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

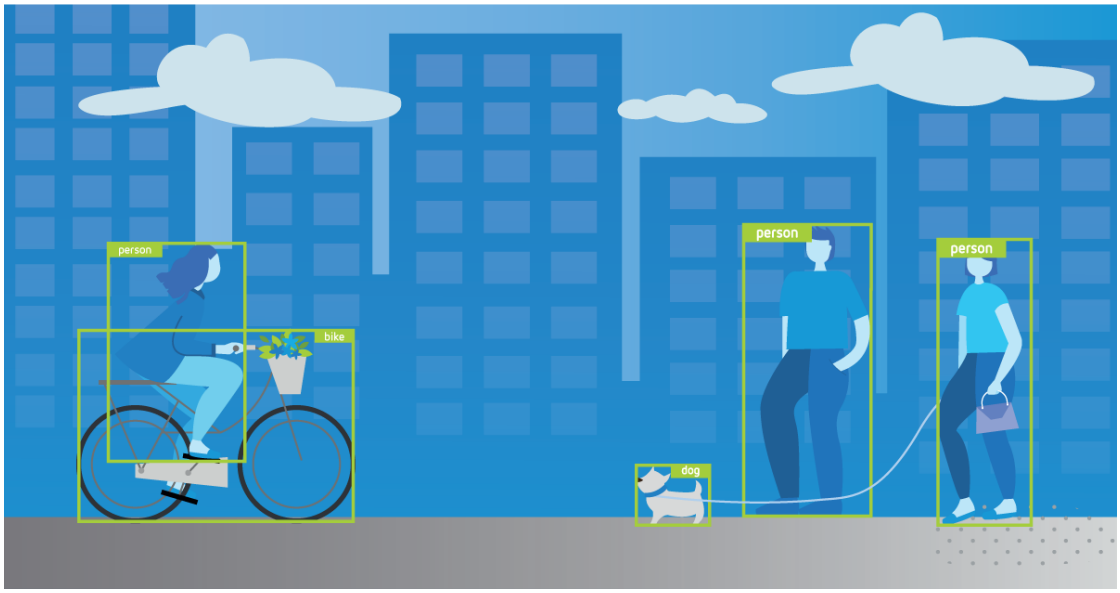
Проект по предметот

Дигитално процесирање на слика

Тема: Алгоритми за сегментација на слики базирани на длабоко учење

YOLO

(You Only Look Once)



Ментор:

Проф. Д-р Ивица Димитровски

Изработила:

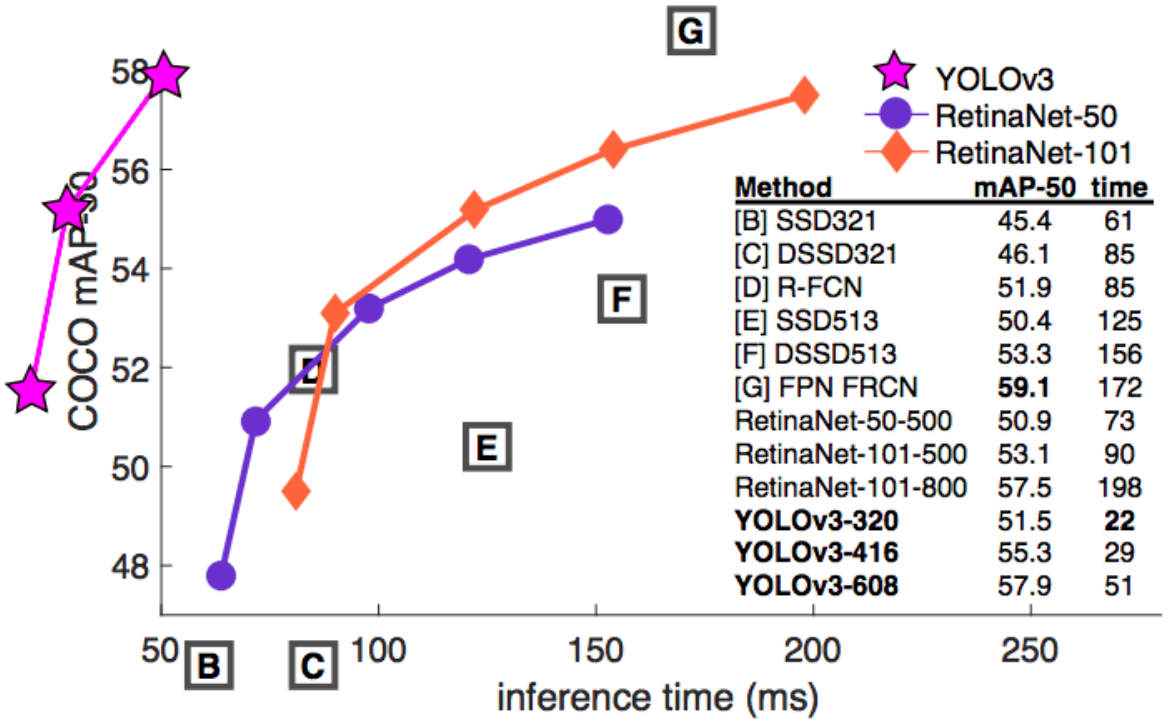
Тијана Атанасовска 196014

1. Вовед

Компјутерска визија е дел од најпопуларните области на истражување во науката. Претставува пресек од повеќе области: Компјутерски науки, Математика, Инженерство, Роботика, Физика, Биологија. Примената ја согледуваме во секојдневието. Мобилни апликации користат препознавање на лице за примена на различни филтри и отклучување на телефон. Надгледување со камери на јавни локации со цел детектирање на сомнителни однесувања. Во развојот на ова поле придонесе и автомобилската индустрија со паметните автомобили кои се движат сами и препознаваат сообраќајни знаци и семафори.

