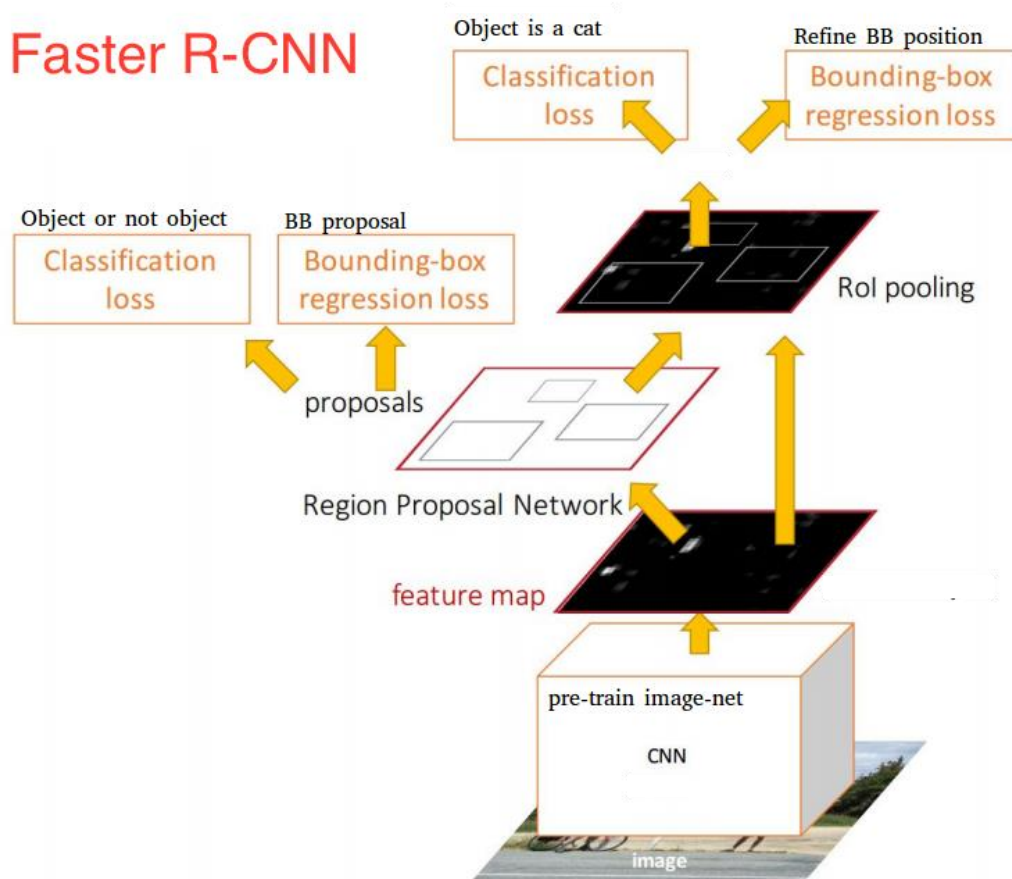


Фигура 1.2 Бърза Р-КНМ

#### 1.1.1.4 По-бърза Р-КНМ (Faster R-CNN)

По-бързата Р-КНМ е наследник на Бързата Р-КНМ. Основната цел на По-бързата Р-КНМ е да замени бавното Избираемо търсене с бърза Невронна мрежа - Региона мрежа за предложения (РМП). Ето как работи РМП:

- На последния слой от Конволюционналната невронна мрежа се добавя плъзгащ се прозорец с големина 3x3, който обикаля картата на свойства и им намаля дименсиите.
- За всяко положение на плъзгащият се прозорец се генерират няколко възможни региона базирани на анкерни кутии.
- Всеки от предложените региони съдържа:
  - Предложения клас
  - Четири координата, които репрезентират Ограничителната кутия.<sup>[2]</sup>

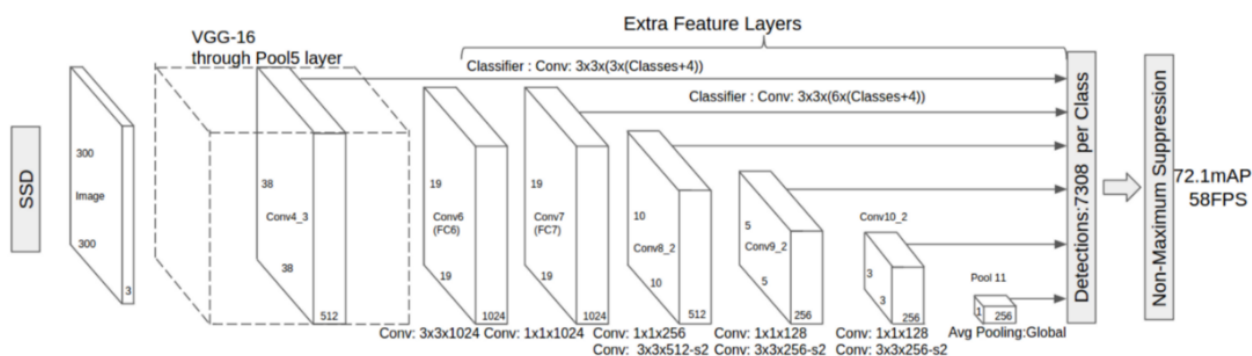


Фигура 1.3 По-бърза Р-КНМ

### 1.1.1.5 Детектор с единичен изстрел (ДЕИ) (SSD (Single Shot Detector))

ДЕИ осигурява по-голяма скорост в сравнение в По-бързата Р-КНМ. Това се постига чрез два отделни модела в две отделни стъпки, които отговарят за правенето на предположения за района и класификацията на региона. Първо те използват РМП за да генерират регионите, които ни интересуват. Следващата стъпка е да се използват изцяло свързани слоеве или позиционни прецизни Конволюционнални слоеве да класифицират генерираните региони. ДЕИ се справя с това в единичен изстрел като едновременно прави предложение за ограничителните кутии и за класовете. Алгоритъма на ДЕИ е:

1. Преминаване на снимка през серия от Конволюционнални слоеве, които предават няколко групи от карти със свойства за различни размери.
2. За всяко от местоположенията в картите на свойствата се използва  $3 \times 3$  Конволюционнален филтър за пресмятане на малки групи от анкерни кутии.
3. За всяка от кутиите се прави предположение за:
  - 3.1. Ограничителната кутия.
  - 3.2. Предположения клас.
4. По време на тренирането се съчетават истинските кутии с тези, които са предположени на база на стойността на Пресичане върху Обединение. Предположенията, които ще се броят ще са тези, които имат най-висока стойност на Пресичане върху Обединение и увереност  $> 0.5$ . Този метод се нарича Non-Max Supression.<sup>[2]</sup>



Фигура 1.4 ДЕИ



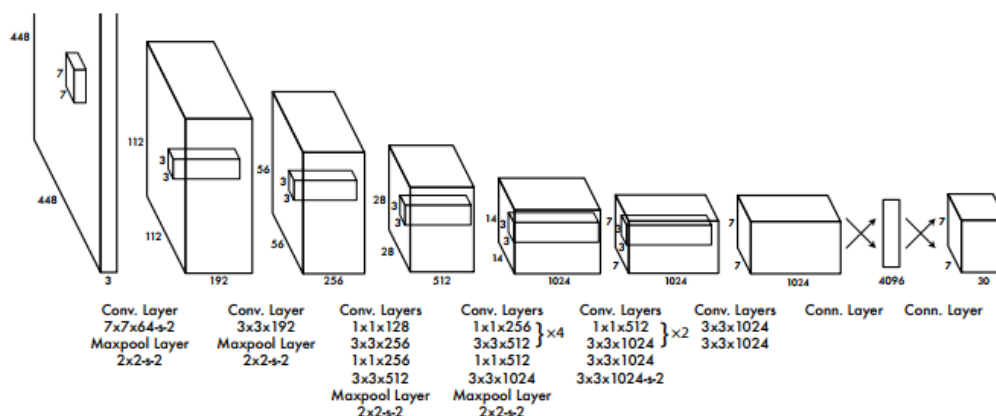
### 1.1.1.6 Ти Само Гледаш Веднъж (ТСГВ) (YOLO (You Only Look Once))

Името Ти Само Гледаш Веднъж идва от това, че снимката преминава само веднъж през една Конволюционна невронна мрежа.

Благодарение на една Конволюционна невронна мрежа, която прави предположения за Ограничаващите кутии и за вероятността на класовете директно за цялата снимка за едно преминаване през Мрежата, проблемът става регресия. Предимството на това е, че цялото откриване на обекта става само с една Невронна мрежа и то може да бъде оптимизирано от край до край.

Основното му предимство пред другите подходи за откриване на обекти на снимка е скоростта. Скоростта на основният модел е 45 кадъра в секунда, а на по-малък вариант на мрежата - Бързо ТСГВ е 155 кадъра в секунда. Независимо от бързината на този подход той успява да се справи по-добре от повечето от другите подходи.

Входът на Конволюционната невронна мрежа е снимка с резолюция 448x448, а изхода е матрица с дименсии 7x7x30. Изхода репрезентира решетка с големина 7x7 като всяка клетка от тази решетка е вектор с големина 30. Съдържанието на този вектор са координатите на обекта, чиито център е открит в клетката(x, y), височината и широчината на този обект(h, w), увереността на предположението за две Ограничителни кутии и 20 вероятности за 20-те различни класа.<sup>[3]</sup>



Фигура 1.5 ТСГВ архитектура

## **1.1.2 Премахване на човек от снимка**

### **1.1.2.1 Adobe Photoshop**

#### **1.1.2.1.1 Метод чрез наслагване на много снимки**

Adobe Photoshop има функция наречена Снимково наслагване, която комбинира група от снимки направени от един и същи ъгъл, но с различно качество или съдържание. Веднъж комбинирани снимките могат да бъдат обработени до получаването на желаната снимка без нежелано съдържание или шум върху нея. За добри резултати снимките съдържани в групата снимки трябва да имат еднакви дименсии и предимно подобно съдържание, примери за подобни групи със снимки са серия снимки направени от фиксирана гледна точка или серия от кадри от статична видео камера.<sup>[4]</sup>

#### **1.1.2.1.2 Метод чрез освежаване на съдържанието**

Adobe Photoshop има функция наречена Освежаване на съдържанието. Чрез тази функция избираме регион на снимката, който искаме да премахне. След избирането на региона за премахване функцията за Освежаване на съдържанието запълва избрания регион с подобни пиксели на съдържанието около него.<sup>[5]</sup>

### **1.1.2.2 Movavi Photo Editor**

Movavi Photo Editor е софтуер за редактиране на снимки.

Чрез него може да се премахват нежелани хора от снимка. Процесът за премахване на човек от снимка е:

1. Добавяна на снимка.
2. Маркиране на нежелания обект на снимката.
3. Натискане на бутона за изтриване на човека.
4. Запазване на снимката.<sup>[6]</sup>

## **1.2 Подходи и технологии**

### **1.2.1 Машинно самообучение**

Машинното самообучение е способността на компютрите да могат да се учат от данни.

Ето една по-обща дефиниция:

"Machine learning is the field of study that gives computers the ability to learn without being explicitly programmed."

- Arthur Samuel, 1959

И още една по инженерно ориентирана:

"A computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E."

