

الجمهورية العربية السورية
المعهد العالي للعلوم التطبيقية والتكنولوجيا
قسم المعلومات
العام الدراسي 2020/2019

مشروع تخرج

أعد لنيل درجة الإجازة في هندسة المعلومات

نظام تقدير التدفق المروري باستخدام الرؤيا الحاسوبية

تقديم

انوار ابراهيم

إشراف

د. أصف جعفر د. عمر حمدون م. سيما أدهم

2019/2020

الخلاصة

Abstract

Translate your abstract here.

المحتويات

Contents .1

IX.....	الفصل الأول	2.
IX.....	التعريف بالمشروع	
ix	2.1. الهدف من المشروع	
ix	2.2. متطلبات المشروع	
ix	2.2.1. المتطلبات الوظيفية	
1.....	الفصل الثاني	3.
1.....	3.1. مقدمة	
1.....	3.2. الرؤيا الحاسوبية	
1.....	3.2.1. خوارزميات كشف الأشكال	
4.....	3.2.2. كشف الأشكال باستخدام Hog Features	
	3.2.3. Region-based Convolutional Neural Networks (R-CNN) الشبكات العصبونية التلافيفية المتعلقة بالمنطقة	
6.....	3.2.4. Spatial Pyramid Pooling (SPP-net) تجميع الهرم المكاني	
6.....	3.2.5. Fast R-CNN	
	3.2.6. Faster R-CNN [1] (Faster Region-Based Convolutional Neural Networks)	
9.....	3.2.7. خوارزمية النظر لمرة واحدة "You Only Look Once" [4] YOLO	
22.....	الفصل الثالث	4.
22.....	بيئة التطوير والأدوات البرمجية المستخدمة	
	يقدم هذا الفصل شرحاً عن البيئة والأدوات المستخدمة في تنجز العمل، بالإضافة إلى أهم المكتبات والتوابع المستخدمة ضمنها.	
22.....	4.1. مقدمة	
22.....	4.2. لغة البرمجة Python	
22.....	4.3. لغة البرمجة C#	
23.....	4.4. Google Colaboratory	
23.....	4.5. Darknet [7]	
24.....	4.6. Cuda Toolkit	
24.....	4.7. CuDNN or NVIDIA CUDA Deep Neural Network	
25.....	4.8. CMake	
25.....	4.9. OpenCV (Open Source Computer Version Library)	

26Microsoft Visual Studio	4.10.
26Alturos.Yolo	4.11.
27OpenCvSharp3-AnyCPU	4.12.
28الفصل الرابع	5.
28التنفيذ العملي للنظام	
28مقدمة	5.1.
29الطريقة الأولى	5.2.
30Colab Notebook مع GPU	5.2.1.
30Darknet	5.2.2.
30تعريف توابع مساعدة	5.2.3.
31colab Dataset إلى بيئة	5.2.4.
32YOLOv4	5.2.5.
32تدريب الشبكة	5.2.6.
32تشغيل كاشف الأشكال الناتج	5.2.7.
33الطريقة الثانية	5.3.
33GPU	5.3.1.
33عملية التدريب	5.3.2.
39بناء التطبيق النهائي	5.3.3.

قائمة الأشكال:

- الشكل 1: a الخرج الناتج عن مصنف القطط والكلاب، b مثال عن المشكلة التي واجهت هذا النوع من المصنفات. 2
- الشكل 2: مثال عن الخرج المطلوب في مسألة كشف الأشكال. 2
- الشكل 3: مثال لتوضيح مشكلة ثبات أبعاد نافذة التنبؤ. 3
- الشكل 4: مثال عن طريقة هرم الصورة. 4
- الشكل 5: مراحل عمل خوارزمية R-CNN. 5
- الشكل 6: شكل توضيحي لمراحل خوارزمية SPP-net. 6
- الشكل 7: تصميم الشبكة في خوارزمية Faster R-CNN. 7
- الشكل 8: الطريقة المتبعة في Faster R-CNN لمعالجة مشكلة الهجوم والمقاييس المتعددة. 8
- الشكل 9: صورة مقسمة إلى شبكة بأبعاد 5x5. 9
- الشكل 10: العلاقة بين معامل الثقة و IOU، المربعات السوداء هي مربعات الحقيقة والحمراء هي المربعات التي تم التنبؤ بها. 10
- الشكل 11: الاحتمالات الشرطية الناتجة عن النموذج الموافقة لكل صف من الصفوف الذي تم تدريب النموذج عليه. 11
- الشكل 12: متحولات خرج النموذج بالنسبة للشبكة. 11
- الشكل 13: مراحل خوارزمية YOLO، والخرج النهائي. 12
- الشكل 14: بنية الشبكة YOLO. 12
- الشكل 15: تمثيل خلية شبكة محاطة بـ 5 مربعات prior بأشكال مختلفة. 16
- الشكل 16: في هذا الشكل يتقاطع الـ 1 prior مع الـ ground truth بأكثر مساحة لذا فهذا المربع هو الذي سيسند لهذا الغرض. 17
- الشكل 17: استخدام مصنفات لوجستية مستقلة لكل صف من الصفوف، لغرض يمكن أن يصنف كـ person، أو كـ woman. 18
- الشكل 18: بنية شبكة darknet-53. 18
- الشكل 19: البنية العامة لأنظمة التعرف على الأشكال. 19
- الشكل 20: مقارنة بين أداء yolov4 مقارنة النماذج الحديثة للكشف عن الأغراض، يمكن بسهولة ملاحظة كون yolov4 قد تفوقت عن النسخة السابقة بحيث تحسن الـ AP و الـ FPS بمقدار 10%، و 12% على الترتيب. 21
- الشكل 21: شعار لغة Python. 22
- الشكل 22: شعار شبكة Darknet. 24
- الشكل 23: شعار CMake. 25

الشكل 24:	شعار مكتبة OpenCV	25
الشكل 25:	شعار حزمة Alturos.Yolo	27
الشكل 26:	flow Chart للبرنامج المستخدم على منصة colab مع YOLOv4	29
الشكل 27:	تعديل الملف Makefile لتأكيد تفعيل GPU, OPENCV	30
الشكل 28:	الصفوف الموجودة في Vehicles-openImageas Dataset، ونسبة كل منها	31
الشكل 29:	تعديل الملف yolov4-custom.cfg لهيئة كاشف الاشكال الناتج للكشف عن الأشكال	33
الشكل 30:	بناء المجلد build لتنزيل مكتبة openCv باستخدام CMake	34
الشكل 31:	تحديد ملفي ال build وال source	34
الشكل 32:	تحديد الplatform التي سنستخدمها لتوليد الملفات	35
الشكل 33:	تغيير نمط العمل في visual Studio من debug إلى release	36
الشكل 34:	نسخ AlexyAB darknet repo باستخدام git	36
الشكل 35:	التعديلات التي سنقوم بإجرائها على ملف Cmakefile من مجلد darknet	37
الشكل 36:	خرج عملية ال detection لـ darknet مع الأوزان المدربة مسبقا و Yolov4	38
الشكل 37:	الصفوف التي يتألف منها التطبيق النهائي	39
الشكل 38:	توابع الصف MainWindow	39
الشكل 39:	الصف Yolo، مع التوابع التي يتألف منها، وتوضيح كيف يتم استدعائها	40
الشكل 40:	تصميم قاعدة المعطيات التي تتألف من الجدول Vehicle، مع توضيح ال attributes التي يتألف منها هذا الصف	42
الشكل 41:	مثال عن طريقة تخزين البيانات المعرفة للسيارة في قاعدة المعطيات	43
الشكل 42:	السيناريو الرئيسي للنظام	43
الشكل 43:	واجهة اختيار أحد النسخ من خوارزمية Yolo	44
الشكل 44:	الواجهة الأساسية للتطبيق	44
الشكل 45:	واجهة عرض السيارات من قاعدة المعطيات وتطبيق فلتر عليها	45
الشكل 46:	واجهة عملية الكشف	45

قائمة الجداول

الجدول 1: مقارنة بين خوارزميات R-CNN, Fast R-CNN, Faster R-CNN من حيث السرعة. 8

قائمة الاختصارات

الرمز	معناه
OCR	Optical Character Recognition
mAP	Mean Average Precision
YOLO	You Only Look Once
GPU	Graphical Processing Unit
RPN	Region Proposal Network
Hog Features	Histogram of Oriented Gradients Features
SVM	Support Vector Machine
R-CNN	Region-based Convolutional Neural Networks
SPP-net	Spatial Pyramid Pooling
VGG	Visual Geometry Group

نعيش اليوم في زمن أصبح فيه الذكاء الصناعي من أساسيات الحياة ابتداءً بأنظمة المساعدة الشخصية مثل Siri, Alexa, Cortana وغيرها من الأنظمة وانتهاءً بسيارات القيادة الذاتية، فقد اجتاحت أنظمة الذكاء الصناعي كل مجالات الحياة وفي هذا البحث سندرس أحد الطرق التي ساهمت فيها أنظمة الذكاء الصناعي في تسهيل عملية مراقبة التدفق المروري وحساب عدد السيارات في كل طريق، لجعل نظام المرور أكثر ذكاءً وموثوقية وقوة، وسنقوم ببناء نظام للتعرف على المركبات على طريق، وملاحقة هذه المركبات، ثم القيام بعدها وحساب سرعة كل منها، وأخيراً تخزينها في قاعدة معطيات، وسنعمد على تطبيقات الذكاء الصناعي من التعرف على الأشكال object detection، ثم ملاحقة الأغراض multi-object tracking، وأخيراً تقنيات معالجة الصورة للتعرف على خواص العربة.

2. الفصل الأول

التعريف بالمشروع

يُهدف هذا الفصل للمشروع، حيث يبيّن فكرة المشروع وأهميته والأهداف المرجوة منه، ويذكر المتطلبات الوظيفية، وغير الوظيفية للمشروع، كما يوضّح الخطة الزمنية للمشروع.

2.1. الهدف من المشروع

نهدف في هذا المشروع إلى بناء نظام لتقدير التدفق المروحي في طريق باستخدام خوارزميات التعلّم العميق، حيث استخدمنا هذه الخوارزميات للتعرف على الأشكال "في حالة مشروعنا قمنا بالتعرف على السيارات"، ثم تطبيق هذه الخوارزميات لملاحقة السيارات التي تمّ التعرف عليها، ثم إيجاد خوارزميات لحساب سرعتها، وعددها، والقيام بتصميم تطبيق برمجي بواجهة سهلة التعامل.

2.2. متطلبات المشروع

نسرد فيما يلي كلاً من المتطلبات الوظيفية وغير الوظيفية للمشروع:

2.2.1 المتطلبات الوظيفية

- 1- التعرف على أشهر خوارزميات التعلّم العميق المتخصصة في التعرف على الأشكال
- 2- التعرف على خوارزمية Yolo بنسخها المختلفة واستخدام هذه الخوارزمية لتدريب نموذج، واستخدامها....

3. الفصل الثاني

الدراسة المرجعية

نبيّن في هذا الفصل المفاهيم النظرية التي تمثل خلفية المشروع، ونشرح فيه بإيجاز أهم الخوارزميات المستخدمة لحل هذا النوع من المسائل.

3.1. مقدمة

سنذكر في هذا الفصل عن التقنيات والمفاهيم النظرية التي سنحتاجها لبناء تطبيقنا، ومن أهم هذه المفاهيم التعرف على الأشكال، وتعقبها... وسنبداً هذا الفصل بالتعريف بالرؤية الحاسوبية وذكر أهم خوارزميات التعرف على الأشكال عبر التاريخ وكيف تطورت كل منها مقارنة بسابقتها، وسنتوسع في شرح الخوارزميات التي استخدمناها في تطبيقنا ومنها خوارزمية Yolo بنسخها المختلفة، ثم سنتكلم عن خوارزمية التعقب المستخدمة.

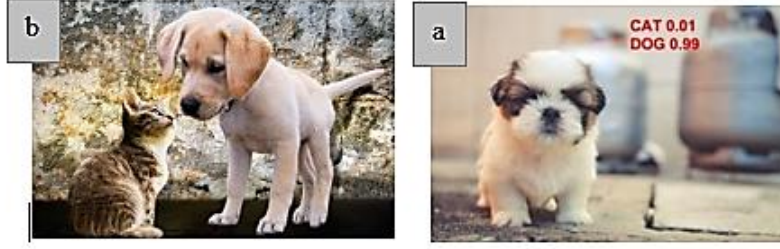
3.2. الرؤية الحاسوبية

إن الرؤية الحاسوبية هي أحد فروع الذكاء الصناعي التي تهدف إلى تدريب الآلات "لترى"، ويهدف على استخلاص معرفة من الـ pixels، أي يسعى لبناء أنظمة يمكنها أن تتقّد المهام التي ينفذها نظام الرؤية البشري، وسنستخدم أحد تطبيقاته في نظامنا ألا وهو كشف الأشكال "Object Detection".

3.2.1 خوارزميات كشف الأشكال

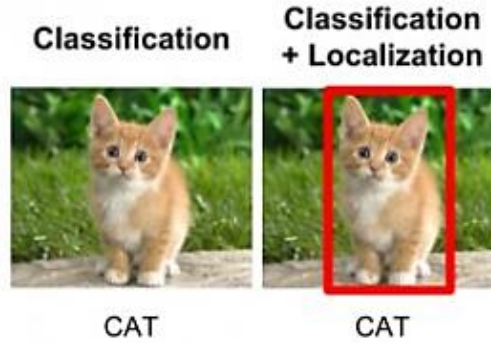
فيما يلي سنتكلم عن تطور خوارزميات كشف الأشكال للوصول إلى Faster R-CNN, YOLO, SSD. وسنبداً من المستوى المبتدأ وصولاً إلى أحدث طرق كشف الأشكال، مراعين محاولة فهم المبدأ وراء الخوارزمية والمنهج المتبع، والخواص المميزة لكل طريقة:

لنبدأ بالحديث عن تصنيف الصور، نعلم أن تصنيف الصور هو العملية التي تأخذ صورة ما وتقوم بالتنبؤ بالغرض الموجود فيها، كمثال يمكن بناء مصنّف للتمييز بين كون الحيوان الموجود في الصورة هو قطة أم كلب، ويأخذ هذا المصنّف كدخل له صورة لقطة أو لكلب ومنتبأ ما هو الصف الذي تنتمي له هذه الصورة.