Да се извлече знаење од слики не е нова идеја, но вистински пресврт настанува во 2012 со примена на длабока невронска мрежа – AlexNet. Оттогаш се развиени повеќе алгоритми и натпреварот за подобри перформанси продолжува. Дел од нив е YOLO (You Only Look Once) – алгоритам за детекција на објекти кој користи карактеристики научени со длабока конволуциска невронска мрежа. Во моментот има 5 верзии. Во овој проект ќе се обработи верзијата 3 – имплементација на код и тренирање на моделот.

Уникатноста на YOLO е во обработката на целата слика и неговата брзина од 45fps (frames per second) што го прави најбрз алгоритам за детекција на објекти во реално време. Останатите пристапи со R-CNN генерираат потенцијални региони и потоа ги класифицираат што претставува долг процес, додека овој систем ја обработува целосната слика и енкодира глобални карактеристики.



Слика 1. Перформанси на алгоритамот споредени во однос на други алгоритми

Конволуциски невронски мрежи (CNN)

Конволуциските невронски мрежи се алгоритми кои се користат во длабокото учење за препознавање објекти и извлекување знаење од слики. Инспирацијата за нивно креирање доаѓа од биолошките процеси во кои невроните примаат стимули од видното поле. Секој неврон реагира на различен дел од видното создавајќи клучна информација за пренос во мозокот.

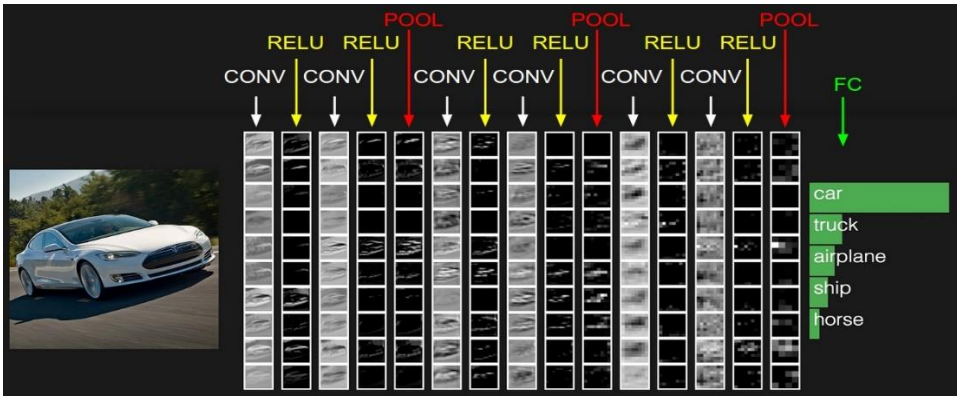
Името доаѓа од примената на математичката операција конволуција која претставува производ на две функции. Во полето на компјутерска визија означува движење на филтер-матрица низ пикселите од оригиналната слика и создавање на нова слика како производ.

Невронските мрежи примаат влез – вектор и го трансформираат низ повеќе скриени слоеви за да продуцираат излез. Секој скриен слој е изграден од неврони кои се целосно поврзани (fully connected) со невроните од претходните слоеви, а се независни од невроните во истиот слој. Последниот слој е излезен слој во кој се преведуваат трансформациите во излезна класа. Иако наоѓаат примена во многу области, кога станува збор за обработка на слики овие мрежи се неефикасни поради големиот број на тежини и останати параметри кои бараат многу ресурси, а дополнително водат до overfitting и слаби резултати.

Конволуциските невронски мрежи очекуваат влез – слика и целата архитектура е градена за да се справи полесно со големиот број на компјутерски пресметки и да ги извлече клучните карактеристики. Карактеристично е што слоевите се тродимензионални – ширина, висина и длабочина. Длабочината претставува простор на бои (RGB, HSV, HSL…). Невроните во секој слој се поврзани со **дел** од невроните во претходниот слој. Често се користат **споделени тежини**, што е различно од регуларните невронски мрежи. Филтрите кои се аплицираат врз влезот не се зависни од големината на сликата. Односно, RGB слика од 10x10px и RGB слика од 100x100px минуваат низ ист филтер 5x5x3 (3 е длабочината = RGB каналите).

Постојат различни архитектури, но дел од слоевите се присутни кај сите нив. Најчесто се применува архитектурата:

[Влез – Конволуциски слој – Активациска функција – Базен – Целосно поврзан слој]



Слика 2. Архитектура на конволуциска невронска мрежа

Влез – претставува пиксел-вредностите на слика со димензии висина x ширина x просторот со бои.