- Tom Mitchell, 1997<sup>[7]</sup>

Машинното самообучение се използва за извличане на знание от данни. То е пресечна точка на статистиката, изкуствения интелект и компютърната наука. Приложението му през последните години е навлязло много силно в ежедневието ни.<sup>[8]</sup>

### **1.2.2 Контролирано машинно самообучение**

Данните за трениране предоставени на алгоритъма за Контролираното Машинно самообучение включват в себе си и отговорите, наречени етикети.<sup>[7]</sup> То се използва, когато искаме да направим предположение на база на някакви данни. Основните проблеми, с които се сблъсква Контролираното Машинно самообучение са регресия и класификация. За регресията целта ни е да правим непрекъснати предположения, а при класификацията целта ни е да направим предположение за някакъв клас, който присъства в някакъв списък от предварително зададени класове.<sup>[8]</sup>

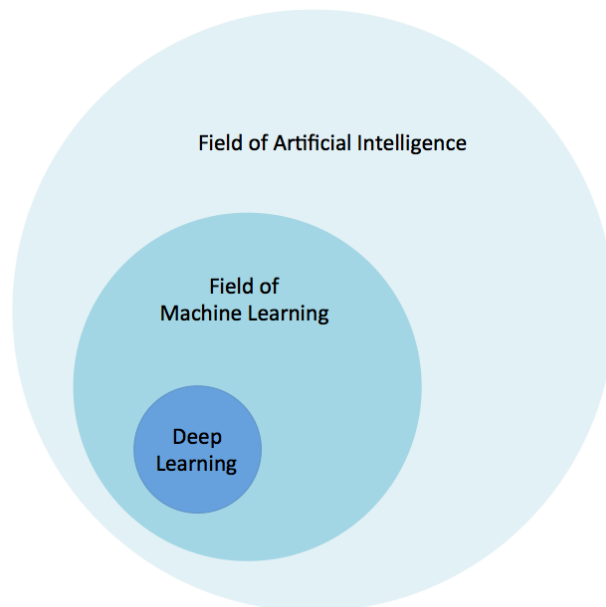
### **1.2.3 Безнадзорно машинно самообучение**

Безнадзорното Машинно самообучение включва в себе си всички видове Машинно самообучение, в които данните за трениране нямат етикети и системата няма учител, който да насочва учещия се алгоритъм. Целта на учещия се алгоритъм в този случай е да извади знания от данните, които са му зададени на входа.<sup>[8]</sup>

### **1.2.4 Дълбоко самообучение**

Дълбокото самообучение е част от Машинното самообучение, която използва Изкуствени невронни мрежи частично вдъхновени от структурата на невроните намиращи се в човешкия мозък.<sup>[9]</sup> Изкуствените невронни мрежи са в сърцевината на Дълбокото самообучение. Те са

гъвкави, мощни и мащабируеми, което ги прави идеални за големи и изключително сложни проблеми свързани с Машинното самообучение.<sup>[7]</sup>



Фигура 1.5 Връзката между Дълбокото самообучение, Машинното самообучение и Изкуствения интелект

Дълбокото самообучение е приложимо в много области свързани със снимки, текст, видео, говора и зрението. Освен многото области, в които е приложимо успехът му идва и от големите количества данни, които са достъпни и от увеличението на изчислителната мощност.

Изкуствените невронни мрежи са вдъхновени от изучаването на централната нервна система на бозайниците. Всяка мрежа е съставена от взаимосвързани неврони, организирани в слоеве, които обменят информация.<sup>[9]</sup>

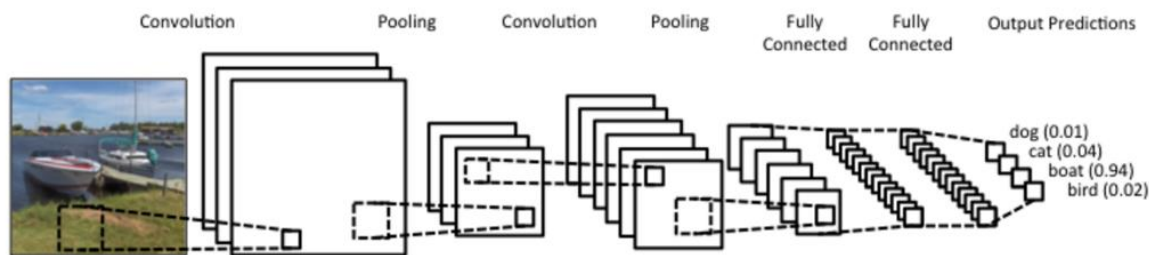
### 1.2.5 Конволюционни невронни мрежи

Конволюционните невронни мрежи (КНМ) са възникнали благодарение на учението на мозъчната зрителна кора. Те се използват основно за разпознаването на изображения и говор и за обработката на естествен език. В последните няколко години Конволюционните невронни мрежи успяват да се справят много добре благодарение на знанията за Изкуствените невронни мрежи, увеличението на изчислителната мощност и големите количества на данни за трениране.

Дейвид Х. Хубел и Торсен Виесел провеждат серия от екскременти върху котки през 1958 и 1959 г. Тези експерименти дават много важна информация за мозъчната зрителна кора. Вдъхновението за Конволюционните невронни мрежи идва именно от тези изследвания. В основата на Конволюционните невронни мрежи са Конволюционните слоеве и Обединяващите слоеве.

При Конволюционните слоеве невроните от първият слой не са свързани с всеки отделен пиксел от входното изображение, а само с пикселите от полето за което отговарят. Подобно нещо ще случва и в следващите слоеве, като всеки неврон от следващия слой бива свързан по-подобен начин само с областта за която отговаря. Тази архитектура позволява на мрежата да използва по-малко свойства и същевременно да извлича повече информация за тях.

Ролята на Обединяващите слоеве е да свиват входните данни с цел да намалят изчислителното натоварване, използването на памет и броя на параметри. Това свиване се случва като данните се разделят на отделни участъци и по някакъв начин се избира една стойност, с която да бъде заместен целият избраният участък. Един от най-използваните Обединяващи слоеве е Максимално обединяващият слой. Начинът, по който той свива входните данни е, като избира стойността, която да бъде заместител на всеки от отделните участъци да бъде най-голямата стойност от всеки от тези участъци.<sup>[7]</sup>



Фигура 1.6 Архитектура на проста Конволюционна невронна мрежа

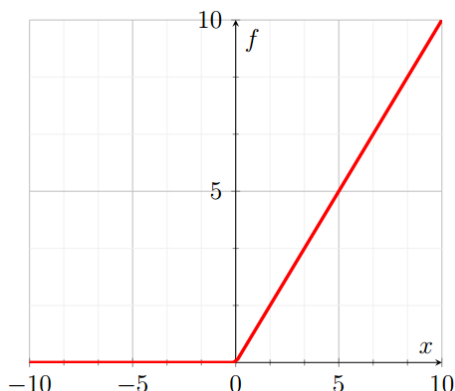
## 1.2.6 Функция на Загубата

Функцията на Загуба е функцията, която показва как се справя учещият се алгоритъм в сравнение с реалността. Смята се на база грешката, която се наблюдава при правене на предположения. Резултатът от Функцията на Загуба е средно аритметично на грешките за всички примери в набора от данни.<sup>[10]</sup>

## 1.2.7 Функция на Активиране

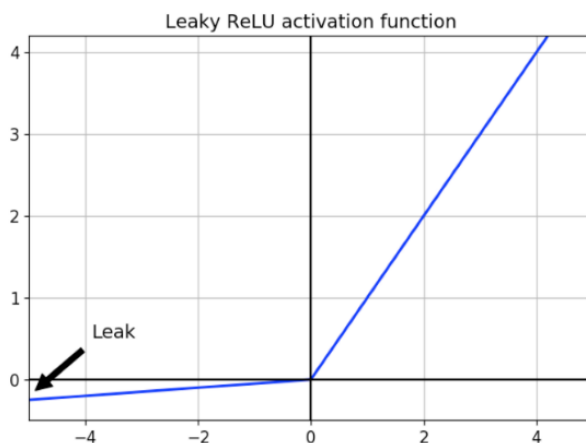
Функцията на Активиране се използва за пресмятане на стойността, която ще се предаде от един неврон към друг в Невронната мрежа. Входа и изхода на функцията са скаларни величини.<sup>[10]</sup> Някои от Функциите на Активиране са:

- Ректифицирана линейна активация (ReLU)



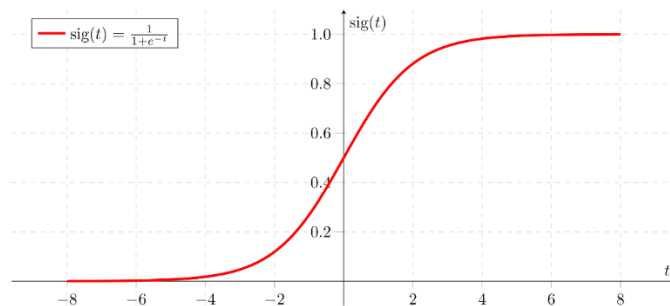
Фигура 1.7 Графика на функцията на Ректифицираната линейна активация

- Пробита Ректифицирана линейна активация (Leaky ReLU)



Фигура 1.8 Графика на функцията на Пробитата Ректифицирана линейна активация

- Сигмоидна активация (Sigmoid) - Използва се за намаляване на екстремни или извънредни стойности.



Фигура 1.9 Графика на функцията на Сигмоидната активация

## 1.2.8 Оптимизатор

Проблемът, който разрешава Машинното самообучение може да се разглежда като оптимизационен. Целта на Оптимизатора е да намери най-малката стойност на Функцията на Загуба.<sup>[10]</sup> Някои Оптимизатори са:

- SGD
- RMSprop
- Adagrad
- Adadelata
- Adam
- Adamax
- Nadam

## 1.2.9 Хипер параметри (Hyperparameters)

Хипер параметрите (Hyperparameters) се използват за контролиране на оптимизационния алгоритъм. Чрез тяхната промяна се контролира обучението на алгоритъмът за Машинно самообучение.<sup>[10]</sup> Част от тези параметри са:

- Скорост на обучение – Скоростта на обучение определя, колко бързо Невронната мрежа актуализира своите параметри. С ниска Скорост на обучение Мрежата успява да се научи по-добре, но е по-бавно. С висока скорост на обучение процесът на трениране е по-бърз, но резултатите не са толкова задоволителни. Методът, който се предпочита за използване е с Намаляваща (Decay) Скорост на обучение.
- Брой на епохи – Това е броят на пъти през които Невронната Мрежа се тренира върху целият Набор от данни за трениране.

- Брой на елементите в групата – Това е броят на елементите в група, която се дава на Невронната Мрежа и след която следва актуализация на параметрите.

### 1.2.10 Нормализиране на групата (Batch Normalization)

Обучението на Дълбоките невронни мрежи се усложнява от факта, че разпределянето на входа на всеки слой се променя по време на трениране, тъй като параметрите на предните слоеве се променя. По този начин се забавя процесът на трениране. Нормализирането на група се добавя в модела на архитектурата на Невронната мрежа. Това нормализиране ни позволява да използваме по-големи Скорости на обучение и да бъдем по-малко внимателни при инициализацията.<sup>[17]</sup>

### 1.2.11 Начини за оценяване на алгоритъма

#### 1.2.11.1 Използвани параметри

За оценяване на алгоритъма съм използвам следните параметри:

- Истински Позитивни (True Positive – TP) - Обект присъства и на снимката и в предположението
- Фалшиви Позитивни (False Positive – FP) - Обект присъства в предположението, но не и на снимката
- Истински Отрицателни (True Negative – TN) - Обекта не присъства нито на снимката, нито в предположението
- Фалшиви Отрицателни (False Negative – FN) - Обекта присъства на снимката, но не и в предположението
- Истински обекти (ИО) - Брой на истинските обекти на снимка.