الشكل 1: a الخرج الناتج عن مصنف القطط والكلاب، b مثال عن المشكلة التي واجهت هذا النوع من المصنفات.

ولكن بعد الحصول على هذا المصنف ظهرت لدينا مشكلة وجود صور تحوي قطة وكلب في نفس الوقت، ما هو الخرج المتوقع من المصنف؟ لحل هذه المشكلة تم تدريب "مصنفات متعددة الصفوف multi-label classifier" وكان الغرض منه أن يقوم بالتنبؤ بكلا الصنفين "أي احتواء الصورة على قطة وكلب بنفس الوقت"، ولكننا كنا مازلنا غير قادرين على كشف موضع كل منهما، وهنا ظهرت لدينا مسألة جديدة وهي مسألة الـ Localization، وتهدف بشكل مبسط إلى كشف موضع كل غرض في الصورة، ومن هاتين المسألتين وصلنا إلى مسألة جديدة وهي مسألة **كشف الأشكال** وتهدف هذه المسألة إلى كشف موضع كل غرض في الصورة بالإضافة إلى الصف الذي ينتمي إليه كل غرض.



الشكل 2: مثال عن الخرج المطلوب في مسألة كشف الأشكال.

كما لاحظنا من الشكل السابق المطلوب في مسائل كشف الأشكال إضافة إلى كشف الصف الذي ينتمي إليه الغرض يجب علينا رسم مربع يحيط بهذا الغرض، وسيتطلب هذا الأمر غالباً أربع متحولات لكشف هذا المربع بدقة وهي، لكل ورود لهذا الغرض يجب أن نتنبأ المتحولات الأربع التالية:

1- اسم الصف.

2- إحداثيات الـ x للزاوية العليا اليسرى.

3- إحداثيات الـ y للزاوية العليا اليسرى.

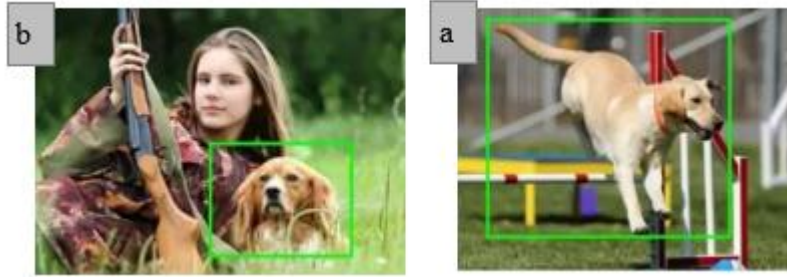
4- عرض المربع المحيط

5- طول المربع المحيط

بعد أن قمنا بتعريف المسألة بشكل دقيق سنتحدث عن المنهجيات المتبعة لحل هذا النوع من المسائل، ويوجد العديد من المنهجيات بداية بـ Haar Cascades التي تم اقتراحها من قبل Viola و Jones في عام 2001، ولكننا سنقوم بالتركيز على الخوارزميات الأحدث التي تعتمد على الشبكات العصبونية والتعلم العميق.

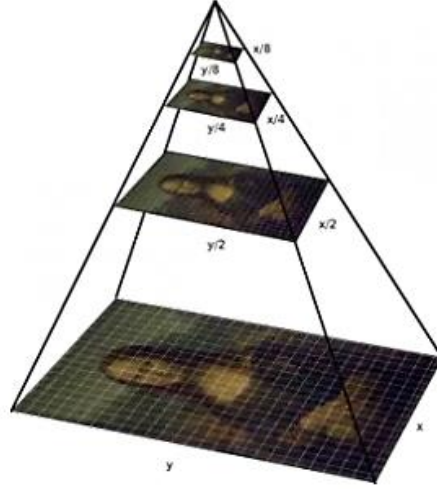
في البداية تم منهجه مسألة كشف الأشكال كمسألة تصنيف "classification problem" بحيث تم أخذ نافذة ثابتة الأبعاد لكل المناطق الممكنة من صورة الدخل، وتم تغذية المصنف بهذه الحزم من الصور، بحيث يقوم المصنف بالتنبؤ بصنف الغرض الموجود فيها، أو يعيد background إذا كانت هذه الصورة لا تحوي أي غرض، وبهذه الطريقة نتمكن من معرفة الغرض الموجود في الصورة وموضعة، ولكن لاحظنا وجود عدة مشاكل في هذا الحل المقترح ألا وهي:

1. كيف يمكننا كشف أبعاد للصورة بحيث تحوي في كل مرة الغرض بغض النظر إذا كان هذا الغرض صغيراً أو كبيراً، وسنوضح هذه المشكلة في الصورة التالية:



الشكل 3: مثال لتوضيح مشكلة ثبات أبعاد نافذة التنبؤ.

ولحل هذه المشكلة تم اقتراح طريقة هرم الصورة، وتعتمد هذه الطريقة على تقييس الصورة باستخدام عدة مقاييس، وثم نعتمد على مبدأ أن نافذتنا ثابتة الأبعاد ستحتوي الغرض المطلوب بشكل كامل في واحدة من هذه الصور المقيسة، ومن الشائع أن يتم تقييس الصورة بمقاييس تصغر هذه الصورة للحصول على الصورة نفسها بعدة مستويات، ويتم تغذية المصنف بكل من الصور الناتجة عن كل المستويات، وبهذه الطريقة يتم حل مشكلة الحجم والموقع، والشكل التالي يوضح الصورة الناتجة عن كل من مستويات التقييس.



الشكل 4: مثال عن طريقة هرم الصورة.

2. المشكلة الثانية هي مشكلة aspect ratio، ويتم توصيف هذه المشكلة في كون العديد من الأشكال يمكن أن تتواجد بوضعيات مختلفة مثلاً شخص جالس واقف أو نائم كلاهما يجب أن يتم تصنيفهما تحت صف انسان. وسيتم الحديث عن حل هذه المشكلة عند الحديث عن الخوارزميات المختلفة.

3.2.2 كشف الأشكال باستخدام Hog Features

وقد تم طرح طريقة Histogram of Oriented Gradients Features أو Hog Features في الورقة البحثية المقدمة من قبل Navneet Dalal and Bill Triggs في عام 2005، وتعد هذه الطريقة غير مكلفة حسابياً بنفس الوقت هي جيدة للتعامل مع العديد من مسائل العالم الحقيقي، بحيث نقوم بحساب خصائص Hog لكل من النوافذ الناتجة عن تمرير النافذة ثابتة الأبعاد على الصور الناتجة من طريقة هرم الصورة، ثم يتم تغذية Support Vector Machine أو SVM بهذه الخصائص لإنشاء المصنفات، وساهمت هذه الطريقة في حل العديد من المسائل ومن أهمها العمل في الوقت الحقيقي (real time) مع فيديو هات الكشف عن المشاة، والكشف الوجوه وغيرها..

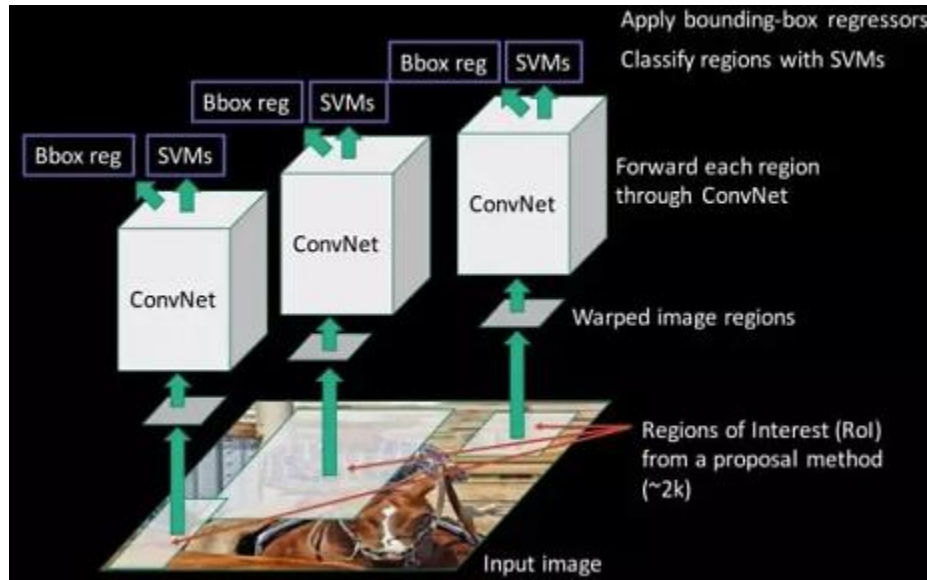
3.2.3 Region-based Convolutional Neural Networks(R-CNN) الشبكات العصبونية التلافيفية المتعلقة بالمنطقة

بما أننا قمنا بنمذجة مسألة كشف الأشكال كمسألة تصنيف، فيعتمد النجاح على دقة المصنّف، بعد تطور تقنيات التعلم العميق، كانت الفكرة البديهية أن نستبدل المصنفات المعتمدة على طريقة خصائص Hog، بمصنّف يعتمد على شبكة عصبونية أكثر دقة، ولكن ظهر لدينا مشكلة عند القيام بهذا التبديل ألا وهي كون الشبكات العصبونية بطيئة، ومكلفة حسابياً، وهذا ما جعل تمرير كل الحزم الناتجة عن النافذة ثابتة الأبعاد على الشبكة العصبونية أمراً مستحيلاً، وقد تم حل هذه المشكلة عن طريق استخدام Object Proposal Algorithms التي من أمثلتها خوارزمية Selective Search أو البحث الانتقائي -

وهي أحد أنواع الخوارزميات المصممة لاستخلاص مناطق الصورة التي يكون احتمال وجود الأشكال فيها هو أكبر ما يمكن- وقد ساهم هذا النوع من الخوارزميات بتقليل عدد المربعات المحيطة التي يتم تغذية المصنف بها إلى حوالي 2000 مربع فقط، وهي المربعات الناتجة عن خوارزميات الـ region proposal، ويستخدم البحث الانتقائي الدلالات الموضعية "المحلية" مثل اللون، القوام، الشدة وغيرها.. لتوليد كل المواقع الممكنة للأشكال، وهذه المربعات الناتجة عن خوارزمية البحث الانتقائي هي ما يستخدم لتغذية الشبكة العصبونية، لتذكر أخيراً أن الطبقات المترابطة بشكل تام في الشبكات العصبونية تأخذ مدخلات ذات حجم ثابت "fixed sized input"، لذا نقوم بتغيير حجم المربعات الناتجة "دون الحفاظ على نسبة العرض على الطول" إلى حجم ثابت وتغذية الشبكة العصبونية بها، لذا يوجد ثلاث أجزاء مهمة من R-CNN، وهي:

- 1- تطبيق خوارزمية بحث انتقائي للحصول على المواقع المحتملة لوجود الأشكال.
- 2- تغذية الشبكة العصبونية CNN، بالمناطق الناتجة عن المرحلة الأولى، ويليها تطبيق SVM للتنبؤ بالصف الذي تنتمي إليه كل منطقة.
- 3- أمثلة المناطق الناتجة عن طريق تدريب المربعات المحيطة الناتجة عن البحث الانتقائي بشكل منفصل باستخدام الانحدار regression.

ويمكن توضيح هذه المراحل بالشكل التالي:

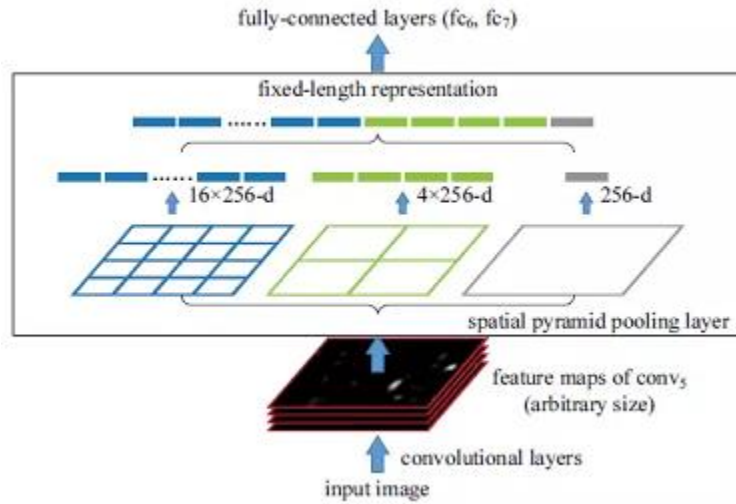


الشكل 5: مراحل عمل خوارزمية R-CNN.

3.2.4 Spatial Pyramid Pooling (SPP-net) تجميع الهرم المكاني

مع كل التعديلات السابقة، كانت خوارزمية R-CNN ماتزال بطيئة للغاية، لأن تشغيل الشبكة العصبونية CNN على كل من ال 2000 منطقة المقترحة من قبل خوارزميات البحث الانتقائي كان لازال يستغرق وقتاً طويلاً، وحاولت هذه الخوارزمية SPP-net تصحيح هذه المشكلة بالشكل التالي:

باستخدام SPP-net، نقوم بحساب تمثيل CNN لكامل الصورة مرة واحدة فقط، ثم باستخدام هذا التمثيل نقوم بحساب تمثيل CNN لكلٍ من المناطق المقترحة من قبل خوارزميات البحث الانتقائي، ويمكن القيام بهذا الأمر عن طريق إجراء نوع من عمليات التجميع على المناطق المقترحة من قبل خوارزميات البحث الانتقائي، وتم تعريف حل لمشكلة كون الطبقات المتصلة بشكل تام تحتاج إلى دخل ثابت الأبعاد، ولكن بالرغم من هذه التحسينات لازال هذه الطريقة تمتلك عدّة نقاط ضعف من أهمها صعوبة إجراء انتشار عكسي "back-propagation" خلال طبقة التجميع المكاني "spatial pooling layer"، فإن الشبكة لم تضبط سوى الجزء المتصل بالكامل من الشبكة. لذا ظهرت طريقة Fast-RCNN.