Конволуциски слој – примена на конволуција со филтри врз влезот. Се пресметува активациска мапа со аплицирање на секој од филтрите. Секој неврон пресметува сума од продукти од вредноста на пикселите и филтерот, што резултира во 2D простор. Предноста на овие мрежи се согледува во локалното поврзување, односно невронот е поврзан со P пиксели, каде P = висина на филтер x ширина на филтер x длабочина на влезот. Секој филтер извлекува различни карактеристики – светлина, рабови, облици. Излезот е слоеви од сите филтери и затоа третата димензија е нивната сума. Влез со димензии *NxNxnum_channels* резултира во излез со димензии *MxMxnum_filters* каде:

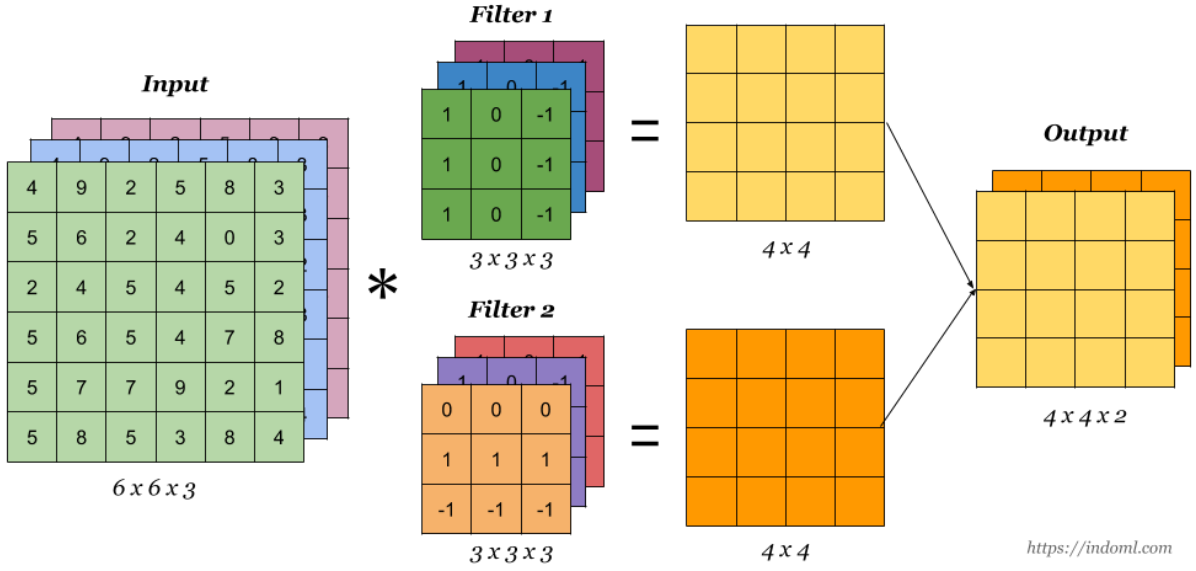
$M = 1 + \frac{N-F+2P}{S}$, F – големина на филтер, P-обвивка со нули, S-чекор на движење

Ефективноста на овој слој зависи од хипер-параметрите кои е потребно да се научат преку тренинг податоци. Такви се обвивка со нули, чекор и длабочина на излез.

Обвивка со нули (zero padding) – додавање на пиксели со вредност 0 околу влезот. Потребата од овој процес е од две причини:

- Димензиите на филтерот не се деливи со димензиите на сликата;
- Пикселите на рабовите на сликата влијаат помалку во резултатот за разлика од тие во центарот кои се опфатени со повеќе лизгачки прозорци за време на конволуцијата.

Чекор (stride) – степен на лизгање на прозорецот, односно колку пиксели да се прескокнат при поместување. Може да биде 1, 2, 3...

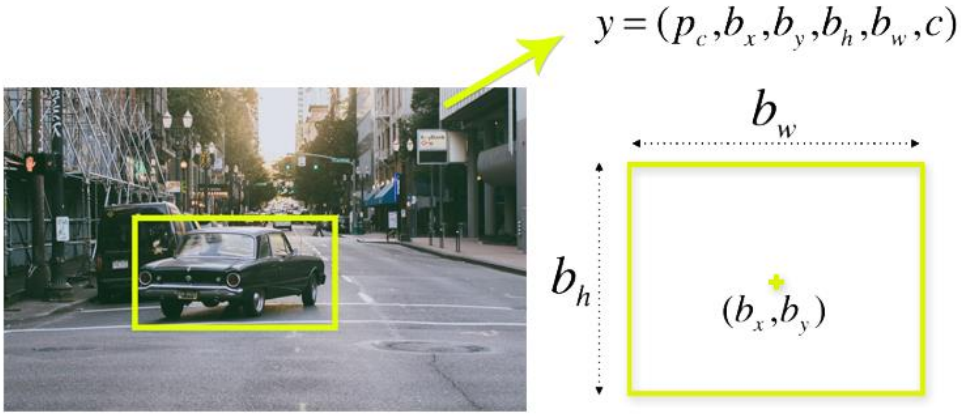


Слика 3. Операции во конволуциски слој

Активациска функција – пресликување на добиените резултати со нелинеарна функција чиј избор е значаен во ефективноста на мрежата.

Базен (pool layer) – намалување на волуменот на пикселите. Се чуваат само дел од пикселите за оптимизација зависно од пристапот кој се користи. Max pooling зема филтер, го дели на региони и од секој регион се зачувува само пикселот со најголема вредност.

Излез - излезот може да варира во зависност од целта на конволуциската мрежа. Доколку се користи за класификација на слика, тогаш излез е само класата. При примена во локализација и детекција на објекти, излезот е вектор со повеќе информации кои вклучуваат димензии на гранична кутија (bounding box), координати на центарот, класа и веројатност за присуство на објект во тој регион.