#### 1.2.11.2 Използвани метрики

Мерките, които съм пресмятал от тези параметри са:

- Прецизност (Precision) – Прецизността показва, какво е отношението на правилно направените предположения към всички обекти, за които са направени предположения.

$$precision = \frac{TP}{TP + FP}$$




- Отзоваване (Recall) – Отзоваването показва, какво е отношението на правилно направените предположения към всички обекти, за които присъстват на снимката.

$$recall = \frac{TP}{TP + FN}$$

- F1 score – F1 score е метриката, която комбинира Прецизността и Отзоваването. Това е метриката, чрез която може да се следи точността на алгоритъма.<sup>[21]</sup>

$$f1 = \frac{precision * recall}{precision + recall}$$

- Пресичане върху Обединение (ПвО) (Intersection over Union (IoU)) - ПвО е мярката, която измерва припокриването между предположената кутийка и истинската кутийка.

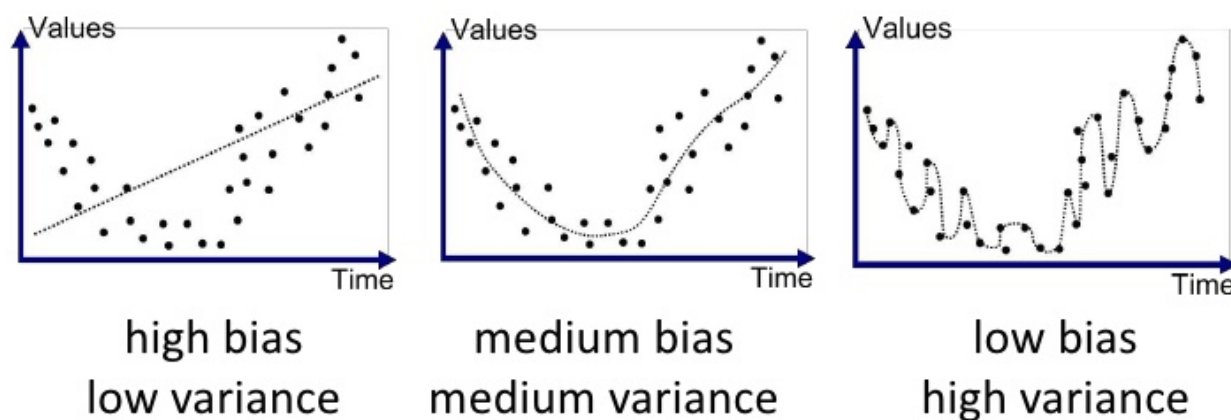


$$IoU = \frac{\text{Area of Overlap}}{\text{Area of Union}}$$

Фигура 1.10 Формула за пресмятане на ПвО

### 1.2.11.3 Компромисът Отклонение-Вариране (Bias-Variance Tradeoff)

Грешката на предположенията може да се раздели на две части. Едната част се дължи на Отклонението (Bias), а другата на Варирането (Variance). Чрез познаването на тези грешки може по-лесно да определяме проблеми свързани с Прекомерното нагаждане (Overfitting) и с Незадоволителното приспособяване (Underfitting). При грешката свързана с Отклонението имаме проблем с Прекомерното нагаждане, а при грешката свързана с Варирането имаме проблем с Незадоволителното приспособяване. Когато някоя от грешките не присъства в нашият алгоритъм и имаме средна грешка на Отклонение и Вариране имаме Точно приспособяване.<sup>[22]</sup>



Фигура 1.11 Компромисът Отклонение-Вариране

### 1.2.12 Без-Максимално Потискане (Non-Max Suppression)

Функцията на Без-Максималното Потискане (БМП) се използва за изваждането на истински предположения от всички предложения. Тази функция приема всички предложени кутии от изхода на мрежата и тяхната увереност. БМП избира кутиятката с най-голяма увереност и след това за всяка от кутийка изчислява ПвО със всяка от вече избраните кутийки и ако е по-малко от някакъв праг я избира. Изхода на тази функция са истинските предположения направени от мрежата.

### 1.2.13 Набор от данни

Едно от най-важните неща от процеса на Машинното самообучение е набора от данни. Много е важно да познаваме данните с които работим.<sup>[8]</sup>

## Глава Втора

## 2 Изисквания и използвани средства

### 2.1 Функционални изисквания на проекта

Проектът трябва да представлява система, която изработва снимка без хора от статичен видеоклип, като изготвената снимка трябва да бъде само на фона. Тази система ще представлява интернет приложение, което ще представлява интерфейс между потребителя и логиката на приложението. Потребителя ще изпраща статичен видеоклип, който ще бъде обработван от логиката на приложението и ще му бъде връщана готова снимка без хора. Логиката на приложението трябва да бъде съставена от Невронна мрежа, която открива хора на снимка и от част, която съединява много кадри от видеоклипа в една обща снимка на фона без хора.

### 2.2 Развойна среда, език и библиотеки

Езикът избран за реализация на проекта е Python. Той се е превърнал в универсален език за много приложения в сферата на анализ и работа с данни. Python има библиотеки за машинно самообучение, дълбоко самообучение, зареждане на данни, визуализация, обработка на изображения и видеоклипове като TensorFlow, Keras, Numpy SciPy, scikit-image, scikit-video, PIL, Matplotlib, Pickle и т.н. Освен библиотеки свързани с анализа и работата с данни Python разполага и с много библиотеки с общо приложение като интернет рамката Django.<sup>[8]</sup>

Keras е приложно програмен интерфейс от високо ниво за Невронни мрежи, написан на Python. Keras е способен да върви като интерфейс за TensorFlow, CNTK и Theano. Той е разработен с фокус върху бързото експериментиране. Keras поддържа Конволюционни невронни мрежи и върви, както върху видео карта, така и върху процесор.<sup>[11]</sup> Използвал съм Keras за проектиране и обучението на модела на Конволюционната невронна мрежа, която се грижи за откриването на хората на снимка.

TensorFlow е библиотека с отворен код за числени изчисления. Разработен е за Машинно самообучение и Дълбоки невронни мрежи.<sup>[12]</sup> Използвам го като основа за Keras, за пресмятане на Функцията на Загуба и за извличането на предложенията от Конволюционната невронна мрежа.

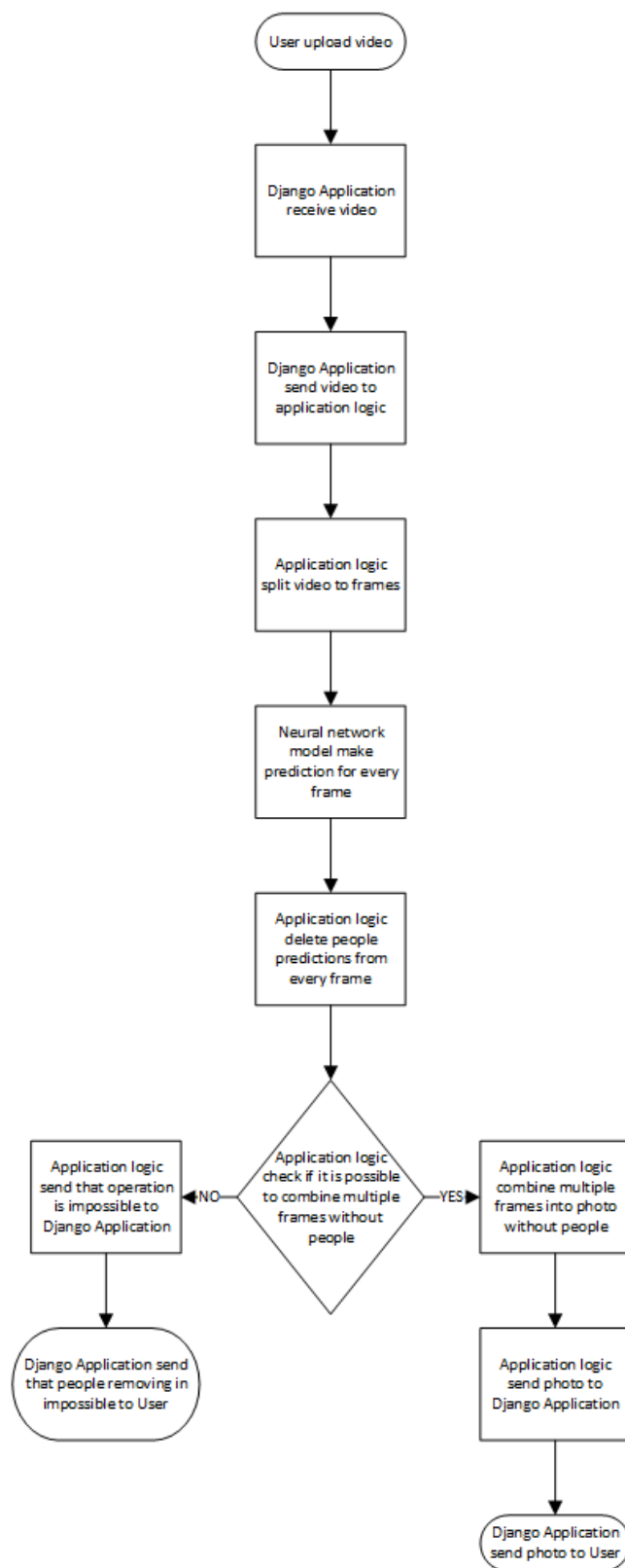
Numpy е един от основните пакети за научни пресмятания в Python. Той съдържа функционалности за многомерни масиви и математически функции от високо ниво. Освен приложенията си в научната сфера, може да се използва като контейнер за данни.<sup>[13]</sup> Numpy е използван за обработка и съхранение на Наборът от Данни, за пресмятането на Функцията на Загуба, за извличането на предположенията от Конволюционната невронна мрежа и за пресмятане на метриците за оценяване на алгоритъма.

Django е интернет рамка за Python от високо ниво, която позволява бързо създаване на нов проект. С Django е много бързо и лесно да се мащабира проект.<sup>[14]</sup> Django се използва за създаване на интернет приложението.

Проектът е написан с помощта на Sublime Text. Той има красив интерфейс, поддържа оцветяване на Python синтаксиса и има много приставки за Python.<sup>[15]</sup>

## **2.3 Общо описание на алгоритъма**

Потребителя изпраща статичен видеоклип посредством Django приложение. Django приложението изпраща този видеоклип към основната логика на приложението. Там видеоклипа бива разделен на кадри. За всеки от кадрите се правят предположения от Конволюционната невронна мрежа за местоположенията на хората на снимките. След това се премахват намерените хора на всеки от кадрите. Последващата стъпка е да се провери дали е възможно да се направи снимка без хора, ако е възможно то се комбинират многото снимки с премахнати хора до една обща снимка без хора. Ако е възможно да се направи снимка на фона последната стъпка е да се изпрати изготвената снимка посредством Django приложението на потребителя, а ако не е възможна последната стъпка е да се изпрати на потребителя, че е невъзможно да се направи снимка на фона посредством Django приложението.



Фигура 2.1 Блок схема на принципа на работа

## 2.4 База Данни

Използвам база данни за съхранение на видеоклипа и изготвената снимка.