الشكل 6: شكل توضيحي لمراحل خوارزمية SPP-net

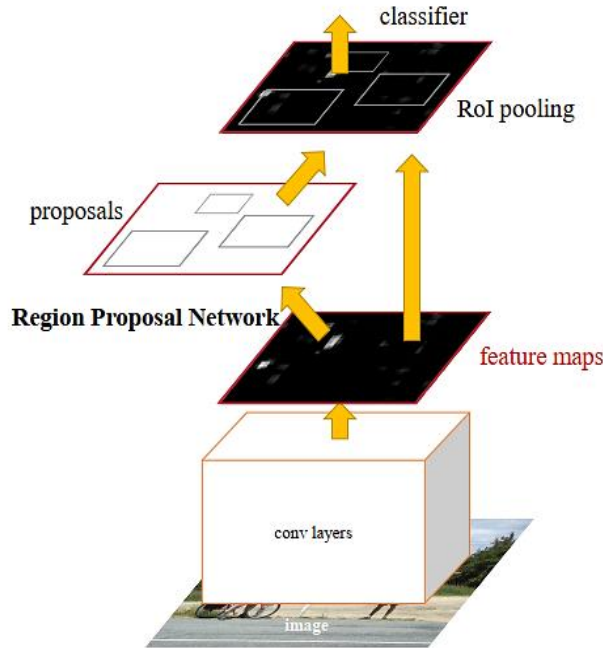
Fast R-CNN 3.2.5

تبنت هذه الطريقة الأفكار المقترحة في SPP-net، و RCNN، كما قامت بإصلاح المشكلة الأساسية التي واجهتها طريقة SPP-net، ألا وهي عدم القدرة على القيام بالتدريب end-to-end، ومعنى كون الشبكة العصبونية تدرب من نهاية إلى نهاية أخرى end-to-end يتم استقبال الدخل من إحدى النهايتين وإعطاء خرج من النهاية الأخرى وبهذه الحالة يتم إزالة كامل الخوارزميات الوسيطة المصنوعة يدوياً "hand-crafted intermediate algorithms" وذلك لأنها تستخدم حساب بسيط للانتشار العكسي الذي يشبه إلى حد كبير حساب التدرج الأقصى للتجميع باستثناء أنه لا يوجد تداخل بين مناطق التجميع وبالتالي يمكن أن تحتوي الخلية على تدرجات تضخ من مناطق متعددة.

وشيء آخر تم إضافته إلى طريقة Fast RCNN هو إضافة انحدار المربع المحيط إلى تدريب الشبكة العصبية نفسها. لذا، أصبح للشبكة الآن رأسان، رأس تصنيف، ورأس انحدار الصندوق المحيط. هذا الهدف متعدد المهام هو ميزة بارزة في Fast-RCNN لأنه لم يعد بحاجة لتدريب الشبكة بشكل مستقل للتصنيف والتوطين "localization". وقد قلل هذان التغيران من وقت التدريب الإجمالي وأديا إلى زيادة في الدقة مقارنة SPP-net.

3.2.6 Faster R-CNN [1] (Faster Region-Based Convolutional Neural Networks)

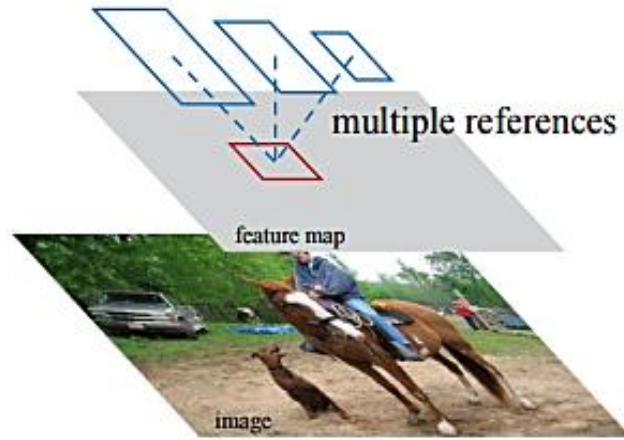
وهي كسابقتها من الخوارزميات التي تُنمذج مسألة كشف الأشكال كمسألة تصنيف "Classification Problem"، تعتمد شبكات كشف الأشكال الحديثة على خوارزميات اقتراح المنطقة "region proposal algorithms" لاقتراح مناطق لوجود الأشكال، لقد أدت التطورات مثل SPP-net [2]، Fast R-CNN [3]، إلى تقليل وقت التشغيل لشبكات كشف الأشكال، مما كشف كون الجزء المتعلق باقتراح المنطقة كالحلقة الأضعف في هذه الخوارزميات، ولحل هذه المشكلة تم تحويل الجزء المتعلق باقتراح المنطقة لتقوم بتنفيذه شبكة تلافيفية "Region Proposal Network (RPN)" تتعامل مع الصور عن طريق الميزات التلافيفية للصورة وكما تم دمج كل من الشبكتين السابقتين RPN و Fast R-CNN في شبكة واحدة بما أنهما يتشاركان بالميزات التلافيفية، بحيث يخبر المكوّن RPN الشبكة الموحدة بمكان البحث. ولُوحظ أن تغيير خطوة اقتراح المناطق لجعل شبكة عصبونية تلافيفية عميقة هي المسؤولة عنها قدّم لنا حلاً أكثر فعالية ودقة ويتناهى لكونه خال من الكلفة.



الشكل 7: تصميم الشبكة في خوارزمية Faster R-CNN.

الفكرة من وراء الخوارزمية:

لوحظ أن خرائط الميزات التلافيفية المستخدمة في المحددات القائمة على أساس المناطق يمكن أن تستخدم لتوليد اقتراحات للمناطق فتم بناء RPN فوق الميزات التلافيفية عن طريق إضافة عدّة طبقات تلافيفية التي تتنبأ "regress" في وقت واحد بحدود الأشكال وتقوم بتقييم مقدار انتماء الأشكال إلى كل من الصفوف في كل منطقة من الصورة وفق شبكة منتظمة، وبالتالي فإن RPN هو نوع من الشبكات التلافيفية التي يمكن تدريبها من البداية إلى النهاية خصيصاً لمهام كشف الأشكال، وقد تم تصميم هذه الشبكات لتقوم بالتنبؤ بكفاءة بمقترحات المناطق على نطاق واسع من المقاييس ونسب العرض إلى الطول "aspect ratios"، فعلى عكس الطرق السابقة كما في [3] و [2] لم يتم استخدام أهرامات الصور بل تم تعريف مفهوم جديد وهو المرساة "anchor" وهي عبارة عن مربعات مرجعية لعدّة مقاييس ونسب عرض إلى طول وهذا الحل يتجنب تعداد الصور والمرشحات ذات المقاييس المتعددة أو نسب العرض إلى الطول المتعددة، وأثبت هذا النموذج كفاءته عند التدريب والاختبار باستخدام صور بمقياس واحد وكما أثبت سرعة في التشغيل، ويمكن فهم طريقة عمله بشكل أفضل بالنظر إلى الشكل أدناه.



الشكل 8: الطريقة المتبعة في Faster R-CNN لمعالجة مشكلة الحجم والمقاييس المتعددة.

وقد كانت هذه الطريقة واحدة من أكثر خوارزميات كشف الأشكال دقةً بالإضافة لكونها أسرع من سابقتها وفيما يلي جدول يبين مقارنة سريعة بينها وبين سابقتها:

	R-CNN	Fast R-CNN	Faster R-CNN
Test Time per Image	50 Seconds	2 Seconds	0.2 Seconds
Speed Up	1x	25x	250x

الجدول 1: مقارنة بين خوارزميات R-CNN, Fast R-CNN, Faster R-CNN من حيث السرعة.

3.2.7 خوارزمية النظر لمرة واحدة "YOLO [4]"

إن خوارزمية Yolo هي واحدة من الخوارزميات الحديثة في كشف الأشكال، ويختلف مبدؤها عن سابقتها في كونها تعتبر مسألة كشف الأشكال من المسائل من نوع "Regression problems" لمربعات مكانية منفصلة عن بعضها البعض مع كشف احتمالات لكل من هذه المربعات موافقة لكل صف من الصفوف الموجودة.

تتنبأ شبكة عصبونية واحدة بمربعات محيطة، واحتمالات لكل صف من الصفوف الموجودة بشكل مباشر من الصورة الكلية في تقييم واحد، وبما أن كامل سلسلة التنبؤ هو عبارة عن شبكة واحدة يمكننا أن نقوم بإجراءات لجعل الشبكة أكثر مثالية بالشكل طرف إلى طرف أثناء عملية التعرف. وقد ظهرت عدة نسخ لهذه الخوارزمية ابتداءً من YOLOv1 وانتهاءً بأحدث إصدار YOLOv5.

YOLOv1 [4]

وفي هذه النسخة كانت الخوارزمية تقوم بتقسيم الصورة إلى شبكة بأبعاد $S \times S$ ، ومن مثالها الشكل الموضح أدناه. إذا وقع مركز الشكل ضمن أحد خلايا الشبكة، فتكون الخلية مسؤولة عن الكشف عن هذا الشكل.



الشكل 9: صورة مقسمة إلى شبكة بأبعاد 5x5.

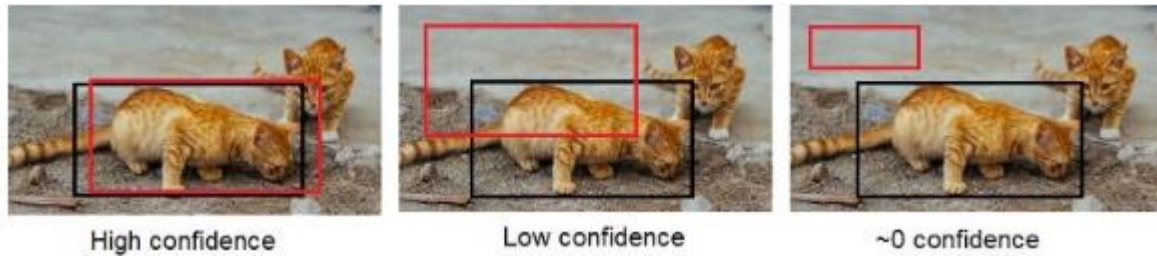
وتعمل الخوارزمية على إجراء عملية "التصنيف وتحديد الموقع" في كل خلية من خلايا الشبكة السابقة على التوازي. وهذا ما يجعل هذه الخوارزمية سريعة. وبما أن الشبكات المصممة لحل مسائل "التصنيف وتحديد الموقع" لا يمكنها الكشف إلا عن

شكل واحد فهذا يعني أن كل خلية يمكنها الكشف عن شكل واحد فقط، ولهذا تواجه خوارزمية YOLO العديد من المشاكل ألا وهي:

- 1- إذا استخدمنا شبكة بأبعاد 7×7 مثلاً، يصبح العدد الأعظمي للأشكال التي يمكن الكشف عنها هو 49.
- 2- إذا احتوت الخلية أكثر من شكل فلا يمكن للنموذج أن يتنبأ بكل هذه الأشكال وتسمى هذه المشكلة بـ "Close object detection".
- 3- يمكن للشكل أن يتوضع على عدّة خلايا مثل السيارة الموضحة في الشكل السابق، فيقوم النموذج في هذه الحالة بالتنبؤ بالشكل أكثر من مرة وقد تم حلّ هذه المشكلة باستخدام "non-max suppression" وستكلم عن هذا المفهوم لاحقاً.

تقوم كل من خلايا الشبكة بالتنبؤ بـ B من المربعات المحيطة (في الخوارزمية غالباً $B=2$)، ولكل مربع من هذه المربعات يحسب النموذج عامل للثقة "c" confidence score، يعكس هذا العامل مدى تأكيد النموذج من كون المربع المحيط حاوٍ على الشكل، ويفيد هذا العامل في منع النموذج من تنبؤ الخلفيات، أي إن لم تحتوي الخلية على أي شكل فسيكون معامل الثقة مساوٍ للصفر، وإلا نريد لهذا المعامل أن يساوي IOU بين المربع المحيط والمربع الحقيقي.

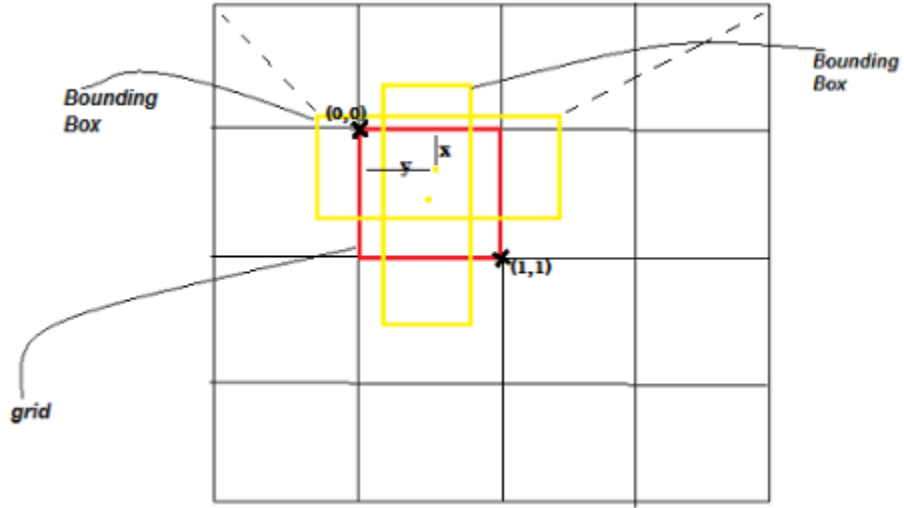
ويجب أن يكون $c=IOU$ لأن المربع الحقيقي يكون مرسوماً باليد لذا فنحن متأكدين 100% من وجود الشكل ضمنه، وبناءً على هذا أي مربع له نسبة IOU عالية مع مربع الحقيقة سيحيط بالشكل المراد التنبؤ بوجوده، أي كلما ازداد احتمال وجود الشكل ضمن المربع ستكون قيمة IOU لهذا المربع مع مربع الحقيقة عالية.



الشكل 10: العلاقة بين معامل الثقة و IOU، المربعات السوداء هي مربعات الحقيقة والخمراء هي المربعات التي تم التنبؤ بها.