Слика 4. Излез-вектор од конволуциска невронска мрежа

2. Гранична кутија на регресија (bounding box regression)

Излезот од конволуциска невронска мрежа за класификација на слика е единствен параметар кој укажува на која класа припаѓа сликата која сме ја задале на влез. Алгоритмите кои детектираат повеќе објекти на слика и притоа вршат локализација имаат имплементација на гранична кутија на регресија. Во овој случај излезот од невронската мрежа е вектор (p, b_x ,b_y ,b_h ,b_w, c). Првиот параметар е веројатност за присуство на некоја од класите на регионот во границите на кутијата. Следните два параметри се координатите на центарот на објектот. Третиот и четвртиот параметар означуваат висина и ширина на објектот, но во однос на целата слика, не само ќелијата која се обработува во невронот. Последниот параметар е нумеричката ознака за класата за тој објект.

3. Пресек врз унија (intersection over union) и non-maximum suppression

Локализацијата на објекти резултира со повеќе гранични кутии за ист објект. Овој проблем се решава најпрвин со пресек врз унија за да утврдиме дали резултатите претставуваат ист објект. Поставуваме прагова вредност (хипер параметар најчесто поголем од 0.5) која доколку е надмината станува збор за ист објект. Потоа одлуката која од граничните кутии е најсоодветна се носи со помош на non-maximum suppression, односно се избира онаа која има најголема веројатност за присуство на детектираниот објект и се отфрлаат сите останати кутии.

Слика 5. Повеќе детекции на ист објект

4. YOLOv3 архитектура на невронска мрежа

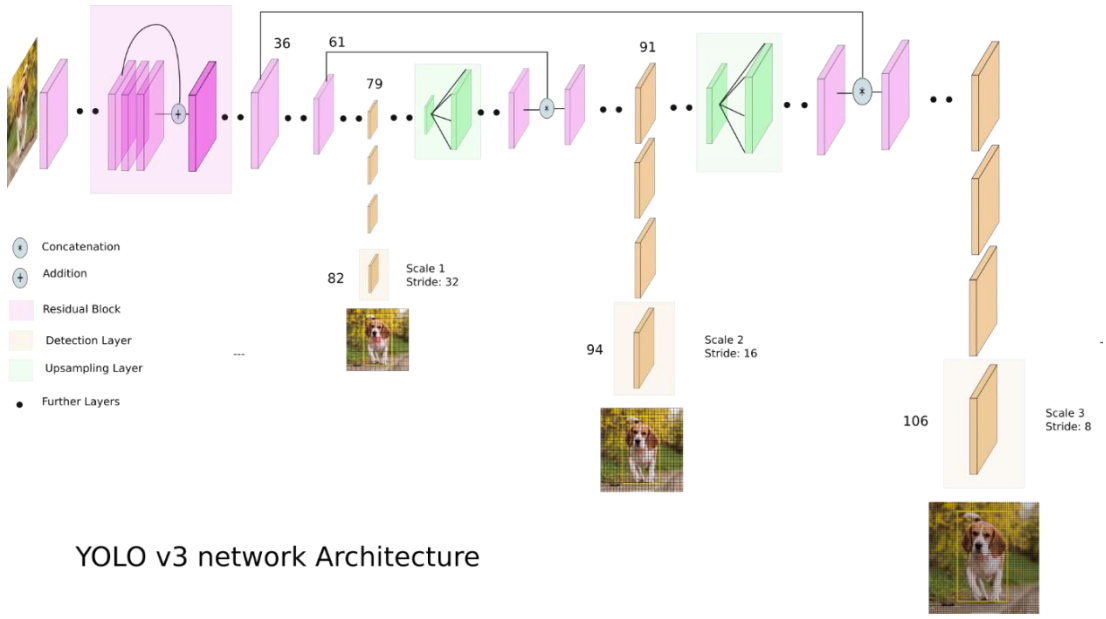
Освен основните слоеви на секоја конволуциска мрежа, архитектурата на невронската мрежа за алгоритамот YOLOv3 содржи дополнителни слоеви. Карактеристични се up-sampling layers и residual blocks.

Up-sampling слоевите ја вршат спротивната функција од Pooling layers. Се користат за зголемување на волуменот на пикселите. Постојат различни начини за да се изврши оваа операција. Во овој алгоритам се користи `cubic interpolation` односно секој пиксел се пресликува во 4 пиксели со иста вредност. Сликата се обработува во три различни димензии што придонесува во детекција на мали детали. Оваа функционалност е имплементирана во верзија 3 и има огромно влијание на перформансите на алгоритамот.

Residual blocks уште наречени и скратени конекции (skip-connections) се блокови кои користат карактеристики научени од подалечни слоеви. Во регуларните невронски мрежи секој слој k на влез прима карактеристики само од слојот k-1, што не е случај во овие блокови.

Архитектурата на YOLOv3 е изградена од 106 слоеви. Од нив 75 се конволуциски слоеви. Предноста во однос на претходните верзии е во детекциите на карактеристики кои се случуваат во три слоеви (претходно само во еден слој). Во нив се прави намалување на волуменот на влезната слика (down-sampling) со чекор 32 во 82 слој.

Кернелот за детекција е со големина 1x1x(B*(5+C)), каде B – број на гранични кутии кои една ќелија може да ги предвиди, C – број на класи. Доколку влезот е слика со големина 416x416px од COCO датасетот (B = 3, C = 80), тогаш излезот во 82 слој е мапа на карактеристики со големина 13x13x255 каде е применето намалување со чекор 32 и кернел за детекција. Во 94 слој се применува намалување со чекор 16 и се добива мапа со големина 26x26x255. Последното намалување е во 106 слој и излезот е 52x52x255. Овој пристап овозможува подобрување во детекција на мали објекти, така што првиот слој служи за детектирање на големи, а секој следен за многу помали објекти.
Алгоритамот пресметува ((52x52)+(26x26)+(13x13))x3=10647 гранични кутии, редуцирани со IoU и non-maximum suppression.