Фигура 2.2 Диаграма на Базата Данни

## 2.5 Използван набор от данни

За откриването на хората е използван наборът от данни TUD Multiview Pedestrians събран от Мича Андрилука, Стефан Ротх и Бернт Шиеле през 2010 г. Структурата на набора от данни включва по една директория за данните за трениране, валидиране и тестване, осем файла с анотации за данните за трениране и по един файл с анотациите за данните за валидиране и тестване. Данните, които включва в себе си са 4732 примера за обучение, 248 примера за валидиране и 248 примера за тестване. Анотациите, които включва в себе си са XML файлове, които съдържат информацията за всяка снимка.<sup>[16]</sup> Анотациите, които гледам за всяка снимка от набора от данни са:

- Името на снимката
- Координатите на най-малката точка на ограничаващата кутийка за всеки човек, който присъства на снимката
- Координатите на най-голямата точка на ограничаващата кутийка за всеки човек, който присъства на снимката
- Координатите на центъра на ограничаващата кутийка за всеки човек, който присъства на снимката

## Глава Трета

### 3 Имплементация

Имплементацията на системата за изготвяне на снимка без хора от статичен видеоклип се разделя на три основни части. Тези части са комуникацията с потребителя, откриването на хора на снимка и изготвянето на снимка без хора.

#### 3.1 Комуникация с потребителите

Комуникацията с потребителите съм избрал да се осъществява чрез Django приложение.

##### 3.1.1 Приемане на видеоклип

Django приложението приема статичен видеоклип през форма в интернет сайт. За формата съм използвал Django форми, които могат да се напасват по модела на базата данни, а за дизайна и съм използвал Bootstrap.

##### 3.1.2 Изпращане на видеоклипа за обработване

След приемането на видеоклипа приложението го запазва в база данни и го изпраща към частта от системата, която се грижи за обработката на видеоклипа до снимка без хора. Това изпращане се осъществява чрез внесеният във файлът с контролери обект **video\_processing**. Инициализацията на този обект се получава във файлът **/web/Aovek/video\_processing.py**.

##### 3.1.3 Получаване на готовата снимка

След обработването на видеоклипа Django приложението приема изготвената снимка, която запазва в Базата Данни и след това показва на потребителя, като му предоставя възможността да си я изтегли.

#### 3.2 Откриване на хора на снимка

За откриването на хора на снимка съм използвал Конволюционна невронна мрежа. То се състои от няколко части. Тези части са обработката на Набора от Данни, моделиране на Конволюционна невронна мрежа, която се грижи за откриването на хора на снимка,

тренирането на Конволюционна невронна мрежа, оценяване на Конволюционна невронна мрежа, правенето на предположения от Конволюционна невронна мрежа и файл чрез който се управлява целият този процес.

### 3.2.1 Файл с конфигурациите за откриването на хора

Във файла `/config.json` се съдържа информация за конфигурациите за Набора от Данни и за Конволюционна невронна мрежа за откриване на хора на снимка. Ключовете, които съдържа са:

- Набора от данни
- Информацията за снимките
- Информацията за етикетите
- Информацията за видеоклипа
- Конволюционна невронна мрежа

### 3.2.2 Обработка на Набора от Данни

#### 3.2.2.1 Скрипт за изтегляне на Набора от Данни

Файлът `/aovek/preprocess/download_dataset.py` съдържа функция за изтеглянето на Набора от Данни. Тази функция приема като параметър файла с конфигурации от който взема интернет адреса на Набора от Данни и пътят в който да бъде записан.

#### 3.2.2.2 Обработка на Набора от Данни

Обработката на Набора от Данни се съдържа в класа **DataProcessing** във файла `/aovek/preprocess/data_preprocessing.py`.

Обработката на Набора от Данни представлява направата на три pickle файла за данните за трениране, валидиране и тестване, който съдържат в себе си речник с два артикула. Единият артикул съдържа в себе си данните, а другият артикул съдържа в себе си етикетите.

Процесът на създаване на pickle файловете е:

Извличане на информацията за снимките и етикетите от файловете съдържащи анотациите. За всяка от снимките вземам името на снимката, двата ъгъла на ограничителната кутия и центърът ѝ (където липсва центърът го пресмятам). След извличането на информацията тя представлява речник съдържащ артикули с ключ – името на снимката и стойност – информацията за всички ограничаващи кутии на присъстващите хора на снимката.



Последващата стъпка е да се вземат пътищата на всички снимки, за които е извлечена информация.

Изготвянето на pickle файл от пътищата на снимките и информацията за етикетите представлява записване на информацията, която е получена от обработката на снимката и обработката на етикетите. Обработката на снимките се съдържа в метода **process\_image(self, image\_file)** във файла **/aovek/utils/image\_processing.py**, а обработка на етикетите става в метода **process\_image\_labels(self, annotations, original\_size)** и **process\_label\_annotation(self, annotation, original\_size)** във файла **/aovek/preprocess/data\_preprocessing.py**.

```
def process_image(self, image_file):
    image_data = \
        ndimage.imread(image_file, mode=self.color_mode).astype(float)

    if self.color_channels == 1:
        image_data = np.expand_dims(image_data, axis=2)

    original_size = np.squeeze(image_data).shape[:-1]

    if image_data.shape != (self.image_size, self.image_size,
                           self.color_channels):
        image = resize(image_data,
                       output_shape=(self.image_size, self.image_size),
                       mode='constant')
    else:
        image = image_data

    image = np.expand_dims(image, axis=0)

    if self.normalizer == '[0, 255]':
        image = self.normalize_image_without_normalization(image)
    elif self.normalizer == '[0, 1]':
        image = self.normalize_image_from_0_to_1(image)
    elif self.normalizer == '[-1, 1]':
        image = self.normalize_image_from_minus1_to_1(image)

    return (image, original_size)
```

Методът **process\_image(self, image\_file)** приема път на снимка и връща обработената снимка и оригиналният размер на снимката. Обработката представлява обработка на цветовете канали, резолюцията и нормализация на снимката.

```
def process_label_annotation(self, annotation, original_size):
    center_x = annotation[0][0] / original_size[0]
    center_y = annotation[0][1] / original_size[1]

    center_w = \
        (annotation[1][1][0] - annotation[1][0][0]) / original_size[0]
    center_h = \
        (annotation[1][1][1] - annotation[1][0][1]) / original_size[1]

    box = np.array([center_x, center_y, center_w, center_h], ndmin=2)

    grid_x = int(center_x * self.grid_size)
    grid_y = int(center_y * self.grid_size)

    return box, grid_x, grid_y
```

Методът **process\_image\_labels(self, annotations, original\_size)** приема извлечените анотации за обработваната снимка от файловете с анотации и оригиналният размер на снимката и връща матрица представляваща репрезентацията на снимката като решетка. Всяка клетка на решетката в която се съдържа център на Ограничителната кутия на човек представлява вектор, който съдържа пет стойности. Тези стойности са две стойности за координатите на Ограничителната кутия около човека, две стойности за височината и широчината на Ограничителната кутия и една стойност за увереността на Ограничителната кутия. Стойностите за Ограничителната кутия и местоположението ѝ се изчисляват в метода **process\_label\_annotation(self, annotation, original\_size)**. Стойности на координатите на центъра, височината и ширината представляват процентно съотношение спрямо оригиналният размер на снимката.

### **3.2.3 Моделиране на Конволюционната невронна мрежа за откриване на хора**

Моделирането на Конволюционната невронна мрежа за откриване на хора на снимка се случва в класа **YOLO** във файлът **/aovek/network/network.py**.

#### **3.2.3.1 Моделиране на архитектурата на Конволюционната невронна мрежа за откриване на хора**

Моделът на Невронната мрежа за откриване на хора на снимка представлява Конволюционна невронна мрежа. Входа на Конволюционната невронна мрежа е с дименсии  $288 \times 288 \times 1$ . Тези размери представляват резолюцията на снимката. Изхода на Невронната мрежа представлява матрица, която е решетка, изобразяваща снимката.

Всяка от клетките, които се намират в решетката на изхода съдържа информация за Ограничителната кутия на човека, чийто център се намира в нея и увереността на предположението. Информацията за Ограничителната кутия представляват четири стойности - две стойности за координатите на центъра на Ограничителната кутия и две стойности за височината и ширината на Ограничителната кутия.

Конволюционните слоеве използват произволно инициализиране на тежестите, което е с нормално разпределение, еднакво уплътнение (same padding), отклонение (bias), размер на ядрото (kernel size) с размер  $3 \times 3$ , големина на стъпката за преместване на ядрото (stride) със стъпки 1 за хоризонталата и 1 за вертикалата, и филтри вариращи между  $2^5$  и  $2^{10}$ . Единственият Конволюционен слой, който е по-различен е последният, чийто филтри са с размер равен на броя на стойностите в Ограничителната кутия и размерът на ядрото е  $1 \times 1$ .

Максималният обединяващ слой използва област на обединението (pool size) с размер  $2 \times 2$ .

Архитектурата е разделена на групи съдържащи няколко Конволюционни слоя, всеки последван от Пробита Ректифицирана линейна активация и Нормализиране на групата последвани от един Максимално обединяващ слой. Тези групи са заменени в края на Невронната мрежа с 2 подобни групи без Максимално обединяващ слой, последвани от единичен Конволюционен слой използващ Сигмоидна активация.

Структурата на Конволюционната невронна мрежа съдържа 5 групи от Конволюционни и Максимално Обединяващи слоеве последвани от края на Невронната мрежа.

Архитектурата на Конволюционната невронна мрежа за откриване на хора представлява:

Layer (type)	Output Shape	Param #
=====		
input_1 (InputLayer)	(None, 288, 288, 1)	0
conv_1 (Conv2D)	(None, 288, 288, 32)	320
norm_1 (BatchNormalization)	(None, 288, 288, 32)	128
relu_1 (LeakyReLU)	(None, 288, 288, 32)	0
pool_1 (MaxPooling2D)	(None, 144, 144, 32)	0
conv_2 (Conv2D)	(None, 144, 144, 64)	18496
norm_2 (BatchNormalization)	(None, 144, 144, 64)	256
relu_2 (LeakyReLU)	(None, 144, 144, 64)	0
pool_2 (MaxPooling2D)	(None, 72, 72, 64)	0
conv_3 (Conv2D)	(None, 72, 72, 128)	73856
norm_3 (BatchNormalization)	(None, 72, 72, 128)	512
relu_3 (LeakyReLU)	(None, 72, 72, 128)	0
conv_4 (Conv2D)	(None, 72, 72, 64)	8256
norm_4 (BatchNormalization)	(None, 72, 72, 64)	256
relu_4 (LeakyReLU)	(None, 72, 72, 64)	0
conv_5 (Conv2D)	(None, 72, 72, 128)	73856
norm_5 (BatchNormalization)	(None, 72, 72, 128)	512
relu_5 (LeakyReLU)	(None, 72, 72, 128)	0
pool_3 (MaxPooling2D)	(None, 36, 36, 128)	0
conv_6 (Conv2D)	(None, 36, 36, 256)	295168
norm_6 (BatchNormalization)	(None, 36, 36, 256)	1024
relu_6 (LeakyReLU)	(None, 36, 36, 256)	0
conv_7 (Conv2D)	(None, 36, 36, 128)	32896
norm_7 (BatchNormalization)	(None, 36, 36, 128)	512
relu_7 (LeakyReLU)	(None, 36, 36, 128)	0
conv_8 (Conv2D)	(None, 36, 36, 256)	295168
norm_8 (BatchNormalization)	(None, 36, 36, 256)	1024
relu_8 (LeakyReLU)	(None, 36, 36, 256)	0

pool_4 (MaxPooling2D)	(None, 18, 18, 256)	0
conv_9 (Conv2D)	(None, 18, 18, 512)	1180160
norm_9 (BatchNormalization)	(None, 18, 18, 512)	2048
relu_9 (LeakyReLU)	(None, 18, 18, 512)	0
conv_10 (Conv2D)	(None, 18, 18, 256)	131328
norm_10 (BatchNormalization)	(None, 18, 18, 256)	1024
relu_10 (LeakyReLU)	(None, 18, 18, 256)	0
conv_11 (Conv2D)	(None, 18, 18, 512)	1180160
norm_11 (BatchNormalization)	(None, 18, 18, 512)	2048
relu_11 (LeakyReLU)	(None, 18, 18, 512)	0
conv_12 (Conv2D)	(None, 18, 18, 256)	131328
norm_12 (BatchNormalization)	(None, 18, 18, 256)	1024
relu_12 (LeakyReLU)	(None, 18, 18, 256)	0
conv_13 (Conv2D)	(None, 18, 18, 512)	1180160
norm_13 (BatchNormalization)	(None, 18, 18, 512)	2048
relu_13 (LeakyReLU)	(None, 18, 18, 512)	0
pool_5 (MaxPooling2D)	(None, 9, 9, 512)	0
conv_14 (Conv2D)	(None, 9, 9, 1024)	4719616
norm_14 (BatchNormalization)	(None, 9, 9, 1024)	4096
relu_14 (LeakyReLU)	(None, 9, 9, 1024)	0
conv_15 (Conv2D)	(None, 9, 9, 1024)	9438208
norm_15 (BatchNormalization)	(None, 9, 9, 1024)	4096
relu_15 (LeakyReLU)	(None, 9, 9, 1024)	0
conv_16 (Conv2D)	(None, 9, 9, 5)	5125
reshape_1 (Reshape)	(None, 9, 9, 5)	0