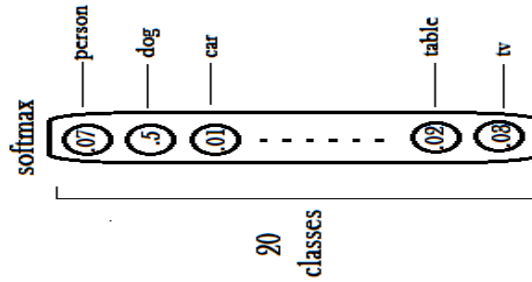
بالرغم من وجود 49 خلية في الشبكة، وكل من هذه الخلايا يمكنها أن تتنبأ بمربعين محيطين أي 98 مربع محيط بالمجمل، ولكن أغلب هذه المربعات توافقها معاملات ثقة صغيرة low confidence، لذا نستطيع أن نتخلص منها. ويتكون خرج الخوارزمية بالإضافة إلى معامل الثقة من أربع أرقام كأغلب خوارزميات الكشف عن الأشكال فيصبح الخرج بالإضافة إلى معامل الثقة بالشكل: (x, y, w, h) وتفيد هذه التنبؤات في تحديد موضع وابعاد المربع المحيط.

تعبّر (x, y) عن إحداثيات مركز المربع بالنسبة لأبعاد خلية الشبكة. ويفيد كل من (w, h) في تحديد العرض والارتفاع بالنسبة للصورة الكلية، لذا $0 < (x, y, w, h) < 1$.



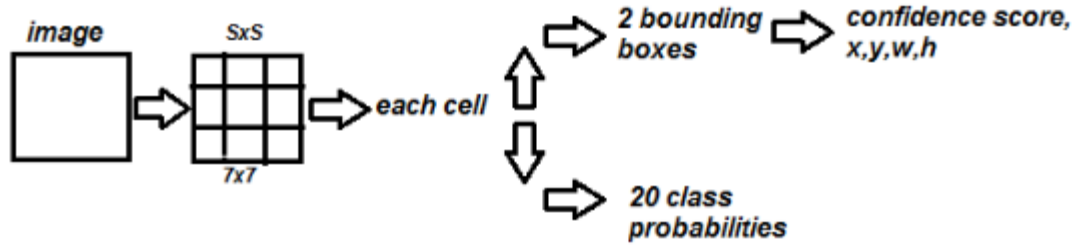
الشكل 12: متحولات خرج النموذج بالنسبة للشبكة.

تم تدريب هذه النسخة من الخوارزمية للتنبؤ بـ 20 صف مختلف، لذا سيعطي النموذج 20 احتمال شرطي واحد لكل من الصفوف.



الشكل 11: الاحتمالات الشرطية الناتجة عن النموذج الموافقة لكل صف من الصفوف الذي تم تدريب النموذج

أي كل خلية من خلايا الشبكة ستعطينا مربعين محيطين محتملين، ولدينا شعاع واحد للاحتمالات الشرطية الموافقة لكل صف من الصفوف التي تم تدريب النموذج عليها، وأخيراً يجب أن نقوم بالتخلص من المربعات المحيطة الموافقة لعوامل ثقة صغيرة.

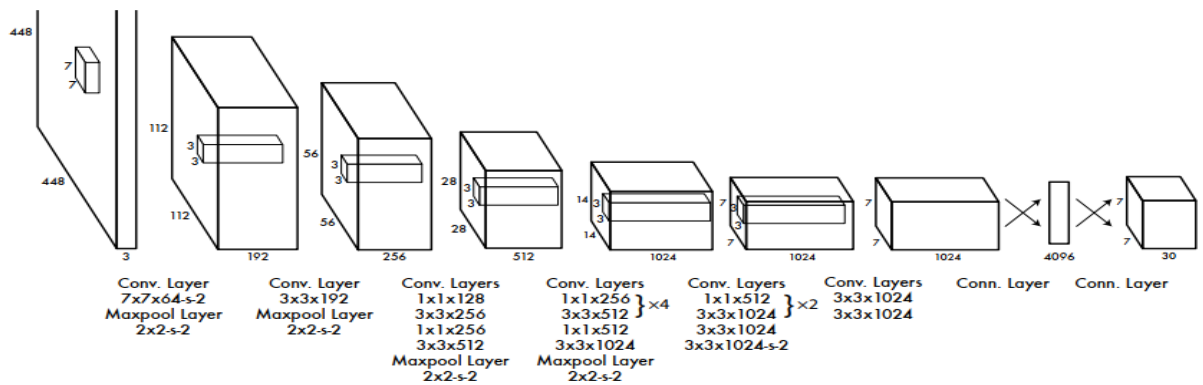


الشكل 13: مراحل خوارزمية YOLO، والخروج النهائي.

يتم ترميز التوقعات كـ tensor $s \times s \times (B * 5 + \text{Classes})$ ، بفرض $S=7$ ، $B=2$ ، $\text{Classes}=20$ سينتج لدينا تينسر بالأبعاد $7 \times 7 \times 30$.

تصميم الشبكة:

تستخدم YOLO شبكة تلافيفية وحيدة للتنبؤ على التوازي بكل من المربعات المحيطة، واحتمالات الصفوف لهذه المربعات، وقد تم بناء هذه الشبكة العصبونية بشكل مشابه لشبكة GoogleNet، المستخدمة لتصنيف الصور، ولكن عوضاً عن ال inception model المستخدم في GoogleNet، تستخدم YOLO طبقة اختصار reduction layer متبوعة بـ 3×3 طبقات تلافيفية. وتحتوي YOLO على 24 طبقة تلافيفية، متبوعة بطبقتين مرتبطتين بشكل كامل. وكما ذكرنا سابقاً فإن خرج هذه الشبكة هو تينسر من التوقعات بالأبعاد $7 \times 7 \times 30$.



الشكل 14: بنية الشبكة YOLO

تابع الخطأ: Loss Function

تستخدم YOLO تابع مجموع مربعات الأخطاء "Sum Squared Error" "SSE"، وذلك لأنه من التوابع التي تعدّ سهلة الأمثلة، ويعطى هذا التابع في نموذجنا بالعلاقة:

$$\begin{aligned} = & \lambda_{\text{coord}} \sum_{i=0}^{s^2} \sum_{j=0}^B \mathbb{I}_{ij}^{\text{obj}} [(x_i - \hat{x}_1)^2 + (y_i - \hat{y}_1)^2] + \lambda_{\text{coord}} \sum_{i=0}^{s^2} \sum_{j=0}^B \mathbb{I}_{ij}^{\text{obj}} \left[(\sqrt{w_i} - \sqrt{\hat{w}_1})^2 + \right. \\ & \left. (\sqrt{h_i} - \sqrt{\hat{h}_1})^2 \right] + \sum_{i=0}^{s^2} \sum_{j=0}^B \mathbb{I}_{ij}^{\text{obj}} (C_i - \hat{C}_1)^2 + \lambda_{\text{noobj}} \sum_{i=0}^{s^2} \sum_{j=0}^B \mathbb{I}_{ij}^{\text{noobj}} (C_i - \hat{C}_1)^2 + \\ & \sum_{i=0}^{s^2} \mathbb{I}_{ij}^{\text{obj}} \sum_{c \in \text{classes}}^B (p_i(c) - \hat{p}_1(c))^2 \end{aligned}$$

يعبر أول حدين عن الخطأ في نتيجة تحديد الموقع، في حين يعبر الحدين الثالث والرابع عن الخطأ في معامل الثقة، في حين يعبر الحد الأخير عن الخطأ في عملية التصنيف، وعند التعامل مع هذه العلاقة، وقبل أن نتعمق في شرح كل حدّ من هذه الحدود، يوجد عدّة نقاط يجب أن يتم الانتباه إليها:

1. يعدّل تابع الخطأ الحد المتعلّق بالتصنيف فقط عند وجود شكل في خلية الشبكة.
2. بما أن $B=2$ أي تتنبأ الخلايا مربعين محيطين لكل خلية من خلايا الشبكة، يجب أن نختار أحد هذين المربعين ليتم معالجته في تابع الخطأ، ويجب اختيار المربع الذي يحقق نسبة ال IOU مع مربع الحقيقة الأعلى، أي لن يقوم تابع الخطأ بالتعديل على الحد المتعلّق بتحديد موقع الشكل إلا عند معالجة المربع المسؤول عن مربع الحقيقة.
3. يسند في علاقة تابع الخطأ SSE نفس الأوزان للحد المسؤول عن الخطأ في تحديد الموقع، وللحد المسؤول عن الخطأ في التصنيف، وهذا ليس لأفضل توزيع.

الحد الأول:

$$\lambda_{\text{coord}} \sum_{i=0}^{s^2} \sum_{j=0}^B \mathbb{I}_{ij}^{\text{obj}} [(x_i - \hat{x}_1)^2 + (y_i - \hat{y}_1)^2]$$

يعبر هذا الحد عن الخطأ بين إحداثيات المربع المحيط الذي تم التنبؤ به، وإحداثيات المربع الحقيقة، ونقوم بإجراء عملية الجمع على كل خلايا الشبكة في الصورة، ولأجل كل خلية يجب أن نقوم بالجمع على كلا المربعين المحيطين لأن $B=2$.

ولتحقيق النقطتين 1، 2 مما سبق يوجد في هذا الحد التابع $\mathbb{I}_{ij}^{\text{obj}}$ وهذا التابع يأخذ القيمة 1 إذا ظهر الشكل في الخلية i وكان المربع j الموافق لهذه الخلية هو المربع المسؤول عن ظهور الشكل، وإلا فإن القيمة هي: $\mathbb{I}_{ij}^{\text{obj}} = 0$.

ونعلم أن المربع يكون مسؤولاً عن تحديد الغرض فقط عندما يمتلك قيمة لـ IOU مع مربع الحقيقة هي الأعلى بين القيم الموافقة لمربعي الخلية.

وبما أن كون الوزن الموافق للحد المسؤول عن الخطأ في تحديد موقع الشكل، مساوٍ للوزن الموافق للحد المسؤول عن تحديد الخطأ في عملية التصنيف، ليس فكرة جيدة فقد تم إضافة ثابت وهو λ_{coord} لإعطاء الخطأ في تحديد الموقع وزناً أكبر في تابع الخطأ ($\lambda_{\text{coord}} = 5$).

الحد الثاني:

$$\lambda_{\text{coord}} \sum_{i=0}^{s^2} \sum_{j=0}^B \mathbb{I}_{ij}^{\text{obj}} \left[(\sqrt{w_i} - \sqrt{\widehat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\widehat{h}_i})^2 \right]$$

مشابه للحد الأول، ولكن هذا الحد مسؤول عن حساب الخطأ في أبعاد المربع المحيط، ويتم في هذا الحد استخدام الجذر التربيعي لأن الأشكال التي نسعى لتحديدتها ستكون بأبعاد مختلفة بعضها كبير، وأخرها صغير ونريد أن نعبّر عن كون الخطأ الصغير في المربعات الكبيرة أقل أهمية منه في المربعات الصغيرة، فمثلاً عوضاً عن التنبؤ بالطول أول العرض لمربع كبير بالقيمة 0.9، و 0.09 لمربع صغير سنتنبأ بـ 0.948 و 0.3 للمربعين السابقين على الترتيب.

الحد الثالث:

$$\sum_{i=0}^{s^2} \sum_{j=0}^B \mathbb{I}_{ij}^{\text{noobj}} (C_i - \widehat{C}_i)^2$$

ويعبر هذا الحد عن الخطأ في معامل الثقة بحيث $\widehat{C}_i = 1$ ، و $0 \leq C \leq 1$.

الحد الرابع:

$$\lambda_{\text{noobj}} \sum_{i=0}^{s^2} \sum_{j=0}^B \mathbb{I}_{ij}^{\text{noobj}} (C_i - \widehat{C}_i)^2$$

إن لم يوجد أي عنصر في الشبكة لن نهتم لكل من الخطأ في عملية التصنيف وفي عملية تحديد الموقع، بل سنهتم فقط بمعامل الثقة C (ويجب أن يكون معامل الثقة مساوٍ للصفر عندما لا نجد أي شكل، ولذا سنستخدم المتحول التالي: $\mathbb{I}_{ij}^{\text{noobj}}$ الذي يساوي الواحد إن لم يوجد أي شكل في الخلية i ، أو إن وجد شكل في الخلية i ، ولكن المربع j هو ليس المربع المسؤول عن هذه الشكل، وفيتم ماعدا ذلك يساوي هذا المتحول للصفر.

نظراً لأن الكثير من خلايا الشبكة لا تحوي أي شكل، فيصبح معامل الثقة لهذه الخلايا مساوٍ للصفر وهي القيمة الموافقة لمعامل الثقة لمربع الحقيقة، ويجعل هذا الأمر عملية التدريب تتقارب بسرعة كبيرة، ولإصلاح هذا الأمر نقوم بتوزيع الحد المسؤول عن الخطأ في معامل الثقة المتنبئ بثابت لتصغير قيمة هذا الحد وهذا الثابت هو $\lambda_{noobj} = 0.5$.

الحد الخامس والأخير:

$$\sum_{i=0}^{s^2} \mathbb{I}_{ij}^{obj} \sum_{c \in \text{classes}}^B (p_i(c) - \hat{p}_i(c))^2$$

وفي هذا الحد نقوم بجمع الأخطاء لاحتمالات كل الصفوف لكل من خلايا الشبكة.

Non-maximal suppression:

- 1- ويعتمد هذا المفهوم الأخير على التخلص من المربعات ذات معامل الثقة الأقل من عتبة محددة
- 2- وترتيب التنبؤات بدءاً من التنبؤات ذات معامل الثقة الأعلى
- 3- اختيار المربعات ذات معامل الثقة الأعلى وترشيحها لتكون تنبؤات الخرج،
- 4- التخلص من كل المربعات التي تكون قيمة IOU الموافقة لها مع مربع الحقيقة أصغر من عتبة معينة طبعاً ويتم تطبيق هذه الغربة فقط على المربعات الناتجة من الخطوة السابقة،
- 5- تكرار الخطوات بدءاً من الخطوة الثالثة، حتى نصل إلى كون كل التنبؤات الباقية صحيحة. وتضيف هذه الطريقة المقياس mAP النسبة 2-3%.

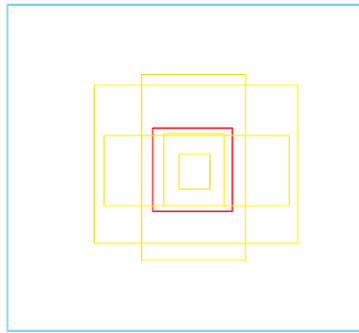
ويوجد لهذه النسخة عدّة حدود "limitations" ألا وهي:

- 1- بما أن كل خلية من الخلايا الشبكية يمكنها أن تتنبأ بمربعين محيطين فقط، فهذا يحد من عدد الأشكال المتقاربة التي يمكن للشبكة التنبؤ بها، فلا يمكننا التنبؤ بالأشكال الصغيرة التي تظهر في مجموعات.
- 2- لا تستطيع YOLO أن تتنبأ بأكثر من $S \times S$ عنصر أي في حالتنا 49 شكل.
- 3- الخطأ الناجم عن تحديد موقع الشكل يعدّ مرتفعاً نسبياً.