Слика 6. Архитектура на невронска мрежа користена во YOLOv3

5. Детекција на објекти во ќелија

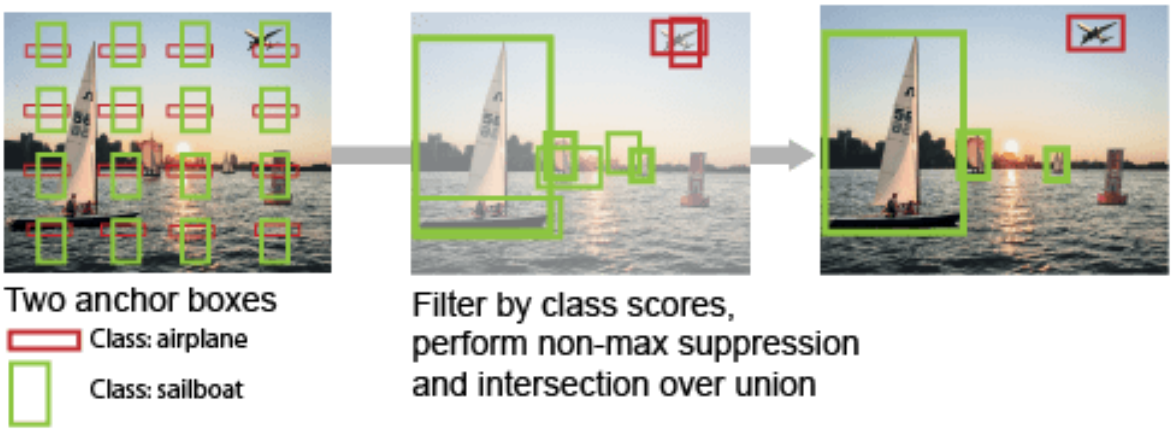
Ќелијата во која се наоѓаат централните координати на објектот врши детекција на истиот. Во првите верзии алгоритамот препознавал само еден објект во дадена ќелија. Подоцна со имплементација на anchors се надминува овој проблем. Anchors претставуваат однапред дефинирани кутии со висина и ширина за детекција на објекти. Тие имаат различни големини. При детекција на објект во ќелијата, се пресметува IoU за секој anchor и дадениот објект и се зема оној со најголем пресек. Ова го менува и излезот, односно нема да биде изграден од векторот $(p, b_x, b_y, b_h, b_w, c)$ туку ќе претставува вектор од вектори со податоци за секој anchor. Доколку при креирање на алгоритамот се предефинирани 2 anchors, тогаш излезот би бил
$((p, b_x, b_y, b_h, b_w, c), (p, b_x, b_y, b_h, b_w, c))$.

Постојат два специфични случаи кога се користи овој метод за повеќе објекти.

- Предефинирани се n anchors, а во ќелијата постојат m објекти, така што n>m. Во тој случај првата вредност од векторот за кутијата која нема детектирано објект ќе укажува дека во тој простор не постои објект, а останатите вредности се недефинирани.

- Предефинирани се n anchors, а во ќелијата постојат m објекти, така што n<m. Во овој случај се детектираат само n објекти – тие со најголеми вредности за IoU. Ова е причина за не-детектирање на објекти кои се премногу близу.

Овие кутии може да се креираат рачно, доколку знаеме какви објекти се очекуваат (нивната висина и ширина) или со помош на машинско учење преку тренирање на алгоритам врз тренинг податоци и укажување на големината на објектите.
Најголемата предност на YOLOv3 е во овој пристап со anchors, кој овозможува да се евалуираат сите предикции наеднаш што е различно од алгоритмите базирани на лизгачки прозорец кој пресметува предикција за секој прозорец во единица време. Детекција на објекти, енкодирање на карактеристики и класификација на објектот се извршуваат заедно што води до значително подобри перформанси.
Оваа верзија користи 9 anchors, по 3 за секоја големина на сликата од каде следуваат 3 предикции за објекти по ќелија.



Слика 7. Детекција на објекти со anchors

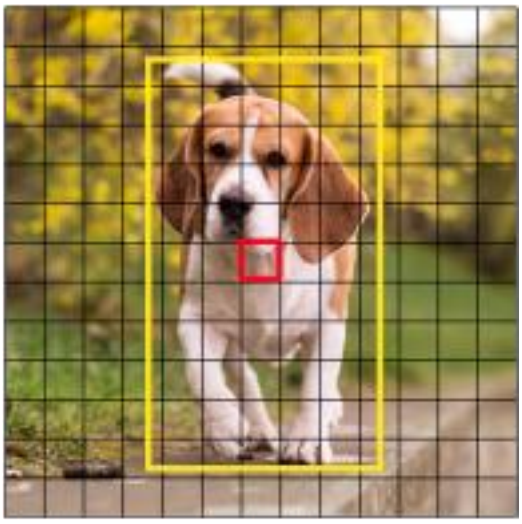
6. Предикција на излез

Вредностите b_x ,b_y ,b_h ,b_w во векторот (p, b_x ,b_y ,b_h ,b_w, c) кои означуваат централни координати, висина и ширина на граничната кутија на предикцијата кој се добива при излезот на алгоритамот се добиваат според формулите прикажани на слика 7. Координатите на горниот лев агол од ќелијата која го детектира објектот се означени со c_x ,c_y. Димензиите на anchors за кутијата се претставени со p_w ,p_h. Се користи Сигмоид функција за пресметување на релативни вредности во однос на ќелијата во која се наоѓа објектот наместо апсолутните вредности во однос на целата слика и нивна нормализација.