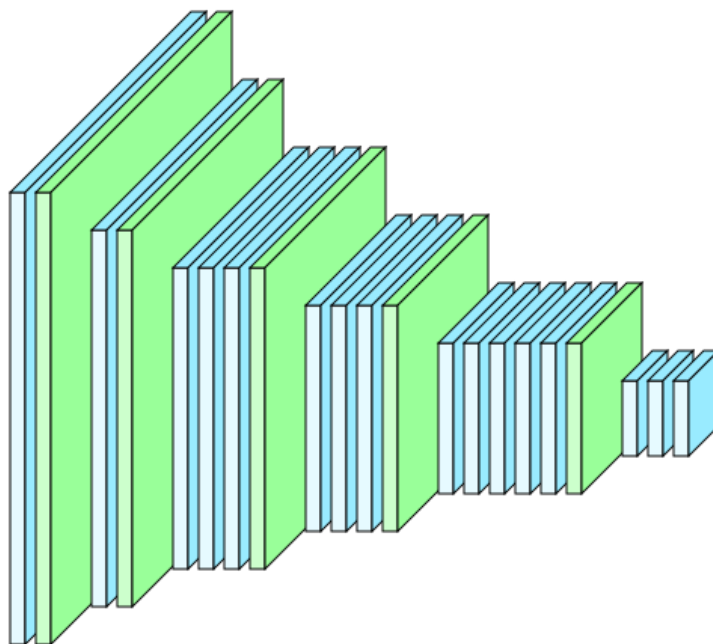
=====

Total params: 18,784,709

Trainable params: 18,774,405

Non-trainable params: 10,304

Фигура 3.1 Архитектура на Конволюционната невронна мрежа за откриване на хора



Фигура 3.2 Структура на Конволюционната невронна мрежа за откриване на хора. Синьо – Конволюционен слой, Зелено – Максимално обединяващ слой

### 3.2.3.2 Създаване на модела на Конволюционната невронна мрежа за откриване на хора

Функцията `create_model(self)` в класа **YOLO** се грижи за създаването на модела на Конволюционната невронна мрежа за откриване на хора. В тази функция се съдържа направата на първия слой, а добавянето на другите слоеве се случва във функцията `create_network(self, input)`. Другото, което се съдържа в тази функция е добавянето на оптимизатора и на функцията на загуба.

### 3.2.4 Трениране на Конволюционната невронна мрежа за откриване на хора

За трениране на Конволюционната невронна мрежа за откриване на хора се грижи класът **Train** във файлът `/aovek/training/train.py`.

Първата стъпка при създаване на обект за трениране е да се зареди Наборът от Данни. Това става чрез метода `load_dataset(self)` в класа **Train**, който извиква метода `load_data(self)` от

класа, който наследява и се грижи за зареждането на набора от данни **DataLoading** във файла **/aovek/utills/data\_loading.py**.

Следващата стъпка е да се извика метода **train(self, config)**, който се грижи за тренирането на Конволюционната невронна мрежа за откриване на хора. Освен за тренирането на Невронната мрежа този метод се грижи за запазване на модела, за оценяване на тренираният модел на Невронната мрежа и за създаването на текст съдържащ информация за процесът на трениране на Мрежата, който съдържа архитектурата на Мрежата, информация за различните епохи през процеса на трениране на Мрежата, информация за Оптимизатора, броят на елементите в групата, резултата от оценяването на Мрежата и времето за което се е тренирала Мрежата.

Към тренирането на Конволюционната невронна мрежа за откриване на хора на снимка спадат и:

### 3.2.4.1 Функция на Загуба

```
def custom_loss(self, true, pred):
    loss = 0

    true = \
        tf.reshape(true, shape=(-1, self.grid_size ** 2,
                                (self.number_of_annotations + 1)))
    pred = \
        tf.reshape(pred, shape=(-1, self.grid_size ** 2,
                                (self.number_of_annotations + 1)))

    x_true = true[:, :, 0]
    x_pred = pred[:, :, 0]

    y_true = true[:, :, 1]
    y_pred = pred[:, :, 1]

    w_true = true[:, :, 2]
    w_pred = pred[:, :, 2]

    h_true = true[:, :, 3]
    h_pred = pred[:, :, 3]

    c_true = true[:, :, 4]
    c_pred = pred[:, :, 4]

    loss += \
        np.sum(
            tf.scalar_mul(
                self.alpha_coord,
                tf.multiply(
```

```

        c_true,
        tf.add(tf.squared_difference(x_true, x_pred),
                tf.squared_difference(y_true, y_pred))))))

    loss += \
        np.sum(
            tf.scalar_mul(
                self.alpha_coord,
                tf.multiply(
                    c_true,
                    tf.add(tf.squared_difference(tf.sqrt(w_true),
                                                  tf.sqrt(w_pred)),
                          tf.squared_difference(tf.sqrt(h_true),
                                                  tf.sqrt(h_pred))))))

    loss += np.sum(tf.multiply(c_true,
                               tf.squared_difference(c_true, c_pred)))

    loss += \
        np.sum(
            tf.scalar_mul(
                self.alpha_noobj,
                tf.multiply((1 - c_true),
                            tf.squared_difference(c_true, c_pred))))

    return loss

```

Функцията на Загуба използва два хипер параметъра - `alpha_coord` и `alpha_noobj`. Параметърът `alpha_coord` се използва за избиране, на каква част от разликата на истинските и на предположените координатите на Ограничителната кутия ще влиза във Функцията на Загуба, а параметърът `alpha_noobj` се използва за избиране на каква част от грешно предположената сигурност се съдържа във функцията.

Функцията на загуба съдържа няколко стъпки:

1. Променя формата на решетката съдържаща предположенията за снимка до едномерен масив съдържащ клетките с Ограничителните кутии.
2. Извличане на следните свойства от предположението - стойностите за Ограничителните кутия и сигурността на предположенията.
3. Пресмятане на функцията на загуба. Елементите, които включва са:
  - Квадратната разлика на координатите на центъра на Ограничителната кутия, за всяка от клетките в решетката на предположението и свойствата на Истинската кутия умножено по коефициентът `alpha_coord`.



- Квадратната разлика на корен квадратен от височината и широчината на Ограничителната кутия, за всяка от клетките в решетката на предположението и свойствата на Истинската кутия умножено по коефициентът `alpha_coord`.
- Квадратната разлика на сигурността на предположението с тази на сигурността на истинските, за Истинските кутии които съдържат хора в тях.
- Квадратната разлика на сигурността на предположението с тази на истинската, за Истинските кутии, които не съдържат хора в тях умножена по коефициентът `alpha_noobj`.

### 3.2.4.2 Оптимизатор

Използваният от мен Оптимизатор е Adam, защото при експериментите показва най-добър резултат. Параметрите на обучение, които съм използвал след много експерименти са:

- Скорост на обучение (Learning rate) = 0.001
- Разпад (Decay) = 0.0005

### 3.2.4.3 Тренировъчни експерименти

Първата серия експерименти са направени с пободна архитектурата на Конволюционната невронна мрежа за откриване на хора от Фигура 3.1. Разликата е, че не е използвано Нормализиране на групата и използваната Функция на Активиране е Ректифицирана линейна активация, а не е Пробита Ректифицирана линейна активация.

Данните от Набора от Данни са с цветови канали RGB и са нормализирани от 0 до 1.

Данни за оптимизатора:

Оптимизатор	Beta 1	Beta 2	Decay	Epsilon	Learning rate	Epoches number
Adam	0.9	0.999	0.0005	1e-08	0.00001	200

Измерените метрики за горният оптимизатор са:

Оптимизатор	Набор от данни	IoU	Precision	Recall	F1 Score
Adam	Train	0.945626	0.99225	0.898379	0.942984
Adam	Validation	0.649082	0.57931	0.289655	0.386207
Adam	Test	0.64199	0.588608	0.300971	0.398287

Следващите експерименти са направени с архитектурата на Конволюционната невронна мрежа за откриване на хора от Фигура 3.1.

Данни за оптимизатора:

Оптимизатор	Beta 1	Beta 2	Decay	Epsilon	Learning rate	Epoches number
Adam	0.9	0.999	0.0005	1e-08	0.00001	100

Измерените метрики за горният оптимизатор са:

Оптимизатор	Набор от данни	IoU	Precision	Recall	F1 Score
Adam	Train	0.936535	0.988987	0.923421	0.95508
Adam	Validation	0.671778	0.771429	0.27931	0.410127
Adam	Test	0.696989	0.682243	0.236246	0.350962

След добавянето на Нормализиране на групата и замяната на Ректифицирана линейна активация с Пробита Ректифицирана линейна активация забелязваме по-бързо трениране и по високи стойности на измерваните метрики.

Данни за оптимизатора:

Оптимизатор	Beta 1	Beta 2	Decay	Epsilon	Learning rate	Epoches number
Adam	0.9	0.999	0.0005	1e-08	0.00001	20

Измерените метрики за горният оптимизатор са:

Оптимизатор	Набор от данни	Loss	IoU	Precision	Recall	F1 Score
Adam	Train	0.000496	0.81242	0.966732	0.956206	0.96144
Adam	Validation	0.012837	0.667789	0.435294	0.127586	0.197333
Adam	Test	0.013732	0.657952	0.54717	0.187702	0.279518

В следващите експерименти Данните от Набора от Данни са чернобели с нормализация от 0 до 1.