YOLOv2 [5]

حوت النسخة الأولى من YOLO بالعديد من الأخطاء في عملية تحديد موقع الشكل المتنبئ به، وكان لها نسبة recall صغيرة، فجاءت النسخة الثانية لتصالح هذين المشكلتين، مع الحفاظ على دقة التصنيف accuracy، ولتحقيق هذه الأهداف تم اعتماد عدّة أفكار:

- Batch Normalization: تم تطبيق هذه الخوارزمية على كل الطبقات التلافيفية في شبكة YOLO، وقد أدت إضافة هذه الخطوة إلى تحسين أكثر ال mPA بـ 2% .
- High Resolution Classifier: تم تدريب النسخة الأولى من خوارزمية YOLO عن طريق تدريب شبكة التصنيف على الأبعاد 224×224 ، ومن ثم رفع هذه الأبعاد إلى 448 للقيام بعملية الكشف. وهذا يعني أنه عند التبديل للقيام بالكشف يجب على الشبكة أن تتغير على التسلسل للتعامل مع مسألة كشف عن الأشكال، وأن تقوم بتعديل الأبعاد. في حين في YOLOv2 تم في البداية تدريب النموذج للتعامل مع صور بأبعاد 224×224 ، ثم ضبط "fine tune" شبكة التصنيف للأبعاد الكاملة 448×448 ، لـ 10 epochs على ImageNet قبل البدء بالتدريب للكشف عن الأشكال، وهذا يمنح الشبكة وقتاً كافياً لتعديل مرشحاتها (filters) للعمل بشكل أفضل على الصور ذات الأبعاد الأكبر أو الأكثر دقة، وقد ساهم التصنيف على هذه الأبعاد العالية في زيادة نسبة mPA إلى بنسبة 4%.
- استخدام anchor boxes أو "prior box" لإعطاء كل خلية من خلايا الشبكة القدرة على التنبؤ بعدة أشكال، ويمكن تعريف ال anchor box بأنه طول وعرض، يتم التنبؤ بالمربعات المحيطة منسوبة لهذين الطول والعرض، عوضاً عن التنبؤ بالمربعات منسوبين لكامل الصورة. واستخدام هذه الطريقة سيجعل عملية التعلم أسهل للشبكة العصبونية، فمثلاً في الصورة التالية يوجد لدينا خلية الشبكة المرسومة باللون الأحمر "اللون الغامق"، ونجد 5 مربعات محيطة بها وهي ال anchor boxes باللون الأصفر "اللون الفاتح"، فتقوم YOLOv2 بإيجاد ال anchor box ذو الشكل الأمثل مما يجعل تعلم الكشف أسهل بالنسبة للشبكة.



الشكل 15: تمثيل لخلية شبكة محاطة بـ 5 مربعات prior بأشكال مختلفة.

إن التصميم الموحد للشبكة سريع للغاية، ويقوم النموذج الأساسي في خوارزمية YOLO بمعالجة الصور في الوقت الحقيقي بمعدل 45 إطار في الثانية الواحدة، كما يوجد تصميم آخر وهو "tiny Yolo" يقوم هذا التصميم بمعالجة 155 إطار في الثانية الواحدة، وبنفس الوقت يحقق ضعف ال mAP "mean Average Precision" مقارنة بباقي المحددات التي تعمل بالزمن الحقيقي، ويجدر بنا أن نعلم أن Yolo تعدّ أفضل من باقي المحددات الحديثة فمعدل الأخطاء

التي من نمط false positive أقل ولكن بنفس الوقت تقوم هذه الخوارزمية بارتكاب أخطاء عندما يتعلق الأمر بكشف الموقع الغرض "Localization"، وأخيراً، تتعلم YOLO على تمثيل عام للأشكال، وتتجاوز بذلك طرق كشف الأشكال الأخرى مثل DPM و R-CNN، عند التعميم من الصور الطبيعية إلى الصور المتخصصة بمجال محدد كالصور الفنية.

وظهر بعد هذه النسخة YOLO9000، ثم تلتها النسخة YOLOV3، وهي إحدى النسخ التي استخدمناها في مشروعنا.

YOLOV3 [6]

وهي أكبر من النسخ السابقة، لكنها تمتاز بكونها أكثر دقة، وتختلف عن سابقتها بـ:

- للتعويض بالمربعات المحيطة تقوم هذه النسخة بالتنبؤ بـ 4 أبعاد لكل مربع محيط، وهذه الأبعاد هي tx, ty, tw, th، لذا إذا كانت الخلية تبعد عن الزاوية العليا للصورة بمقدار (cx,cy)، ويبلغ طول وعرض الـ prior box حولي Pw, Ph، فتكون التوقعات متوافقة مع المعادلات التالية:

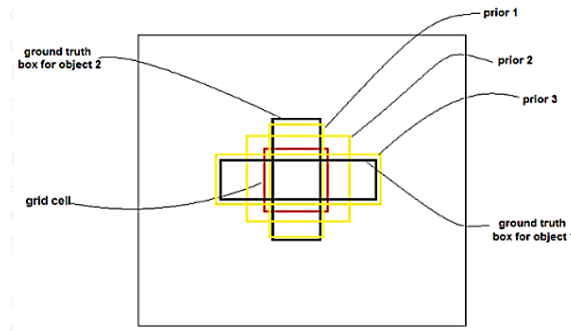
$$b_x = \sigma(t_x) + c_x$$

$$b_y = \sigma(t_y) + c_y$$

$$b_w = p_w e^{t_w}$$

$$b_h = p_h e^{t_h}$$

كما يأخذ معامل الثقة في هذه النسخة القيمة 1 عندما يكون المربع prior المحيط متقاطعاً مع المربع الحقيقي بمساحة أكبر من مساحة تقاطع الشكل مع أي مربع prior آخر محيط، أي في النهاية يقوم النظام بإسناد مربع prior محيط واحد إلى كل غرض حقيقي "ground truth object" وسيهمل باقي المربعات الـ prior.

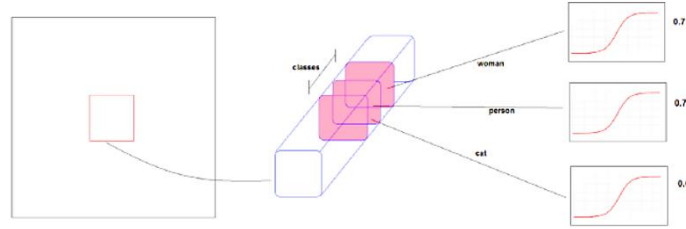


الشكل 16: في هذا الشكل يتقاطع الـ prior 1 مع الـ ground truth بأكثر

مساحة لذا فهذا المربع هو الذي سيسند لهذا الغرض.

- التنبؤ بعدة صفوف لنفس الغرض، وذلك لأن بعض الـ datasets قد تسند أكثر من صف واحد للشكل، فمثلاً صورة لامرأة قد تصنف كـ person، woman، وقد تم تحقيق هذا الأمر عن طريق استبدال تابع softmax المستخدم

في النسخ السابقة، وعوضاً عنه يتم استخدام مصنفات لوجستية مستقلة independent logistic classifiers لكل صف.



الشكل 17: استخدام مصنفات لوجستية مستقلة لكل صف من الصفوف، لغرض يمكن أن يصنف كـ **person**، أو كـ **woman**

- بينما تواجه النسخ القديمة من Yolo صعوبات في التعرف على الأشكال الصغير، نلاحظ أن النسخة 3 من Yolo، حققت أداءً أفضل عند التعامل مع الأشكال الصغيرة، وذلك نظراً لاستخدام short cut connections، وذلك لأن هذه الطريقة تمكننا من استخراج معلومات أكثر دقة من خريطة المعالم "feature map"، ولكن بالمقابل فإن هذه النسخة قد حققت نتائج اسوء عند التعامل مع الأشكال المتوسطة والكبيرة الحجم.
- تستخدم هذه الخوارزمية شبكة جديدة لاستخراج المعالم "feature extraction network" وهي darknet-53، وهي عبارة عن نسخة هجينة من الشبكة المستخدمة في نسخة Yolov2، ألا وهي darknet-19، و residual network [5]، وهذه الشبكة تحتوي على small cut connections، وسميت بـ darknet-53، لأنها تحتوي على 53 طبقة تلافيفية، وتعد هذه الشبكة أسرع من سابقتها.

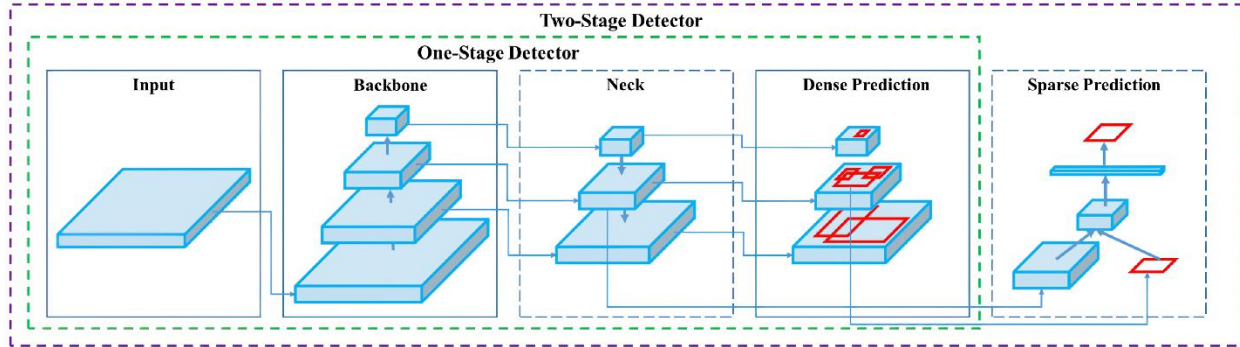
Type	Filters	Size	Output
Convolutional	32	3 × 3	256 × 256
Convolutional	64	3 × 3 / 2	128 × 128
1×	Convolutional	32	1 × 1
	Convolutional	64	3 × 3
	Residual		128 × 128
	Convolutional	128	3 × 3 / 2
2×	Convolutional	64	1 × 1
	Convolutional	128	3 × 3
	Residual		64 × 64
	Convolutional	256	3 × 3 / 2
8×	Convolutional	128	1 × 1
	Convolutional	256	3 × 3
	Residual		32 × 32
	Convolutional	512	3 × 3 / 2
8×	Convolutional	256	1 × 1
	Convolutional	512	3 × 3
	Residual		16 × 16
	Convolutional	1024	3 × 3 / 2
4×	Convolutional	512	1 × 1
	Convolutional	1024	3 × 3
	Residual		8 × 8
Avgpool		Global	
Connected		1000	
Softmax			

الشكل 18: بنية شبكة darknet-53

ولكن وجب التنبيه إلى أنه بالرغم من كل هذه التحسينات إلى أن أداء هذه النسخة لاتزال تواجه صعوبة بتوقع مربعات محيطية حاوية للشكل ككل، ولكنها لاتزال أسرع من الطرق السابقة.

[8] YOLOv4

وقد تم نشر هذه النسخة في شهر نيسان من هذه السنة 2020، لذا فهي لاتزال حديثة، ولكن هذه النسخة لم يتم تطويرها من قبل المطور الأصلي لنسخ YOLO الماضية، بل تم تطويرها من قبل ثلاثة مطورين آخرين، وقد كان الهدف من هذه الخوارزمية هو تصميم نظام كشف للأشكال > امتازات yolo3 بالقوة والسرعة، ولكن yolo4 هي تحديث واضح لها من حيث السرعة والأداء، وفي هذه الفقرة سنتناول طريقة عملها، وكيف تم تطويرها، وماهي المفاهيم التي اعتمدت عليها، ولماذا تم اختيار هذه المفاهيم، مقارنة بينها وبين طرق التعرف على الأشكال المنافسة، ولماذا تعتبر من أفضل الطرق للتعرف على الأشكال.. نعلم أن أنظمة التعرف على الأشكال تتألف من عدة مكونات وهي تبعاً لكون المصنّف مصنّف بمرحلة واحدة أو مصنّف بمرحلتين كما يلي:



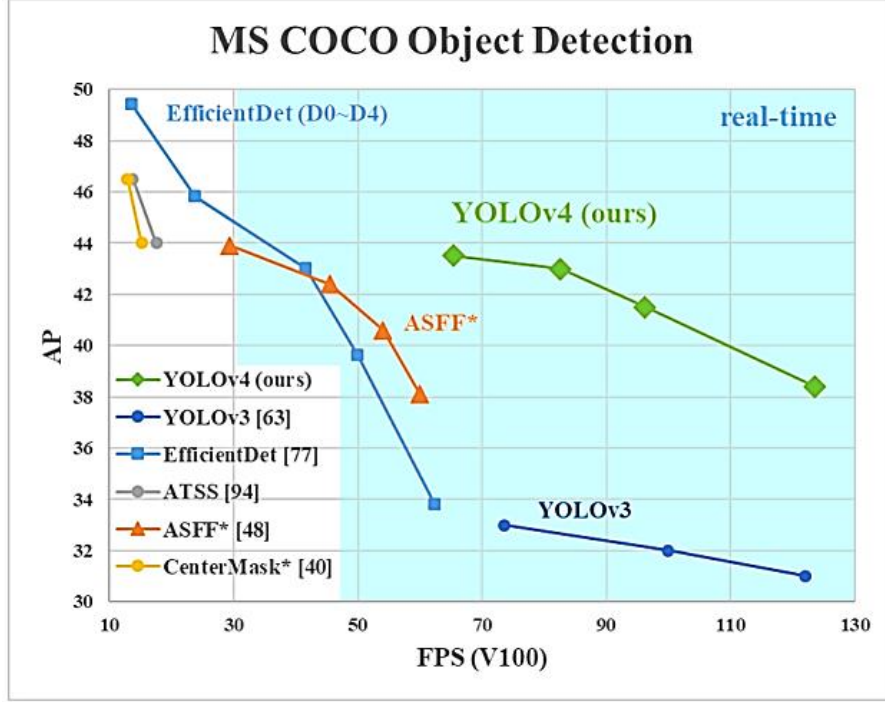
الشكل 19: البنية العامة لأنظمة التعرف على الاشكال.