$$b_x = \sigma(t_x) + c_x$$
$$b_y = \sigma(t_y) + c_y$$
$$b_w = p_w e^{t_w}$$
$$b_h = p_h e^{t_h}$$

Слика 8. Формули за пресметување на вредностите во излезниот вектор

Пример: Да ја земеме сликата 8. Ако предикцијата за центарот е (0.4, 0.7) ова означува дека центарот е во (6.4, 6.7) на 13x13 мапата со карактеристики бидејќи координатите на горниот лев агол се (6,6). Оваа нормализација не дозволува да се надмине вредноста на ќелијата, односно да даде резултат 7.2 што би означило дека објектот е во соседната ќелија.
Димензиите на граничната кутија се пресметуваат со помош на логаритамска трансформација на излезот и множење со димензиите на anchor (формулите 3 и 4 од слика 7).
Предикцијата во резултатот се нормализира со големината на сликата. Иако детекцијата на објект е на ниво на ќелија, граничната кутија се пресметува во однос на целата слика што овозможува опфаќање на целиот објект.
За сликата со кучето, доколку предикциите b_x ,b_y се (0.3, 0.8) тогаш висината и ширината на предикцијата во 13x13 мапа со карактеристики би била (13x0.3 , 13x0.8).



Слика 9. Детектирање на објект чиј централни координати се во црвеното поле

7. Имплементација на алгоритамот во код

Имплементацијата на алгоритамот во код ја изведуваме во програмскиот јазик Python и библиотеката PyTorch. Оваа open-source библиотека нуди манипулации потребни во компјутерската визија и НЛП. Овозможува акцелерација со користење на GPU и градење на невронски мрежи со модулот torch.nn.
Големиот број на слоеви во YOLOv3 поттикнува да формираме текстуална датотека во која ги запишуваме клучните информации за креирање на секој блок и со помош на код да направиме автоматизирано градење на конволуциската невронска мрежа. Еден блок може да содржи повеќе слоеви, како што е конволуцискиот блок во кој се вградени информации за градење на convolutional layer, batch normalizing layer, activation layer.
Разликуваме 5 слоеви – convolutional, shortcut, upsample, route, yolo.

- Convolutional block:
  Batch Normalize Layer се користи за нормализација на вредностите на влезот. Идеално, овие вредности би требало да се фиксирани околу нулата, со мала варијација. Се користи само во тренинг множеството за да го направиме моделот неосетлив на големите отстапувања од одредени слики. За да се постигне ова се користи техниката на feature scaling. Ова измазнување се користи по

конволуцијата за која се запишани хипер параметрите - број на филтери, обвивка со нули и чекор на движење. Пресметките завршуваат со примена на активациска функција која во овој случај е LeakyReLU.

- Shortcut block:
  Овде користиме знаење научено од претходните слоеви и правиме конкатенација на карактеристиките научени во претходниот слој и слојот запишан како аргумент (from).
- Upsample block:
  Зголемување на мапата на карактеристики со чекорот кој е запишан и користење на bilinear upsampling.
- Route block:
  Слично со shortcut block, но овде не мора да се земат карактеристики од претходниот слој и некој подалечен туку од два подалечни слоја. Излезот е конкатенација на тие слоеви.
- YOLO block:
  Овој блок се појавува три пати во мрежата и со него е имплементирана детекцијата на објекти објаснето погоре. Запишани се ширина и висина за 9 anchors – по три за секој слој на детекција, бројот на класи и праговата вредност за отфрлање на кутиите во non-maximum suppression.
  Во нашата имплементација овие слоеви се со големина 13x13, 26x26, 52x52.

Иако секоја конволуциска мрежа користи pooling layer, овде користиме конволуциски слој со чекор 2 за намалување на мапите на карактеристики со цел зачувување на low-level карактеристики што е проблем во pooling.

```
[convolutional]
batch_normalize=1
filters=64
size=3
stride=2
pad=1
activation=leaky
```

```
[shortcut]
from=-3
activation=linear

[upsample]
stride=2

[route]
layers = -1, 61
```

```
[yolo]
mask = 0,1,2
anchors = 10,13,
16,30,  33,23, 30,61,
62,45,  59,119,
116,90,  156,198,
373,326
classes=80
num=9
ignore_thresh = .5
truth_thresh = 1
random=1
```

Од библиотеката torch.nn користиме ModuleList(). Оваа листа има предност во однос на обичните листи бидејќи очекува елементи чии класи наследуваат од класата Module. Таквите елементи ја имплементираат функцијата forward која се повикува кога ги градиме слоевите во конволуциската невронска мрежа. Ние користиме SequentialModule, со што сме сигурни дека влезот во алгоритамот ќе мине низ сите трансформации од имплементираните слоеви пред да добиеме излез. Доколку не постоеше, после секој слој би добивале излез кој треба да го пуштаме на следен влез итн. PyTorch има вградено имплементација за дел од слоевите кои ги повикуваме со предавање на аргументите.

```
nn.Conv2d(prev_filters, filters, kernel_size, stride, pad, bias = bias)
nn.BatchNorm2d(filters)
nn.LeakyReLU(0.1, inplace = True)
nn.Upsample(scale_factor = 2, mode = "nearest")
```