Оптимизатор	Beta 1	Beta 2	Decay	Epsilon	Learning rate	Epoches number
Adam	0.9	0.999	0.0005	1e-08	0.00001	20

Измерените метрики за горният оптимизатор са:

Оптимизатор	Набор от данни	Loss	IoU	Precision	Recall	F1 Score
Adam	Train	0.000447	0.822859	0.939471	0.93322	0.936335
Adam	Validation	0.009902	0.643798	0.47343	0.337931	0.394366
Adam	Test	0.011348	0.650726	0.468599	0.313916	0.375969

В последващата серия от експерименти тренирането на Конволюционната невронна мрежа за откриване на хора на снимка е проведено за 10 епохи. Разликата на експериментите е оптимизатора:

Данни за оптимизаторите:

Оптимизатор	Beta 1	Beta 2	Decay	Epsilon	Learning rate	Momentum	Nestrov	rho	Schedule decay
SGD	-	-	0.0	-	0.01	0.0	False	-	-
RMSprop	-	-	0.0	1e-08	0.001	-	-	0.9	-
Adagrad	-	-	0.0	1e-08	0.01	-	-	-	-
Adam	0.9	0.999	0.0	1e-08	0.001	-	-	-	-
Adamax	0.9	0.999	0.0	1e-08	0.002	-	-	-	-
Nadam	0.9	0.999	-	1e-08	0.002	-	-	-	0.004
Adadelta	-	-	0.0	1e-08	1.0	-	-	0.95	-

Измерени метрики за горните оптимизатори:

Оптимизатор	Набор от данни	Loss	IoU	Precision	Recall	F1 Score
SGD	Train	0.024466	0.598528	0.163519	0.042947	0.068027
SGD	Validation	0.022902	0.699019	0.069767	0.010345	0.018018
SGD	Test	0.023842	0.562822	0.045455	0.006472	0.011331
RMSprop	Train	0.004473	0.783587	0.898651	0.798089	0.84538

RMSprop	Validation	0.0072	0.649677	0.655405	0.668966	0.662116
RMSprop	Test	0.007815	0.655257	0.626667	0.608414	0.617406
Adagrad	Train	0.002363	0.81438	0.944424	0.921002	0.932566
Adagrad	Validation	0.008998	0.649619	0.553785	0.47931	0.513863
Adagrad	Test	0.009099	0.662126	0.596838	0.488673	0.537367
Adam	Train	0.003263	0.793316	0.907716	0.849625	0.87771
Adam	Validation	0.007297	0.663885	0.698529	0.655172	0.676157
Adam	Test	0.007981	0.659545	0.686792	0.588997	0.634146
Adamax	Train	0.003267	0.778843	0.909389	0.838979	0.872766
Adamax	Validation	0.007821	0.652577	0.647887	0.475862	0.548708
Adamax	Test	0.008354	0.645215	0.700935	0.485437	0.573614
Nadam	Train	0.003686	0.783522	0.90989	0.85143	0.879695
Nadam	Validation	0.007446	0.684177	0.548387	0.644828	0.5927
Nadam	Test	0.00812	0.679968	0.560224	0.647249	0.600601
Adadelta	Train	0.003427	0.728824	0.813878	0.817324	0.815597
Adadelta	Validation	0.010758	0.618112	0.325503	0.334483	0.329932
Adadelta	Test	0.012358	0.615552	0.334375	0.346278	0.340223

От следните експерименти моделът с най-добри метрики на тестовите данни е този, който е с оптимизатор Adam. Заради това по следващите два експеримента са с този оптимизатор.

По-следващите експерименти показват:

Данни за оптимизаторите:

Оптимизатор	Beta 1	Beta 2	Decay	Epsilon	Learning rate	Epoches number
Adam	0.9	0.999	0.0	1e-08	0.0001	10
Adam	0.9	0.999	0.0005	1e-08	0.001	30

Измерени метрики за горните оптимизатори:

Оптимизатор	Набор от данни	Loss	IoU	Precision	Recall	F1 Score
Adam	Train	0.001547	0.772877	0.885908	0.898984	0.892398
Adam	Validation	0.009742	0.647855	0.529880	0.458621	0.491682
Adam	Test	0.010796	0.656667	0.498282	0.469256	0.483333
Adam	Train	0.000801	0.871971	0.979402	0.966368	0.972841
Adam	Validation	0.007801	0.692451	0.78341	0.586207	0.670611
Adam	Test	0.008452	0.687512	0.763393	0.553398	0.641651

### 3.2.5 Оценяване на Конволюционната невронна мрежа за откриване на хора

Основният файл, който се грижи за оценяването на Конволюционната невронна мрежа за откриване на хора на снимка е `/aovek/validate/metrics.py`. Класът **Metrics** в този файл се грижи за пресмятането на метриците. Методът, който осъществява пресмятането на метриците е `eval_metrics(self, images, labels)`. Този метод първо намира нужните параметри за изчисляването на метриците и после ги изчислява. Параметрите, който съм използвал за изчисляването на метриците са брой на Истински Позитивни, Фалшиви Позитивни, Фалшиви Отрицателни, Истински кутии и ПвО. Намиране на нужните параметри става, като се параметрите се намерят за всяка снимка от Набора от Данни една по една и след това се събират. Намирането на тези параметри за една снимка става по-следният начин:

1. Вземат се Истинските кутии от свойствата на снимките и се правят предположенията от модела на Конволюционната невронна мрежа, които са кутиите, които мрежата смята за реални предположения.
2. Намира се ПвО за всички кутии, които са истински и присъстват в свойствата на снимката.

```
def get_iou_for_image(self, gt, pred):
    iou_image = np.ndarray(shape=(0, pred.shape[0]),
                           dtype=np.float32)

    for box in gt:
        gt_box = np.full(pred.shape, box)

        iou_box = self.bboxes_iou(gt_box, pred)
```

```

iou_box = np.expand_dims(iou_box, axis=0)
iou_image = np.concatenate((iou_image, iou_box))

return iou_image

```

```

def boxes_iou(self, box1, box2):
    ymin_1 = np.minimum(box1[:, 0], box1[:, 2])
    xmin_1 = np.minimum(box1[:, 1], box1[:, 3])
    ymax_1 = np.maximum(box1[:, 0], box1[:, 2])
    xmax_1 = np.maximum(box1[:, 1], box1[:, 3])
    ymin_2 = np.minimum(box2[:, 0], box2[:, 2])
    xmin_2 = np.minimum(box2[:, 1], box2[:, 3])
    ymax_2 = np.maximum(box2[:, 0], box2[:, 2])
    xmax_2 = np.maximum(box2[:, 1], box2[:, 3])

    area_1 = (ymax_1 - ymin_1) * (xmax_1 - xmin_1)
    area_2 = (ymax_2 - ymin_2) * (xmax_2 - xmin_2)

    ymin_inter = np.maximum(ymin_1, ymin_2)
    xmin_inter = np.maximum(xmin_1, xmin_2)
    ymax_inter = np.minimum(ymax_1, ymax_2)
    xmax_inter = np.minimum(xmax_1, xmax_2)

    area_inter = (np.maximum(ymax_inter - ymin_inter, 0.0) *
                  np.maximum(xmax_inter - xmin_inter, 0.0))

    iou = area_inter / (area_1 + area_2 - area_inter)

    iou[np.where(area_1 < 0) or np.where(area_2 < 0)] = 0

    return iou

```

3. На база ПвО се изчисляват желаните параметри.

След като имаме нужните параметри изчисляването на метриците става по формулите от т. 1.2.9.

Файлът **/aovek/validate/model\_metrics.py** съдържа интерфейс на оценяването на Конволюционната невронна мрежа, който представлява отворъждане, което се подава на Невронната мрежа докато се тренира и се използва за изчисляването на метриците за

Наборът от Данни за валидиране по време на тренирането и за оценяване на моделът след неговото трениране върху трите части от наборът от данни.

Файлът `/aovek/validate/eval_metrics.py` служи за оценяване на трениран модел на Конволюционната невронна мрежа върху трите части от Наборът от Данни.

### 3.2.6 Правене на предположения от Конволюционната невронна мрежа за откриване на хора

Файлът `/aovek/visualization/predict.py` представлява интерфейс за визуализиране на предположенията направени от Конволюционната невронна мрежа за откриване на хора на снимка. Класът **Predict**, който се намира в този файл има възможност да визуализира снимка по снимка от трите части на Наборът от Данни с предположенията, които са направени от Невронната мрежа за нея.

Същинското правене на предположения на снимка се случва в класа **YOLO**, който се намира във файла `/aovek/network/network.py`. За правене на предположение на цялата решетка с предположения на снимката може да се използва функцията **predict(self, image)**. За правене на предположение само на кутиите, които Конволюционната невронна мрежа смята за правилни може да се използва метода **predict\_boxes(self, image)**. Тази функция ще вземе всички предположения с помощта на метода **predict(self, image)** и след това ще извлече само реалните предположения от решетката с предположения с помощта на метода **non\_max\_suppression(self, predict)**.

Правене на предположения за видео се случва, като се прави предположение за всеки кадър от това видео. Това нещо се случва във функцията **predict\_video(self, video)**. На изхода на тази функция получаваме масив с размери (броят кадри, максималният брой кутии за предположение на снимка, елементите на кутия). Втората дименсия е с максималният брой кутии, защото където са по-малко са запълнени с кутии на които всички стойности са 0.

Методът **non\_max\_suppression(self, predict)** се грижи за извличане на реалните предположения, които прави Конволюционната невронна мрежа за откриване на хора. Функцията на този метод е да редуцира предположенията по сигурност на предположението, като вземе тези предположения, които имат сигурност по-голяма от някакъв праг. След като имаме предположенията за които е сигурна невронната мрежа ги прекарваме през функцията

**non\_max\_suppression( boxes, scores, max\_output\_size, iou\_threshold=0.5, name=None )**, която идва от **tensorflow.image.non\_max\_suppression**. Резултатът на тази функция са индексите на кутиите подредени по сигурност, които имат по-малко ПвО с другите избрани Ограничителни кутии от някакъв праг.

```
def non_max_suppression(self, predict):
    predict = predict[predict[:, 4] > self.prob_threshold]

    probabilities = predict[:, 4]
    boxes = predict[:, :4]

    true_boxes_idx = \
        tf.image.non_max_suppression(boxes, probabilities,
                                     self.grid_size ** 2,
                                     iou_threshold=self.iou_threshold)
    true_boxes = tf.gather(boxes, true_boxes_idx)
    true_probabilities = tf.gather(probabilities, true_boxes_idx)

    true_boxes = tf.concat([true_boxes, true_probabilities[:, None]], axis=1)

    return true_boxes
```

### 3.2.7 Файл за управление на процеса за откриване на хора

Файлът **/aovek.py** служи за управление на процесите свързани с Конволюционната невронна мрежа за откриване на хора. При зададен конфигурационен файл чрез него може да се:

- Изтегля Набора от Данни.
- Обработка Набора от Данни.
- Тренира Конволюционната невронна мрежа за откриване на хора.
- Правят предположения върху Наборът от Данни от Конволюционната невронна мрежа за откриване на хора.
- Оценява трениран модел на Конволюционната невронна мрежа за откриване на хора.

## 3.3 Изготвяне на снимка без хора

Изготвянето на снимка без хора се осъществява в класа **VideoToImage** във файлът **/aovek/video/video\_to\_image.py**. Методът **process\_video\_file(self, video\_path)** в този клас се грижи за обработката на видеоклипа и за изготвянето на снимка.



### 3.3.1 Обработване на видеото

Класът **VideoProcessing** във файлът `/aovek/utils/video_processing.py` бива наследен от класа, който се грижи за изготвяне на снимка от статичен видеоклип **VideoToImage** и наследява класа **ImageProcessing**, който се грижи за обработката на снимки.

В класа **VideoToImage** се зарежда видеоклипа до numpy масив с помощта на метода `process_video(self, video_path)` от класа **VideoProcessing**. Следващата стъпка е да се направи версия на видеото с променени дименсии, за да може върху него Конволюционната невронна мрежа за откриване на хора да направи своите предположения. Тази обработка на видеоклипа се осъществява с помощта на метода `resize_video(self, video_path)`, съдържащ се в класа **VideoProcessing**.