- The input: وهذا هو المكون الذي ندخل الصور عن طريقه.
 - The backbone: وهي الشبكة التي تأخذ كدخل لها الصورة، وخرجها هو ال feature map، خريطة المعالم ويمكن لهذه الشبكة أن تكون إما: VGG16، RESNET-50، DARKNET52، RESNEXT50.
 - The neck: إن ال neck وال head هي مجموعات جزئية من ال backbone، ويفيد هذا الأمر في تحسين التمييز بين المعالم، وزيادة قوة الخوارزمية عن طريق استخدام FPN، RFB، PAN وغيرها، وال head، وهو الجزء الذي يقوم بالتعامل مع التنبؤات.
- ومن أهم أمثلة أنظمة التعرف على الأشكال بمرحلة واحدة: Yolo، SSD، RPN، في حين يعتبر كل من Faster R-CNN، RFCN من أهم الأمثلة لأنظمة تعرف على الأشكال بمرحلتين.

وتتيح لنا Yolov4 اختيار ال backbone ليكون إما CSPDarknet53، CSPResnext50، أو EffcientNet B3 ، وقد أوضحت الدراسات أن CSPDarknet53 هي النموذج الأمثل، وهو النموذج الذي قمنا باستخدامه في مشروعنا، ومن أحد الأسباب لكون darknet هي أفضل خيار هي وجود ال SPP block وقد سبق أن تكلمنا عنها في الفقرة [2.2.8 Spatial Pyramid Pooling](#) بحيث تمتاز هذه الكتلة بأنها تحسّن القدرة على الفصل بين أهم ال context features بشكل ملحوظ، ولا تتسبب في تقليل سرعة الشبكة. كما قام المطورين باستخدام PANet المعروفة ب Path Aggregation Network كطريقة متكاملة المتحولات من المستويات المختلفة لل backbone، للمستويات المختلفة من الكاشف “detector” عوضاً عن FPN التي تم استخدامها في yolov3، وأخيراً قام المطورين باستخدام yolov3 ك head لهذه النسخة من yolo. وهذه هي البنية العامة ل yolov4، ويمكن القراءة عنهم أكثر في الورقة البحثية المنشورة عن هذه النسخة من yolo،

وهناك العديد من الميزات الإضافية ل yolov4 بالإضافة إلى بنيتها، ومن أهم هذه الميزات، بحيث قام المطورين بعدة خطوات لأتملة عملية التدريب، لزيادة دقة هذه الخوارزمية دون زيادة الكلفة، وقد أطلق المطورين على هذه الطريقة الاسم “bag of freebies” ، ويمكن تعريفها بمجموعة من الطرق التي تغير استراتيجية التدريب فقط وتزيد من كلفة التدريب فقط، وتؤدي إلى الحصول على دقة أعلى دون زيادة تكلفة العتاد، أي أننا نحصل على أداء أفضل بشكل مجاني، وهناك العديد من الاستراتيجيات ضمن هذه الحقيبة، كما قام المطورين بتعريف المصطلح “bag of specials”، وقد عرّف المطورين special بالحصول على شيء مقابل قيمة مخفضة أو بسعر قليل، وبالتالي هي مجموعة من النماذج “models” التي تقوم بزيادة الكلفة بمقدار قليل، ولكنها تساهم في تحسين ملحوظ في دقة عملية الكشف عن الأشكال. كما عرّف المطورين عدة تحسينات إضافية، لتصميم نظام يمكن أن يتم تدريبه على GPU واحد..

وبعد التعريف بجزء من التحسينات التي ميزت هذه النسخة من yolo عن النسخ السابقة، لنقم بعرض النتائج التي حققتها هذه الخوارزمية:



الشكل 20: مقارنة بين أداء **yolov4** مقارنة النماذج الحديثة للكشف عن الأغراض، يمكن بسهولة ملاحظة كون **yolov4** قد تفوقت عن النسخة السابقة بحيث تحسن الـ **AP** و الـ **FPS** بمقدار 10%، و 12% على الترتيب.

4. الفصل الثالث

بيئة التطوير والأدوات البرمجية المستخدمة

يقدم هذا الفصل شرحاً عن البيئة والأدوات المستخدمة في تنجيز العمل، بالإضافة إلى أهم المكتبات والتوابع المستخدمة ضمنها.

4.1. مقدمة

لقد استخدمنا في هذا المشروع لغتي برمجة في بداية العمل استخدمنا لغة Python للتعامل مع منصة colab المقدمة من google، ثم انتقلنا لاستخدام لغة C# من أجل بناء التطبيق النهائي، وفي هذا الفصل سنذكر لغات البرمجة والبيئات المختلفة التي استخدمناها بالإضافة إلى أننا سنذكر المكتبات المختلفة التي استخدمناها.

4.2. لغة البرمجة Python

هي لغة برمجة من اللغات عالية المستوى، تتميز هذه اللغة ببساطتها، وكونها سهلة التعلم، وهي لغة مفتوحة المصدر وقابلة للتطوير، وتعتبر أيضاً لغة تفسيرية متعددة الأشكال وتستخدم بشكل واسع في العديد من المجالات كمحال تحليل المعطيات والذكاء الصناعي. ولقد لجأنا إلى هذه اللغة لوجود العديد من المكتبات المفيدة التي تدعم التعلم بالتعزيز وسنأتي على ذكر هذه المكتبات في الفقرات التالية. ولتنزيل هذه اللغة كل ما علينا هو زيادة موقع اللغة الرسمي الذي يحوي كل المعلومات المطلوبة.



الشكل 21 شعار لغة Python

4.3. لغة البرمجة C#

وهي لغة برمجة متعددة الأشكال، حديثة وقائمة برمجة الأشكال object-oriented، وقد تم تطويرها من قبل شركة Microsoft بقيادة Anders Hejlsberg وفريقه ضمن مبادرة .NET، وتمت الموافقة عليها من قبل جمعية مصنعي الحواسيب الأوروبيين ECMA، ومنظمة المعايير الدولية ISO، وتعد لغة سهلة التعلم لمن لديه معرفة بلغات C أو C++ أو Java. وسنستخدم هذه اللغة في مشروعنا من أجل بناء التطبيق النهائي وتصميم واجهاته.

4.4 Google Colaboratory

يطلق عليها أيضاً اسم Google Colab، وهي خدمة سحابية مجانية تتيح لمستخدميها سواء كانوا طلاب أو data scientist أو AI researcher الوصول إلى GPU's مجانية، وبالتالي الوصول إلى قدرات حاسوبية كبيرة، ويمكنهم colab من تحسين مهاراتهم في لغة البرمجة Python، وتطوير تطبيقات في التعلم العميق باستخدام المكتبات الشائعة مثل Keras, TensorFlow, PyTorch, OpenCv.

كما أن Google colab يعد من أفضل المنصات للتعامل مع ال datasets وذلك لأنه يمكنك رفع قواعد البيانات على ال Google Drive الخاص بك والوصول إليها بهذه الطريقة كما سنوضح فيما يلي:

Mounting Google Drive to My Colab:

```
from google.colab import drive
drive.mount('/content/gdrive')
```

Copy a file such as video or image to my drive:

```
!wget URL
```

وسنجد الرماز البرمجي الناتج عن هذه المرحلة مرفق مع التقرير بالإضافة لإمكانية إيجاده على my drive. حيث سنجد الرماز الموافق لخوارزمية YOLOv3، ولخوارزمية YOLOv4، كما ذكرنا سابقاً ستعامل مع هاتين الخوارزميتين ضمن بيئة darknet، التي سنتحدث عنها لاحقاً.

4.5 Darknet [7]

إن Darknet هي عبارة عن شبكة عصبونية مفتوحة المصدر مكتوبة بلغة C و CUDA، سهلة التنصيب وتدعم كل من المعالجات الرقمية CPU، والمعالجات الصورية GPU. كما تتيح هذه الشبكة العديد من الخدمات منها:

- التعامل بسهولة مع عدّة خوارزميات رائدة في مجال كشف الأشكال ومن أبرزها خوارزمية YOLO.
- تصنيف الصور بسهولة باستخدام نماذج تصنيف مشهورة مثل ResNet و ResNeXt.
- يمكنك أيضاً أن تلعب لعبة GO باستخدام شبكة مدربة مسبقاً عن طريق Darknet.
- وغيرها من الميزات الأخرى، كما يوجد نسخة مصغرة لهذه الشبكة تتيح لنا القيام بتصنيف الصور.



الشكل 22: شعار شبكة Darknet.

وسنستخدم هذه الشبكة لتدريب خوارزميتي yolov3، yolov4 على مجموعات البيانات datasets المتوفرة لدينا، ووجب الملاحظة أننا في مشروعنا هذا سنستخدم نسخة darknet المعدلة من قبل AlexyAB المتاحة على منصة GitHub، وذلك لأن النسخة الأصلية من Darknet التي بناها Joseph Redmon لا تدعم windows.

4.6 Cuda Toolkit

وتوفر NVIDIA CUDA toolkit بيئة تطوير لإنشاء تطبيقات عالية السرعة والأداء بالاستفادة من خواص ال GPU، بحيث عن طريق هذه ال Toolkit، يمكن أن تطوير وتحسين ونشر تطبيقات على الأنظمة المضمنة المسرّعة بواسطة ال GPU، و desktop workstations، enterprise data centers، وأجهزة الحواسيب العملاقة HPC. تتضمن هذه ال toolkit مجموعة من المكتبات المسرّعة بواسطة ال GPU، وأدوات تصحيح الأخطاء "debugging tools"، و مترجم C/C++، ومكتبة runtime لتشغيل التطبيقات.

وفي تطبيقنا هذا قمنا باستخدام cuda Toolkit بالنسخة v10.2 وذلك لأنها نسخة تدعم ال GPU الموجود لدينا كما أنها النسخة الموافقة لعمل المكتبة المستخدمة.

4.7 CuDNN or NVIDIA CUDA Deep Neural Network

وهي عبارة عن مكتبة من المكتبات المسرّعة من خلال ال GPU، وتتألف من عناصر أولية للشبكات العصبية العميقة، وتوفر cuDNN تنجيزات مضبوطة للغاية للإجراءات الأساسية مثل forward and backward convolution, pooling, normalization, and activation layers.

ويعتمد باحثو التعلّم العميق والمطورون في جميع أنحاء العالم على cuDNN لخلق أداء عالي عند التعامل مع وحدات المعالجة الصورية GPU، بحيث تسمح لهم هذه المكتبة بالتركيز على تدريب الشبكة العصبونية وتطوير التطبيقات بدلاً من قضاء الوقت بإجراء ضبط لأداء ال GPU على مستويات منخفضة low level performance tuning، ويستخدم التسريع بـ cuDNN على نطاق واسع مع العديد من تطبيقات التعلّم العميق ومن ضمنها: Caffe2, Chainer, Keras, MATLAB, MxNet, PyTorch, and TensorFlow.

وفي برنامجنا قمنا باستخدام النسخة v7.6.5، وذلك لأنها النسخة المتوافقة مع نسخة CUDA toolkit المستخدمة.

4.8 CMake

وهي عبارة عن مجموعة أدوات مفتوحة المصدر ومتعددة المنصات مصممة لإنشاء الراجم واختبارها وحزمها. يتم استخدام CMake للتحكم بعملية ال compilation التحويل البرمجي للبرامج باستخدام ملفات ضبط configuration files بسيطة ومستقلة عن المنصة وعن المترجم compiler، فتقوم CMake بخلق ملف makefiles ومساحة للعمل workspace يمكن استخدامها في بيئة مترجم من اختيارنا، تم إنشاء مجموعة أدوات CMake بواسطة Kitware استجابةً للحاجة إلى بيئة بناء قوية عابرة للأنظمة cross-platform للمشاريع مفتوحة المصدر مثل ITK و VTK.



الشكل 23: شعار CMake

4.9 OpenCV (Open Source Computer Vision Library)

هي عبارة عن مكتبة برمجية مفتوحة المصدر للرؤية الحاسوبية وتعلّم الآلة، وتم بناء هذه المكتبة لتوفر بنية تحتية لتطبيقات الرؤية الحاسوبية، ولتسريع استخدام تصوّر الآلة machine perception في التطبيقات التجارية ولكونه منتجاً مرخصاً من قبل BSD فإن OpenCV يسهّل على الشركات استخدام وتعديل النصوص البرمجية. تحتوي المكتبة على أكثر من 2500 خوارزمية مُحسّنة، والتي تتضمن مجموعة شاملة من كل من الرؤية الحاسوبية الكلاسيكية والحديثة وخوارزميات التعلم الآلي. يمكن استخدام هذه الخوارزميات لاكتشاف الوجوه والتعرف عليها، وتحديد الأشياء، وتصنيف الأفعال البشرية في مقاطع الفيديو، وتتبع حركات الكاميرا، وتتبع الأجسام المتحركة، واستخراج نماذج ثلاثية الأبعاد للكائنات، وإنتاج سحب نقطية ثلاثية الأبعاد من كاميرات stereo، وتجميع الصور معاً لإنتاج دقة عالية صورة مشهد كامل، والعثور على صور مماثلة من قاعدة بيانات الصور، وإزالة العيون الحمراء من الصور التي تم التقاطها باستخدام الفلاش، ومتابعة حركات العين، والتعرف على المشهد وإنشاء علامات لتراكبها مع الواقع المعزز، وما إلى ذلك. OpenCV لديه أكثر من 47 ألف مستخدم مجتمع والعدد المقدر للتنزيلات يتجاوز 18 مليون. تستخدم المكتبة على نطاق واسع في الشركات والمجموعات البحثية والهيئات الحكومية.



الشكل 24: شعار مكتبة OpenCV

وفي مشروعنا استخدمنا النسخة OpenCV 4.3.0.