За слоевите shortcut и route го пишуваме целосниот код (векторски збир на мапите со карактеристики) и ги обвиткуваме со класата EmptyLayer која мора да наследува од nn.Module за да се прикачи во ModuleList().
YOLO го имплементираме со Detection Layer кој ги чува anchors и наследува од класата nn.Module.
Класата Darknet ја претставува конволуциската невронска мрежа и аплицирање на нејзините пресметки врз влезот слика. Функцијата forward прима слика како аргумент-влез, а враќа тензор од вектори-детекции за објектите на влезот. Во класата се чува листата од блокови кои претходно ги изградивме и сликата минува низ сите нив. Мапите на активација добиени после секој слој се чуваат во речник бидејќи се потребни за ре-искористување на научените карактеристики што се користи во Route Layer. Математичките пресметки, објаснети во секција 7, потребни за пресметување на граничната кутија за предикцијата на објект (bounding box) се извршуваат во функцијата predict_transform. Излезот е тензор со големина B x 10647 x 85 каде:
B – број на слики во пакетот на влез
10647 – број на предвидени гранични кутии
85 – атрибути за секоја гранична кутија (80 класи, висина, ширина, централни координати и веројатност за присуство на објект во кутијата).

Следува прочистување на определените гранични кутии со IoU и Non-maximum suppression извршено во функцијата write_results(). Првин, се доделува вредност нула за сите кутии чија веројатност за присуство на објект е помала од зададениот threshold.

```
conf_mask = (prediction[:,:,4] > confidence).float().unsqueeze(2)
prediction = prediction*conf_mask
```

Итерираме низ сите слики во влезниот пакет и за секоја гранична кутија пресметана за определената слика ги зачувуваме индексот и вредноста на класата која има најголема веројатност.

```
max_conf, max_conf_score = torch.max(image_pred[:,5:5+ num_classes], 1)
max_conf = max_conf.float().unsqueeze(1)
max_conf_score = max_conf_score.float().unsqueeze(1)
seq = (image_pred[:,:5], max_conf, max_conf_score)
image_pred = torch.cat(seq, 1)
```

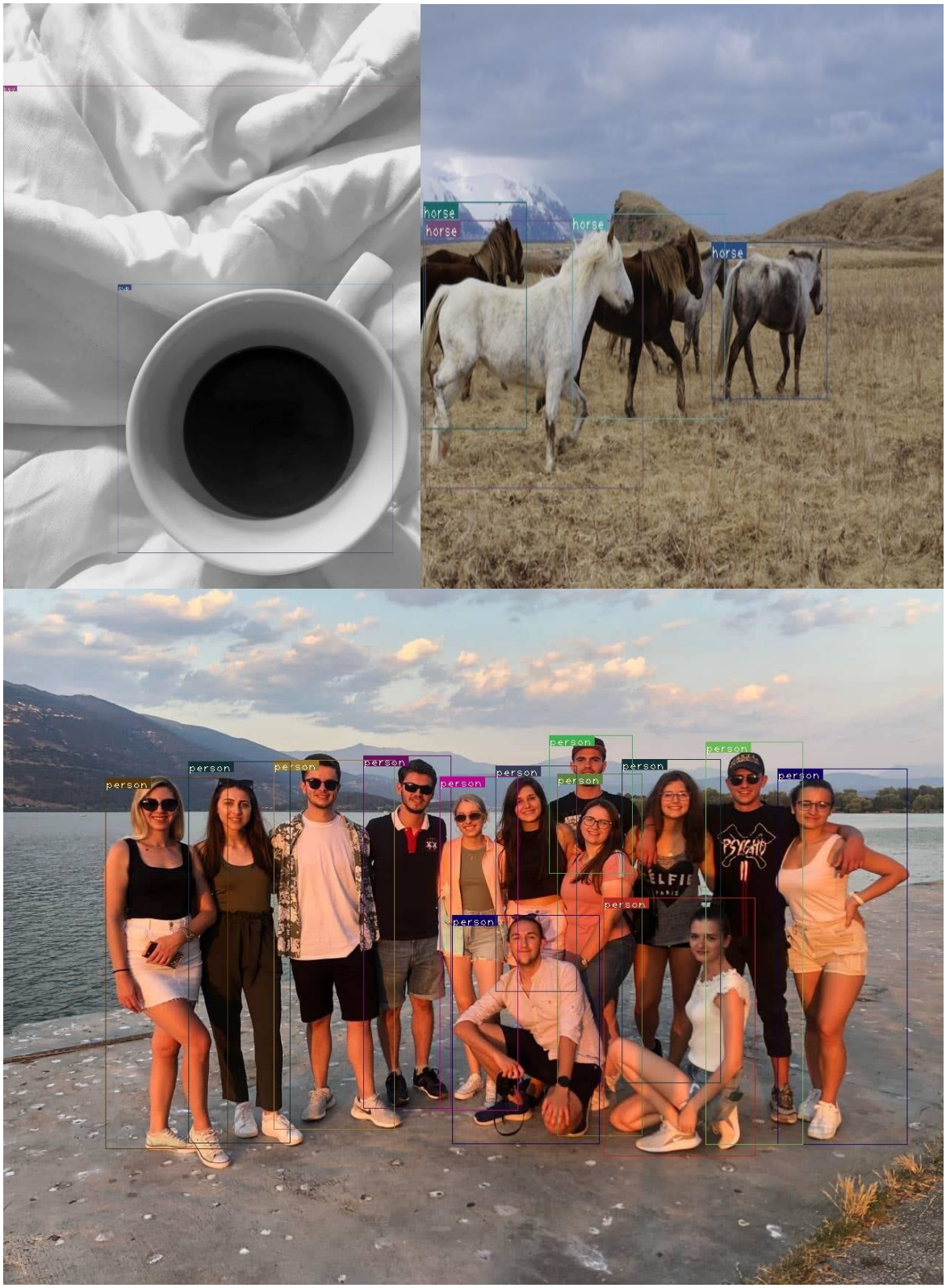
Бидејќи една класа може да има повеќе детекции чии вредности го поминуваат threshold, со функцијата `unique` ги отстрануваме дупликати вредностите.
За секоја класа ги земаме сите предвидени гранични кутии, ги сортираме според веројатностите за присуство на објект и за секој пар од гранични кутии пресметуваме IoU. За гранични кутии чиј сооднос е поголем од поставениот threshold за IoU, се зачувува онаа со поголема веројатно за присуство на објект, а другата се отфрла.