### 3.3.2 Вземане на предположенията на Конволюционната невронна мрежа за откриване на хора за видеоклип

След като имам версията на видеоклипа с променени дименсии изпраща тази версия на тренираният модел на Конволюционната невронна мрежа за откриване на хора да направи своите предположения за хората на снимките. След получаването на тези предположения ги преобразувам до оригиналният размер на видеото. При това преобразуване умножавам предположенията с някакво отместване. Правя това, за да уголемя Ограничителните кутии, за да са по-точни.

### 3.3.3 Изготвяне на снимка от статичен видеоклип

Методът `make_image(self, video, predictions)` се грижи за създаването на снимка на фона от статичен видеоклип.

```
def make_image(self, video, predictions):
    image = np.zeros(video.shape[1:])

    up_border = 0
    down_border = image.shape[0]
    left_border = 0
    right_border = image.shape[1]

    for frame, prediction in zip(video, predictions):
        if np.sum(prediction[0:4]) == 0:
```

```

        continue
    for pred in prediction:
        if np.sum(pred[0:4]) == 0:
            continue

        image[up_border:int(pred[1]), left_border:right_border] =\
            frame[up_border:int(pred[1]), left_border:right_border]
        if int(pred[1]) > up_border:
            up_border = int(pred[1])

        image[int(pred[3]):down_border, left_border:right_border] =\
            frame[int(pred[3]):down_border, left_border:right_border]
        if int(pred[3]) < down_border:
            down_border = int(pred[3])

        image[up_border:down_border, left_border:int(pred[0])] =\
            frame[up_border:down_border, left_border:int(pred[0])]
        if int(pred[0]) > left_border:
            left_border = int(pred[0])

        image[up_border:down_border, int(pred[2]):right_border] =\
            frame[up_border:down_border, int(pred[2]):right_border]
        if int(pred[2]) < right_border:
            right_border = int(pred[2])

    return image.astype('uint8')

```

Изготвянето на снимка на фона се осъществява, като се обикаля видеоклипа кадър по кадър. За всеки от кадрите, за когото Конволюционната невронна мрежа е направила реално предположение се вземат пиксели, които не принадлежат на предположението и вече не са добавени в изготвяната снимка. За определяне, на кои пиксели не съм добавил в изготвената снимка съм използвал променливи за горният, долният, левият и десният край, които пазят в себе си границата на взетите пиксели за тези посоки.

## Глава Четвърта

### 4 Ръководство на потребителя

#### 4.1 Инсталация на пакети

Дипломната работа разчита на няколко външни пакета. За набавянето на тези пакети съм използвал Anaconda. Начинът по-който може да се инсталира на Linux<sup>[18]</sup>, macOS<sup>[19]</sup>, Windows<sup>[20]</sup>.

Стъпката след инсталацията на Anaconda е да се създаде виртуална среда чрез командата:

```
$ conda env create -f environment.yml --name aovek
```

където aovek е името на виртуалната среда, а environment.yml е файлът, който съдържа информация за средата.

Активирането на виртуалната среда става чрез командата:

```
$ source activate aovek
```

А деактивирането чрез командата:

```
$ source deactivate
```

Изтриването на тази виртуална среда става чрез командата:

```
$ conda env remove -name aovek
```

#### 4.2 Файл за контрол на процеса свързан с Конволюционната невронна мрежа за откриване на хора

Чрез файлът **/aovek.py** се осъществява контролът върху процесите свързани с Конволюционната невронна мрежа за откриване на хора. За изпълняване на някоя от функциите, които са част от този файл трябва да се подаде параметъра **-config\_file** със стойност пътя на конфигурационния файл. Пътят на този файл е **/config.json**. Преди да се тренира Конволюционната невронна мрежа за откриване на хора в този файл могат да се

нанесат корекции по начина на трениране на Невронната Мрежа или по обработката на Набора от Данни.

```
$ python3 aovek.py --help
usage: aovek.py [-h] -config_file CONFIG_FILE [-dataset download]
               [-processes_dataset] [-train] [-predict] [-evaluate]

4ovek: File for contolling people detection process

required arguments:
  -config_file CONFIG_FILE
                        Path to config file.

optional arguments:
  -dataset_download    Download dataset.
  -processes_dataset    Processes dataset.
  -train               Train convolutional neural network.
  -predict             Make predictions for entire dataset.
  -evaluate            Evaluate trained model.
```

Фигура 4.1 Помощното меню на файлът `/aovek.py`

## 4.3 Трениране на Конволюционната невронна мрежа за откриване на хора

За да се тренира Конволюционната невронна мрежа за откриване на хора трябва да се осъществят следните процеси:

- Изтегляне на Набора от Данни
- Обработване на Набора от Данни
- Трениране на Конволюционната невронна мрежа за откриване на хора на снимка

Тези стъпки могат да се извършат чрез следната команда:

```
$ python3 aovek.py -config_file ./config.json -dataset_download -processes_dataset -train
```

## 4.4 Стартиране на интернет приложението

Преди първото стартиране на интернет приложението трябва да се приложат миграциите върху базата данни.

```
$ python3 web/manage.py migrate
```

Стартирането на интернет приложението се осъществява чрез командата:

```
$ python3 web/manage.py runserver
```

## 4.5 Използване на Jupyter Notebook

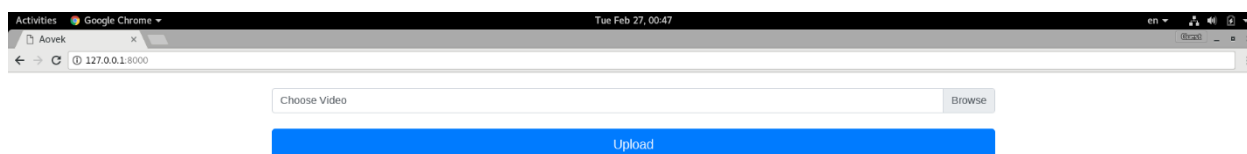
Проектът съдържа Jupyter Notebook, в който могат да се правят предположения от Конволюционната невронна мрежа за откриване на хора. Тези предположения могат да се правят както и върху снимка, така и върху видео кадър по кадър.

Този Jupyter Notebook се намира в `/notebooks/Predict.ipynb`. За използване на тази тетрадка трябва да се стартира Jupyter сървър, което става по следният начин:

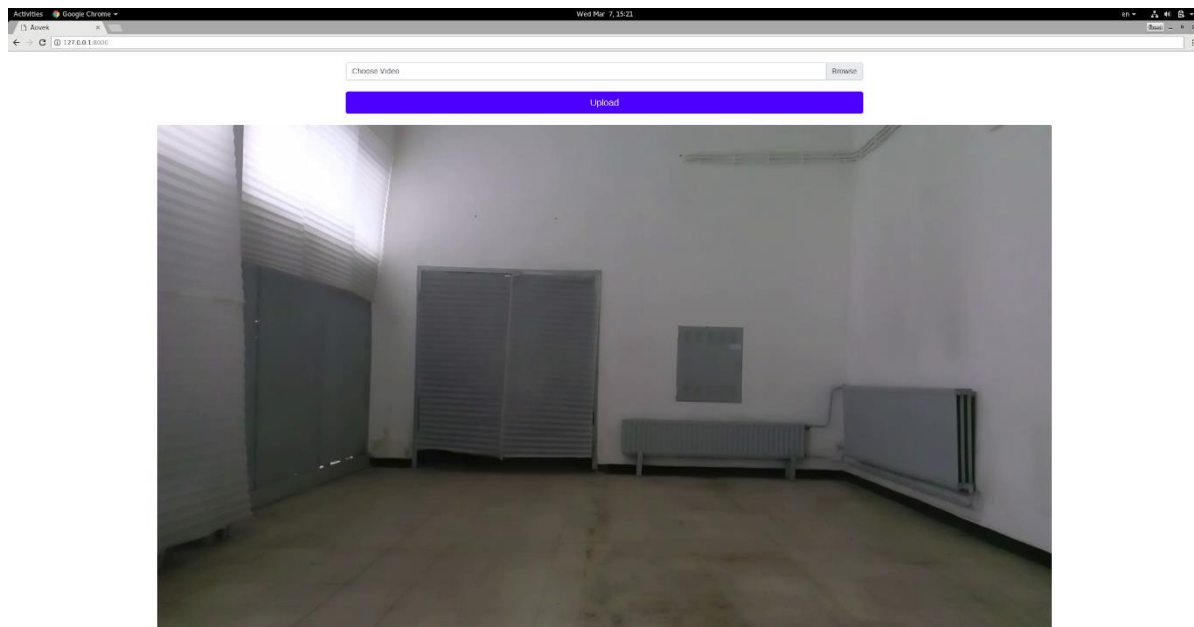
```
$ jupyter notebook
```

## 4.6 Използване на интернет приложението

Интернет приложението представлява една страница, която има форма, чрез която могат да се изпращат видеоклипове на същинската част от приложението. След като е изготвена снимката от видеото тя бива върната на интернет приложението, което я показва на потребителя и му дава възможността да си я изтегли.



Фигура 4.2 Интернет приложението в очакване на видеоклип от потребителя



Фигура 4.3 Интернет приложението е върнало готова снимка на потребителя

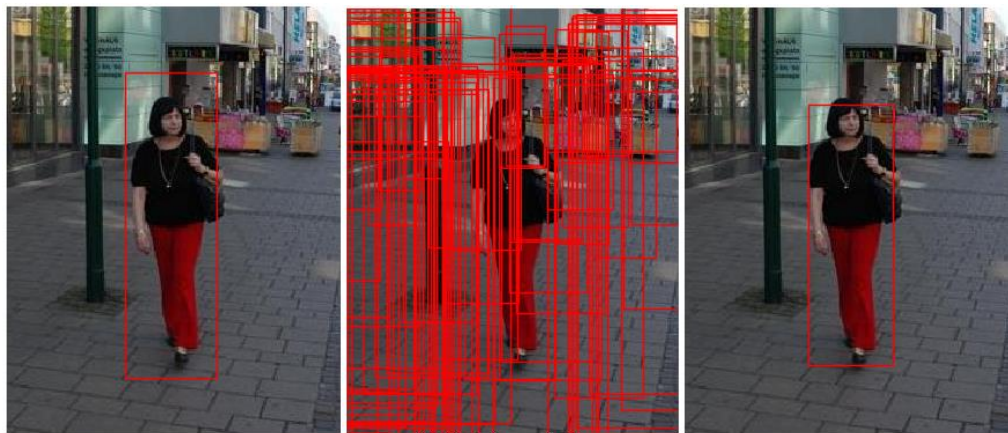
## 4.7 Правене на предположения от Конволюционната невронна мрежа за откриване на хора

### 4.7.1 Предположения за Наборът от Данни за трениране



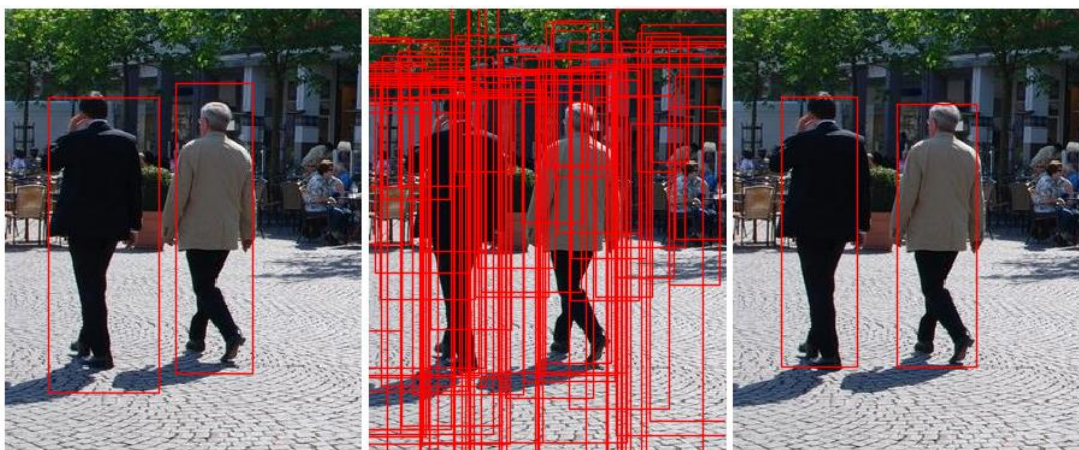
Фигура 4.4 Предположения от Конволюционната невронна мрежа за откриване на хора. Вляво – Предположения от Невронната мрежа, за които е сигурна (след Без-Максимално Потискане). В средата всички предположения направени от Невронната мрежа (преди Без-Максимално Потискане). Вдясно са свойствата от анотациите за снимката.