4.10 Microsoft Visual Studio

هو بيئة تطوير متكاملة (IDE) من Microsoft، يستخدم لتطوير برامج الحاسب، ومواقع الويب وتطبيقات الويب وخدماتها، وتطبيقات الأجهزة المحمولة، وقد استخدمنا في مشروعنا نسخة Visual Studio 2019 community، يستخدم Visual Studio منصات تطوير برامج Microsoft مثل Windows API، Windows Forms، Windows Presentation Foundation، Windows Store، Windows Silverlight، وسنقوم في هذا المشروع بإنتاج برنامج من النوع WPF أو Windows Presentation Foundation وذلك لأن هذا النمط من البرامج يحمل العديد من الميزات عند مقارنته بـ Windows Forms التقليدية، ومن أهمها:

- تعتبر Microsoft الآن Windows Forms تقنية قديمة ولن تقوم بتحديثها داخل visual Studio.
- تمتاز بأنك تكتب نفس التعليمات البرمجية في C#، ولكن يمكنك تصميم الواجهة بسهولة أكبر بحيث تعتمد على تخطيط الشاشات، ويمكننا التحكم في شكل ومظهر عناصر التحكم.
- يتم تصميم الواجهة باستخدام لغة XAML وهي لغة XML-based language، وهي لغة مشابهة إلى حد كبير طريقة التصميم في ASP.NET.
- الأنماط والسمات الديناميكية، بحيث يمكننا إنشاء جميع الأنماط، وحتى السمات الكاملة لتطبيق WPF، كمورد خارجي. هذا يعني أنه يمكنك تغيير شكل ومظهر التطبيق دون إعادة تجميع أي من التعليمات البرمجية الخاصة بنا، ويعتبر هذا إنجازاً صعباً مع تطبيق Windows Forms التقليدي، وفي حالتنا قمنا بالاستفادة من [8] Material Design In XAML Toolkit.
- سهولة الفصل بين منطق الأعمال وواجهة المستخدم، صممت Microsoft الـ WPF من الألف إلى الياء للحصول على فصل فضفاض loosely couple بين طبقة العرض، وطبقة منطق العمل، كما تسمح هذه البنية باستخدام أنماط التصميم مثل MVC (Model View Controller)، وغيره من الأنماط.
- ربط البيانات، في WPF قدمت Microsoft ربطاً أكثر قوة للبيانات وهناك العديد من الروابط التي تتيح المرونة اللازمة لتطبيقات الأعمال المعقدة. سنستخدم XAML للتعبير عن ربط البيانات الذي كان علينا كتابة التعليمات البرمجية له في تطبيقات Windows Forms. مما يؤدي إلى زيادة الإنتاجية بشكل كبير.
- أبعاد الشاشة، يحتوي WPF على تسهيلات مضمنة للتعامل مع دقة شاشة مختلفة مماثلة للطريقة التي يعمل بها HTML مع دقة الشاشة. [9]

4.11 Alturos.Yolo

بما أن البرنامج النهائي المطلوب هو بلغة C#، فيجب أن نبحث عن تغليف لـ darknet المكتوبة بلغة C لنتمكن من التعامل معها في مشاريع C#، وبعد البحث وجدنا الحزمة Alturos.Yolo وهي عبارة عن واحد من أحدث أنظمة الكشف عن

الاشكال في الوقت الحقيقي real-time، للـ Visual Studio C#، وكما تتضمن هذه الحزمة دعماً للتعامل مع كل من CPU، GPU لكون التعامل مع الـ GPU أسرع بكثير من التعامل مع الـ CPU، والهدف الأساسي من هذه الحزمة هو تسهيل استخدام Yolo، ووجب التنويه إلى كون هذه الحزمة متاحة على nuget، وتستخدم هذه الحزمة كخلفية لها نسخة Yolo لحوايب الـ windows التي قام بإنشائها AlexyAB/Darknet، وتدعم هذه المكتبة النسخ YoloV3، YoloV2 المدربة مسبقاً على مجموعات بيانات، وسنستخدم هذه المكتبة للتعامل مع YoloV4.



الشكل 25: شعار حزمة Alturos.Yolo

4.12 OpenCvSharp3-AnyCPU

وهي عبارة عن مكتبة لتغليف مكتبة OpenCv لتسهيل التعامل معها في المشاريع البرمجية المكتوبة بلغة C#.

5. الفصل الرابع

التنفيذ العملي للنظام

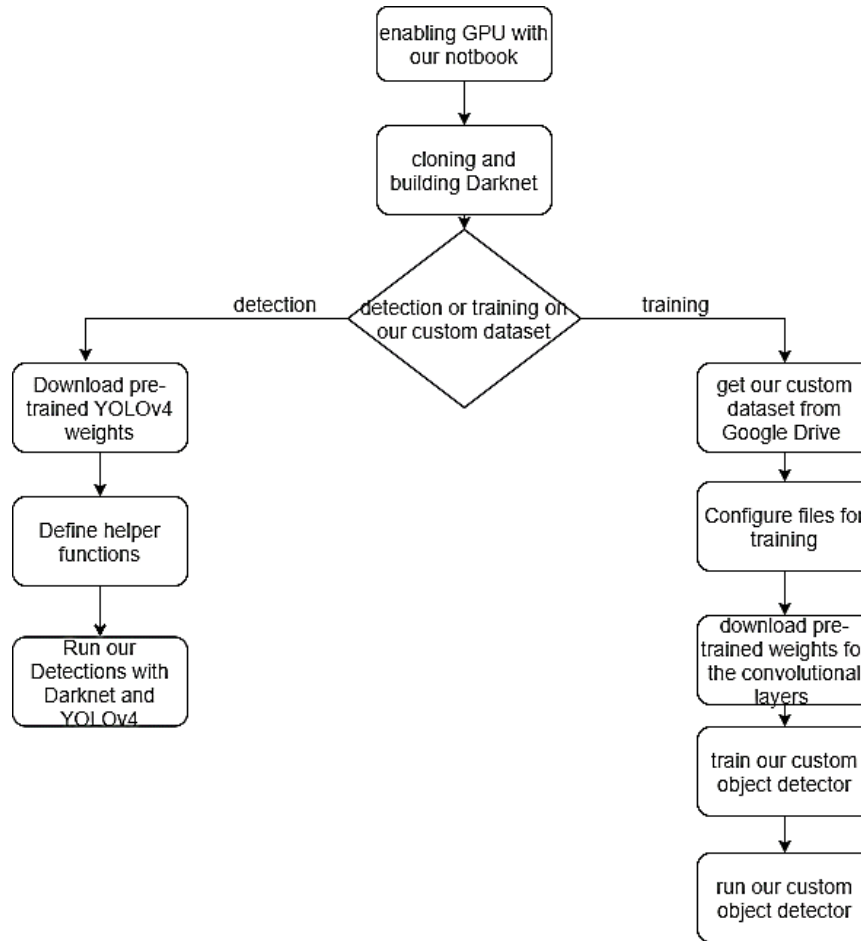
يقدم هذا الفصل التصميم المعتمد للوحدات الأساسية التي تقدّم الوظائف المطلوبة، ثم نعرض الواجهة البريانية للتطبيق وكيفية التعامل معها.

5.1. مقدمة

سنقوم في هذا الفصل بشرح الكود البرمجي الذي قمنا بكتابته لحل المسألة المطروحة، وكما ذكرنا في الفصل السابق فقد قمنا بتنفيذ النص البرمجي أولاً على منصة colab باستخدام لغة البرمجة python، وكما استفدت من كون هذه المنصة تتيح للمستخدم استخدام GPUs موفرة من قبل google لتدريب نماذجنا our models، وكما تتيح للمستخدم ربط الكود البرمجي مع google drive لتسهيل تحميل الأوزان ومجموعة البيانات dataset على منصة colab لاستخدامها في الكود البرمجي، ثم انتقلنا لاستخدام visual studio، لأنه طلب مني تنفيذ البرنامج النهائي بلغة C#، لأسباب فرضها الدكتور المشرف.

5.2. الطريقة الأولى

وهي الطريقة التي قمنا بتنفيذها على منصة google colab، وكانت مراحل التنفيذ كما هو موضح في الـ flow chart الموضح أدناه، وكما نجد هذا الكود بكامله في الملحق، وكما سنذكر فيما يلي أهم التعليمات، والملاحظات التي يجب أن نضعها في الحسبان عند كتابة هذا الكود.



الشكل 26: flow Chart للبرنامج المستخدم على منصة colab مع YOLOv4

الكود موجود على الموقع:

<https://colab.research.google.com/drive/1ilYRyFIW1LTzmViSmYRZtrkvMwdvUtRR#scrollTo=a09SdqkYYUgs&uniqifier=1>

5.2.1 تفعيل الـ GPU مع Colab Notebook

كما ذكرنا سابقاً تتيح منصة Colab Notebook إمكانية تنفيذ الكود على GPU، موفر من قبل Google، ويمكن تفعيل هذه الإمكانية بسهولة عن طريق زر edit في الزاوية العليا اليسرى من Colab Notebook، ثم اختيار Notebook settings، ثم اختيار خيار GPU.

5.2.2 نسخ وبناء Darknet

بما أننا سنستخدم YoloV4 ضمن darknet، فيجب علينا أن نقوم بنسخها إلى Colab Notebook، ثم بناءها، ولنسخ darknet، سنقوم بتنفيذ التعليمة التالية:

```
!git clone https://github.com/AlexeyAB/darknet
```

ونلاحظ من هذه التعليمة أننا نستخدم نسخة darknet المعرفة ضمن AlexeyAB's Repository، ويجب قبل أن نقوم ببناء darknet أن نقوم بإجراء عملية ضبط للتناسب مع إمكانيات العتاد hardware المتاح وللقيام بهذه العملية نعدل على الملف المسمى بـ "Makefile" والتعديلات التي سنجرىها هي تأكيد وجود GPU، ومكتبة OPENCV، ومكتبة CUDNN كما يلي:

```
1 # change makefile to have GPU and OPENCV enabled
2 %cd darknet
3 !sed -i 's/OPENCV=0/OPENCV=1/' Makefile
4 !sed -i 's/GPU=0/GPU=1/' Makefile
5 !sed -i 's/CUDNN=0/CUDNN=1/' Makefile
6 !sed -i 's/CUDNN_HALF=0/CUDNN_HALF=1/' Makefile
```

الشكل 27: تعديل الملف Makefile لتأكيد تفعيل GPU, OPENCV

5.2.3 تعريف توابع مساعدة

قمنا في هذا الكود بتعريف عدّة توابع لمساعدتنا في عرض النتائج ورفع الصور إلى الـ Colab Notebook، وتحميل النتائج منه، وهذه التوابع هي:

- `imshow(path)` ويأخذ هذا التابع كدخل له موضع الصورة في بيئة الـ Colab الخاصة بنا، ويقوم بعرض النتائج باستخدام مكتبة OPENCV، ومكتبة matplotlib.
- `Upload()` وهو تابع يستخدم المكتبة google.colab وتسهل هذه المكتبة تعاملنا مع الـ Colab Notebook، فيمكننا تحديد import drive للربط بين google drive، أو يمكننا تحديد import files ونستخدمها لتحميل ملف من الحاسب الخاص بنا إلى الـ Colab Notebook.
- `Download(path)` ويأخذ هذا التابع كدخل له موقع الملف على بيئة الـ Colab الخاصة بنا، ويقوم بتحميل الملف من هذه البيئة إلى حاسبنا.

5.2.4 تحميل الـ Dataset إلى بيئة colab.

الـ dataset المستخدمة:

لقد استخدمنا في هذه المرحلة الـ dataset المعرفة مسبقاً من الموقع الإلكتروني: <https://roboflow.ai/> [10]، وسنستخدم هذا الموقع لأنه يتيح عدد كبير من الخصائص التي تسهل التعامل مع الـ datasets مثل صنع dataset خاصة بنا وتحديد صيغة الـ annotations لتناسب مع أي نوع من الـ models، حيث يمكننا تعديل الـ annotations لإعادة استخدام الـ dataset بسهولة. وسنستخدم الـ [11] Vehicles-OpenImages Dataset وتحتوي هذه الـ dataset على 627 صورة بالأبعاد 1024x751، وتحتوي هذه الـ dataset على الصفوف التالية:

Class Balance

Car	651	<div></div>	over represented
Bus	141	<div></div>	under represented
Motorcycle	140	<div></div>	under represented
Truck	136	<div></div>	under represented
Ambulance	126	<div></div>	under represented

الشكل 28: الصفوف الموجودة في Vehicles-openImageas Dataset، ونسبة كل منها.

ومن هذا التمثيل نلاحظ أن هذه الـ dataset ليست أفضل خيار ويمكن ملاحظة هذا الأمر أيضاً من النتائج التي ظهرت لدينا، لذا استخدمنا في المراحل اللاحقة dataset أخرى ولكن هذه هي أول dataset قمنا بالتدريب عليها.

تهيئة darknet بما يوافق مواصفات الـ dataset المستخدمة:

لتهيئة الـ dataset يجب القيام بالعديد من الخطوات وهي:

- يجب أولاً أن نقوم بإنشاء الملف yolov4-custom.cfg عن طريق نسخ الملف yolov4.cfg من الملف cfg ثم القيام بالتعديلات التالية:

♦ تعديل قيمة patch=64 و subdivisions=16.

♦ تعديل قيم width=614 و height=614 ويمكن اختيار أي قيمة من مضاعفات العدد 32 لتكون قيمة للـ height ولكن يجب الملاحظة أنه كلما كانت هذه القيمة كبيرة كلما كانت عملية التدريب ابetterاً.

♦ تحديد قيمة max_batches من المعادلة:

`max_batches=max ((number of classes) *2000,6000)`

وفي حالتنا عدد الصفوف مساوٍ لـ 5، لذا `max_batches= 10000`.

◆ تحديد قيمة `steps` من المعادلة التالية:

`steps= (80% of max_batches), (90% of max_batches)`

◆ تحديد قيمة المتحول `filters` من المعادلة:

`Filters= (number of classes +5) *3`

ويجب تعديل هذه القيمة في الطبقات التلافيفية الثلاث السابقة لطبقات YOLO.

◆ وأخيراً يجب تعديل عدد الصفوف في كامل الملف لتساوي 5.

- يجب إنشاء الملف `obj.names` ليحوي أسماء الصفوف الموجودة في الـ `dataset` بنفس الترتيب والكتابة الموجودة في الـ `annotations` للـ `dataset`، ويمكن معرفة هذه المعلومات من الملف `classes.txt` المرفق للـ `dataset`.