Функцијата forward во класата Darknet враќа тензор со големина Dx8 каде D е бројот на сите реални детекции (кои ги исполнуваат горе-наведените услови) и за секоја од нив се чуваат 8 атрибути:
Индекс на сликата од влезниот пакет на која припаѓа детекцијата, 4 координати за граничната кутија, веројатно за присуство на објект, индекс и вредност на класата со најголема веројатност.

Во кодот имаме имплементирано и детекција на објекти во реално време, каде може да ја забележиме брзината иако постои мало задоцнување.
Тежините кои ги користиме во невронската мрежа се преземен, но истите би можеле и да ги учи алгоритамот за време на тренирање што е важен момент од машинското учење.

8. Тренирање на моделот

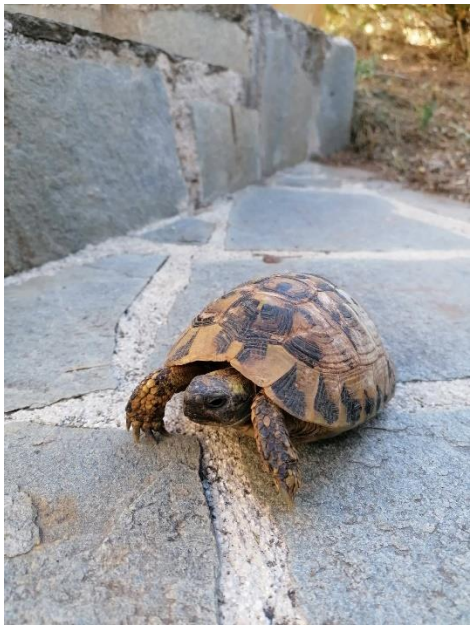
Клучна улога во перформансите на секој алгоритам базиран на длабоко учење е тренинг множеството кое се користи. За алгоритмите во машинска визија најчесто се користи COCO дата-сетот. Содржи 164 илјади слики од кои 83К се во тренинг множеството и 41К во валидациското и тест множеството. Препознава 80 категории (растение, градба, човек...) и 91 под-категории (цвеќе, куќа, жена...). Во имплементацијата на кодот во овој проект се користи овој дата-сет.
Возможно е и креирање на сопствен дата-сет. За секоја слика од тренинг множеството се креира текстуална датотека во која се запишуваат класата на објектот, координатите на центарот на објектот, ширина, висина. Доколку постојат повеќе објекти на сликата тогаш се пишуваат податоците за секој од нив во датотеката. Различни open-source софтвери го олеснуваат формирањето на сопствен дата-сет, меѓутоа овој процес не е едноставен. Потребно е да имаме огромен број на слики, различни агли, светлина и големини за секој објект.



Детекција на објекти со алгоритамот YOLOv3

9. Недостатоци во алгоритамот

Како и секој алгоритам базиран на невронски мрежи, така и YOLOv3 е зависен од тренинг податоците. При тестирање со различни слики, забележав дека во COCO дата-сетот не постојат слики од некои животни. Доколку на влез ја зададеме сликата 9, излезот е “No detections were made”, воопшто не препознава објект.



Слика 9. Влез кој алгоритамот не го препознава

Точноста на детекцијата и бројот на детектирани објекти зависи и од anchors со кои определуваме колку објекти може да се детектирани во една ќелија. На слика 10 забележуваме дека поради близината на објектите, поголем дел од нив треба да бидат препознаени во една ќелија. Поради бројот на anchors, резултира во не-детектирање на објект (објектот обележан со црвено).

Слика 10. Излез-слика во кој не се детектирани сите објекти

10.Заклучок:

Постојат различни алгоритми за детекција на објекти, меѓутоа од големо значење се нивните перформанси. Нивната употреба во значајни области како автомобили кои сами се управуваат, безбедносни системи за детектирање на сомнително однесување користени од безбедносни сили води кон создавање на програми чии грешки тежнеат кон нула. За подобри перформанси влијаат тренинг податоците, тежините и архитектурата на невронската мрежа кои водат до успешно препознавање на објекти.

Користена литература:

[1]https://www.youtube.com/watch?v=RTlwl2bv0Tg&list=PLkDaE6sCZn6Gl29AoE31iwdVwSG-KnDzF&index=29

[2]https://www.mathworks.com/help/vision/ug/anchor-boxes-for-object-detection.html

[3]https://blog.paperspace.com/how-to-implement-a-yolo-v3-object-detector-from-scratch-in-pytorch-part-5/

[4]https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

[5] http://cs231n.stanford.edu/

[6]https://thebinarynotes.com/how-to-train-yolov3-custom-dataset/