### 4.7.2 Предположения за Наборът от Данни за валидиране



Фигура 4.5 Предположения от Конволюционната невронна мрежа за откриване на хора. Вляво – Предположения от Невронната мрежа, за които е сигурна (след Без-Максимално Потискане). В средата всички предположения направени от Невронната мрежа (преди Без-Максимално Потискане). Вдясно са свойствата от анотациите за снимката.

### 4.7.3 Предположения за Наборът от Данни за тестване



Фигура 4.6 Предположения от Конволюционната невронна мрежа за откриване на хора. Вляво – Предположения от Невронната мрежа, за които е сигурна (след Без-Максимално Потискане). В средата всички предположения направени от Невронната мрежа (преди Без-Максимално Потискане). Вдясно са свойствата от анотациите за снимката.



## Заклучение

Разработената дипломна работа отговаря на всички изисквания.

Дипломна работа разглежда различни начини за откриване и премахване на хора от снимка. Разгледани са различни техники и проекти свързани с тези области. Частта за откриване на хора може да открива Ограничителни кутии на хора на снимка с Пресичане върху Обединение 68.75 %, Прецизност 76.34 %, Отзоваване 55.34 и F1 Score 64.17 %.

Производителността на Конволюционната Невронна Мрежа за откриване на хора може да се подобри, чрез нагласяне на хипер параметрите на оптимизатора.



## Използване литература

- [1] Chris McCormick, HOG Person Detector Tutorial, <http://mccormickml.com/2013/05/09/hog-person-detector-tutorial/>, May 2013
- [2] Joyce Xu, Deep Learning for Object Detection: A Comprehensive Review, <https://towardsdatascience.com/deep-learning-for-object-detection-a-comprehensive-review-73930816d8d9>, September 2017
- [3] Joseph Redmon, Santosh Divvala, Ross Girshick, Ali Farhadi, You Only Look Once: Unified, Real-Time Object Detection, <https://arxiv.org/pdf/1506.02640v5.pdf>, May 2016
- [4] Image stacks, <https://helpx.adobe.com/photoshop/using/image-stacks.html>, February 2017
- [5] Fill and stroke selections, layers, and paths, [https://helpx.adobe.com/photoshop/using/filling-stroking-selections-layers-paths.html#content\\_aware\\_pattern\\_or\\_history\\_fills](https://helpx.adobe.com/photoshop/using/filling-stroking-selections-layers-paths.html#content_aware_pattern_or_history_fills), February 2017
- [6] How to Remove People, Shadows, Objects, and Other Unwanted Items from Photos, <https://www.movavi.com/support/how-to/how-to-remove-people-and-objects-from-photos.html>
- [7] Aurélien Géron, Hands-On Machine Learning with Scikit-Learn and TensorFlow, O'Reilly Media, March 2017, p. 4, 253, 355, 363
- [8] Sarah Guido, Andreas Müller, Introduction to Machine Learning with Python, O'Reilly Media, October 2016, p. 9, 13, 33, 34, 127
- [9] Sujit Pal, Antonio Gulli, Deep Learning with Keras, Packt Publishing, April 2017
- [10] Adam Gibson, Josh Patterson, Deep Learning A Practitioner's Approach, O'Reilly Media, August 2017, p. 65, 66, 71, 78, 96
- [11] Keras, <https://keras.io/>
- [12] TensorFlow, <https://www.tensorflow.org/>
- [13] Numpy, <http://www.numpy.org/>
- [14] Django, <https://www.djangoproject.com/>
- [15] PythonEditors, <https://wiki.python.org/moin/PythonEditors>
- [16] M. Andriluka, S. Roth, B. Schiele, Monocular 3D Pose Estimation and Tracking by Detection, <https://www.mpi-inf.mpg.de/departments/computer-vision-and-multimodal->

[computing/research/people-detection-pose-estimation-and-tracking/monocular-3d-pose-estimation-and-tracking-by-detection/#c5123](#), June 2010

- [17] Sergey Ioffe, Christian Szegedy, Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift, <https://arxiv.org/pdf/1502.03167.pdf>, March 2015
- [18] Installing Anaconda on Linux, <https://docs.anaconda.com/anaconda/install/linux>
- [19] Installing Anaconda on macOS, <https://docs.anaconda.com/anaconda/install/mac-os>
- [20] Installing Anaconda on Windows, <https://docs.anaconda.com/anaconda/install/windows>
- [21] Afzal Godil, Roger Bostelman, Will Shackleford, Tsai Hong, Michael Shneier, Performance Metrics for Evaluating Object and Human Detection and Tracking Systems, <https://nvlpubs.nist.gov/nistpubs/ir/2014/NIST.IR.7972.pdf>, July 2014
- [22] Scott Fortmann-Roe, Understanding the Bias-Variance Tradeoff, <http://scott.fortmann-roe.com/docs/BiasVariance.html>, June 2012

## Линкове

1. Хранилище на кода - [https://github.com/nikolaystanishev/remove\\_4ovek](https://github.com/nikolaystanishev/remove_4ovek)
2. Тази документация може да бъде намерена на -  
<https://drive.google.com/open?id=1mvRtSqtXHjteKFsepMG5rjhxrwuLfNmt>

# Съдържание

ЗАДАНИЕ за дипломна работа .....	3
Отзив на Научният ръководител .....	3
Увод.....	4
Глава Първа .....	5
1 Проучване .....	5
1.1 Съществуващи решения .....	5
1.1.1 Откриване на обект на снимка.....	5
1.1.1.1 Хистограми от Ориентирани Градиенти (ХОГ) за откриване на хора (HOG Person Detector).....	5
1.1.1.2 Регионална Конволюционна невронна мрежа (P-KHM) (R-CNN) .....	5
1.1.1.3 Бърза P-KHM (Fast R-CNN).....	6
1.1.1.4 По-бърза P-KHM (Faster R-CNN).....	7
1.1.1.5 Детектор с единичен изстрел (ДЕИ) (SSD (Single Shot Detector)) .....	8
1.1.1.6 Ти Само Гледаш Веднъж (ТСГВ) (YOLO (You Only Look Once)).....	9
1.1.2 Премахване на човек от снимка.....	10
1.1.2.1 Adobe Photoshop .....	10
1.1.2.1.1 Метод чрез наслагване на много снимки.....	10
1.1.2.1.2 Метод чрез освежаване на съдържанието.....	10
1.1.2.2 Movavi Photo Editor .....	10
1.2 Подходи и технологии.....	10
1.2.1 Машинно самообучение .....	10
1.2.2 Контролирано машинно самообучение .....	11
1.2.3 Безнадзорно машинно самообучение.....	11
1.2.4 Дълбоко самообучение .....	11
1.2.5 Конволюционни невронни мрежи .....	12
1.2.6 Функция на Загубата.....	13
1.2.7 Функция на Активиране .....	14
1.2.8 Оптимизатор .....	15
1.2.9 Хипер параметри (Hyperparameters).....	15
1.2.10 Нормализиране на групата (Batch Normalization) .....	16
1.2.11 Начини за оценяване на алгоритъма.....	16

1.2.11.1	Използвани параметри .....	16
1.2.11.2	Използвани метрики .....	16
1.2.11.3	Компромисът Отклонение-Вариране (Bias-Variance Tradeoff).....	17
1.2.12	Без-Максимално Потискане (Non-Max Suppression).....	18
1.2.13	Набор от данни.....	18
Глава Втора.....		19
2	Изисквания и използвани средства .....	19
2.1	Функционални изисквания на проекта .....	19
2.2	Развойна среда, език и библиотеки .....	19
2.3	Общо описание на алгоритъма .....	20
2.4	База Данни .....	22
2.5	Използван набор от данни.....	22
Глава Трета .....		23
3	Имплементация .....	23
3.1	Комуникация с потребителите .....	23
3.1.1	Приемане на видеоклип.....	23
3.1.2	Изпращане на видеоклипа за обработване .....	23
3.1.3	Получаване на готовата снимка.....	23
3.2	Откриване на хора на снимка .....	23
3.2.1	Файл с конфигурациите за откриването на хора.....	24
3.2.2	Обработка на Набора от Данни.....	24
3.2.2.1	Скрипт за изтегляне на Набора от Данни .....	24
3.2.2.2	Обработка на Набора от Данни.....	24
3.2.3	Моделиране на Конволюционната невронна мрежа за откриване на хора ...	27
3.2.3.1	Моделиране на архитектурата на Конволюционната невронна мрежа за откриване на хора .....	27
3.2.3.2	Създаване на модела на Конволюционната невронна мрежа за откриване на хора	30
3.2.4	Трениране на Конволюционната невронна мрежа за откриване на хора .....	30
3.2.4.1	Функция на Загуба .....	31
3.2.4.2	Оптимизатор .....	33
3.2.4.3	Тренировъчни експерименти .....	33
3.2.5	Оценяване на Конволюционната невронна мрежа за откриване на хора .....	37

3.2.6	Правене на предположения от Конволюционната невронна мрежа за откриване на хора .....	39
3.2.7	Файл за управление на процеса за откриване на хора .....	40
3.3	Изготвяне на снимка без хора .....	40
3.3.1	Обработване на видеото .....	41
3.3.2	Вземане на предположенията на Конволюционната невронна мрежа за откриване на хора за видеоклип .....	41
3.3.3	Изготвяне на снимка от статичен видеоклип .....	41
Глава Четвърта	.....	43
4	Ръководство на потребителя .....	43
4.1	Инсталация на пакети .....	43
4.2	Файл за контрол на процеса свързан с Конволюционната невронна мрежа за откриване на хора .....	43
4.3	Трениране на Конволюционната невронна мрежа за откриване на хора .....	44
4.4	Стартиране на интернет приложението .....	44
4.5	Използване на Jupyter Notebook .....	45
4.6	Използване на интернет приложението .....	45
4.7	Правене на предположения от Конволюционната невронна мрежа за откриване на хора .....	46
4.7.1	Предположения за Наборът от Данни за трениране .....	46
4.7.2	Предположения за Наборът от Данни за валидиране .....	47
4.7.3	Предположения за Наборът от Данни за тестване .....	47
Заклучение	.....	48
Използване литература	.....	49
Линкове	.....	51
Съдържание	.....	52