- إنشاء الملف `obj.data` الذي يحوي على مسارات وجد كل من الصورة التي نريد التدريب عليها، والصور التي نريد الاختبار عليها، وكان وجود الـ `backup weights` التي تظهر بعد كل عدد من الـ `iterations`.
- وأخيراً يجب توليد الملفان `train.txt` و `test.txt` اللذان يحويان على مسارات كل الصور المستخدمة للتدريب أو للاختبار وسنستخدم توابع بسيطة لتوليد هذين الملفين.

5.2.5 تحميل الأوزان الأولية لشبكة YOLOv4

وقمنا في هذه الخطوة بتحميل الأوزان المدربة مسبقاً والمحددة في `AlexyAB repository` على `github`، وهذا يفيد في تسريع عملية التدريب.

5.2.6 تدريب الشبكة

بعد أن انتهينا من كل الخطوات السابقة أصبح بإمكاننا بسهولة تدريب الشبكة العصبونية عن طريق تشغيل الـ `command` التالي:

```
# !./darknet detector train <path to obj.data> <path to custom config> yolov4.conv  
.137 -dont_show -map
```

بعد تعبئة الفراغات بالمعلومات المناسبة الموضحة وأخذت عملية التدريب وقتاً طويلاً حوالي 48 ساعة.

5.2.7 تشغيل كاشف الأشكال الناتج

لتشغيل النموذج الناتج للقيام بكشف الأشكال يجب أن نقوم بتعديل الملف `yolov4-custom.cfg` كما يلي:

```

1 # need to set our custom cfg to test mode
2 %cd cfg
3 !sed -i 's/batch=64/batch=1/' yolov4-custom.cfg
4 !sed -i 's/subdivisions=16/subdivisions=1/' yolov4-custom.cfg
5 %cd ..

```

الشكل 29: تعديل الملف **yolov4-custom.cfg** لتهيئة كاشف الاشكال الناتج للكشف عن الأشكال

وأخيراً نشغل الكاشف عن طريق ال **command** التالي:

```

/./!darknet detector test data/obj.data cfg/yolov4-obj.cfg
/mydrive/yolov4/backup/yolov4-obj_last.weights /mydrive/images/car2.jpg -thresh
0.3

```

وهذا هو أول كود قمنا بتنجيذه.

5.3 الطريقة الثانية

في هذه الطريقة قمنا باستخدام ال **server** الذي وفره لنا المعهد وذلك لأنه يحوي على **GPU** من النوع: **NVIDIA GeForce RTX 2080** ونخص في هذه الطريقة عدّة مراحل مهمة ألا وهي:

5.3.1 التهيئة للتعامل مع ال GPU

قمنا في هذه المرحلة بتنزيل كافة البرامج والمكاتب اللازمة لنتمكن من التعامل مع ال **GPU** ، ولتهيئة الحاسب للمرحلة التالية وهي مرحلة التدريب فقمنا بتنزيل **NVIDIA Driver** موافق لنسخة ال **GPU** الموجودة لدينا، ثم قمنا بتنزيل **Cuda Toolkit**، وأخيراً قمنا بتنزيل المكتبة **CuDNN**، ويجب الانتباه أن أهم خواص هذه المرحلة هو أننا يجب أن نقوم بتنزيل نسخ متوافقة مع المكتبات اللاحقة ومع ال **GPU**، وقد حددنا النسخة الموافقة لكل برنامج من الفصل السابق.

5.3.2 عملية التدريب

ونظراً لأن اللغة المختارة للبرمجة هي **C#**، وأننا استخدمنا **darknet** للتدريب وهي مكتوبة باللغة **C**، لذا فإن طريقة التنزيل هي:

1- تنزيل البرنامج **Cmake**:

نقوم أولاً بتنزيل هذا البرنامج بالنسخة الموافقة لـ **windows**، ثم نقوم بتنصيبه بسهولة.

2- تنزيل **visual Studio 2019 community**، وتنصيبه ويجب الانتباه عند التنصيب أن نقوم بإضافة **Desktop**

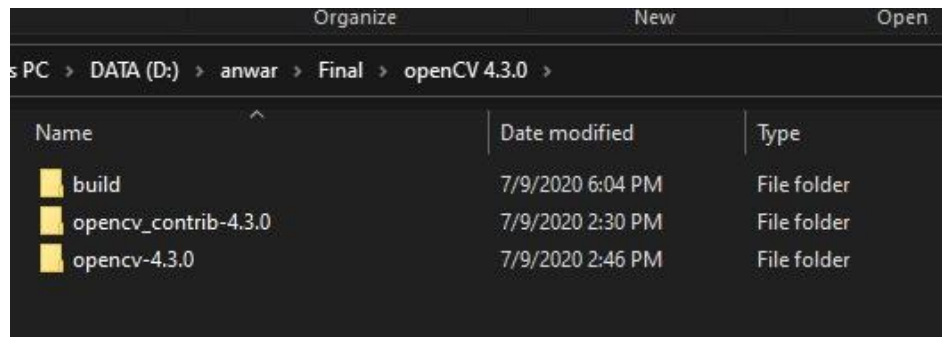
.development with **C++**

3- تنزيل مكتبة **OpenCV** باستخدام البرنامج السابق **Cmake**.

نقوم بتنزيل مكتبة OpenCV بالنسخة التي ذكرناها سابقاً، وبعد انتهاء التنزيل علينا أن نقوم بتنزيل OpenCV - contrib من github repository الموافق لها، مع ملاحظة تنزيل النسخة الموافقة لنسخة OpenCV التي قمنا بتنزيلها.

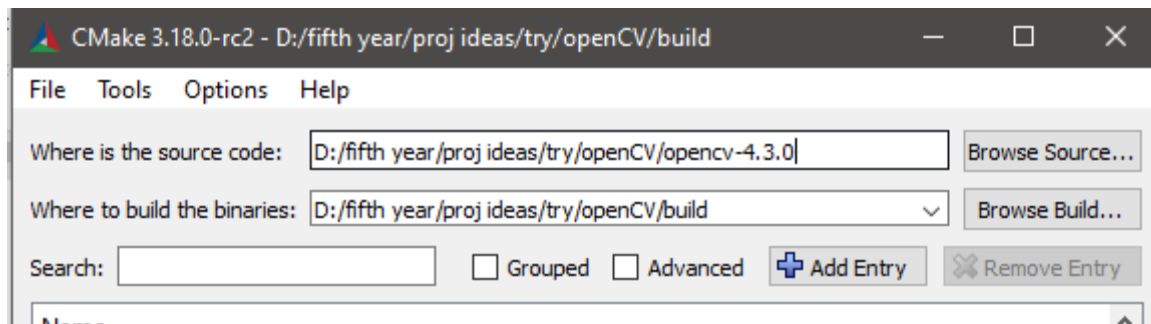
بعد تنزيل هاذين الملفين سنقوم ببنائهما باستخدام CMake كما يلي:

- سنقوم أولاً ببناء المجلد build الذي سيحوي علة الملفات الناتجة كخروج للأداء CMake.



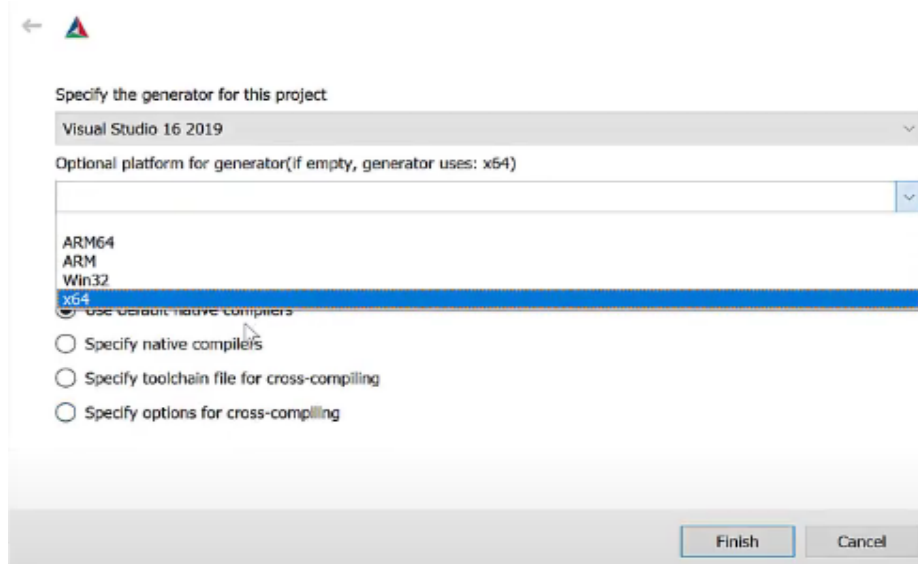
الشكل 30: بناء المجلد build لتنزيل مكتبة openCv باستخدام CMake

- بعد فتح الأداة سنقوم بتحديد المجلد الذي قمنا ببنائه ليحوي الخرج كـ build، كما سنحدد المجلد الذي قمنا بتحميله الذي يحتوي على ملفات المكتبة كـ source، كما يلي:



الشكل 31: تحديد ملفي الـ build والـ source

- قمنا بتحديد الـ platform التي سنقوم باستخدامها لتوليد النص البرمجي وفي حالتنا هي كما يلي في الصورة التالية



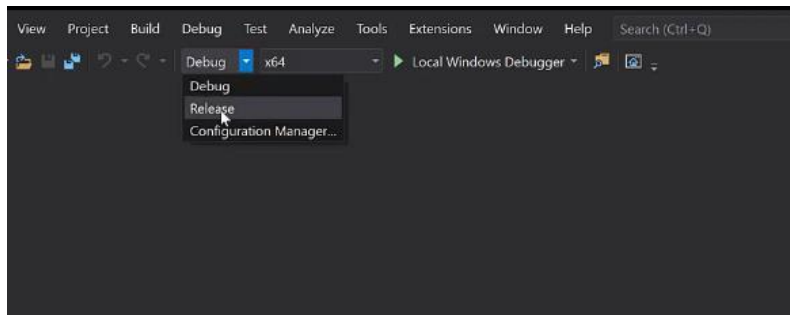
الشكل 32: تحديد الـ platform التي سنستخدمها لتوليد الملفات.

- بعد الانتهاء من هذه الخطوة سنقوم بإجراء أول `configure` وهذه العملية ستأخذ وقتاً وتنتهي بظهور عبارة `configuration done` في نهاية الخرج، وظهور عدد من المربعات الحمراء التي تحوي باقي خيارات الـ `configuration`، وفي هذه المربعات سنبحث عن الخيار `ENABLE_EXTRA_MODULES_PATH` بحيث سنحدد في هذا الخيار مسار الـ `modules` من المجلد `opencv_contrib`، كما سنقوم بوضع إشارة في كل من المربعين `WITH_CUDA`، والمربع `WITH_CUDNN`، مع ملاحظة أننا قمنا بتنصيبهما في الخطوة السابقة، ثم سنقوم بإجراء `configure` مرة ثانية.

- بعد الانتهاء من عملية الـ `configuration` للمرة الثانية ستظهر لدينا مجموعة جديدة من المربعات الحمراء سنبحث فيها عن المربع `CUDA_ARCH_BIN` وسنقوم بحذف كل النسخ الأدنى من نسخة الـ `Compute capability (version)` الموافقة للـ `GPU` الخاص بنا وفي حالتنا هذه النسخة هي 7.5، لذا سنقوم بحذف كل النسخ الأدنى من 7، يرجى الملاحظة أننا يمكننا الحصول على هذا الرقم من صفحة `cuda` على موقع `wikipedia`، عن طريق البحث في جدول الـ `GPUs supported`، عن اسم الـ `GPU` المتوفر لدينا، ومعرفة الرقم الموافق له، وتساهم هذه الخطوة في تسريع عملية البناء بشكل كبير نلاحظ عدم وجود مربعات حمراء، وإن وجدت مربعات حمراء فيجب القيام بإجراء `configuration` مرة ثانية، ثم سنقوم

بالضغط على زر generate، وهذا هو الزر الذي سيقوم بتوليد الحل أو المشروع الذي سنقوم ببنائه لتنصيب مكتبة OpenCV على حاسبنا، وتنتهي هذه المرحلة بظهور العبارتين: Configuration done و Generating done على نافذة الخرج.

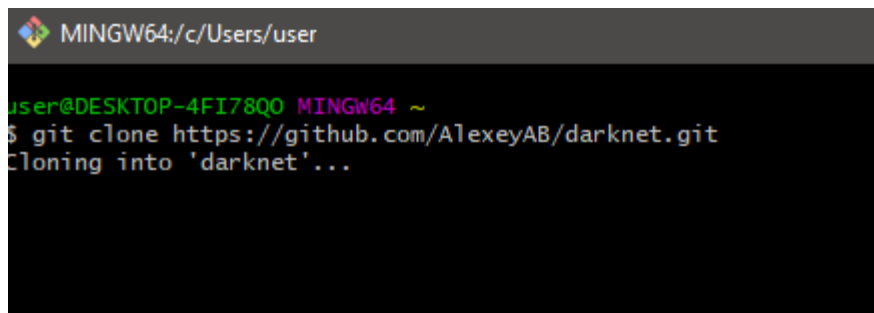
- نذهب إلى الملف build، ونلاحظ أنه الآن يحوي على خرج العمليات السابقة ثم نقوم بفتح ملف ال solution باستخدام visual studio 2019، ونقوم بتغيير نمط العمل إلى Release كما يلي:



الشكل 33: تغيير نمط العمل في visual Studio من debug إلى release

- اخر خطوة هي القيام ببناء الملف BUILD_ALL من المجلد CMake Targets، وستأخذ هذه العملية وقتاً طويلاً، واخيراً سنقوم ببناء الملف INSTALL من نفس المجلد السابق، وبهذه الطريقة نكون قد قمنا بتنصيب المكتبة OpenCV ويمكن التأكد من هذا الأمر عن طريق تنفيذ أحد تعليمات هذه المكتبة.

4- نسخ ال Darknet repository لـ AlexyAB:



الشكل 34: نسخ AlexyAB darknet repo باستخدام git.

- 5- بما أننا قمنا بتنصيب كل من CUDA، و cuDNN ومتطلبات استخدام ال GPU، ومكتبة OpenCV فسنقوم باستخدامهما عند بناء darknet، وذلك لأنهما يساهمان بشكل كبير في تسريع عمليات التدريب وعملية الكشف عن الاشكال، وللقيام بهذا الأمر سنتبع الخطوات التالية:

- سنقوم باستخدام CMake لتنصيب Darknet، فلنبداً بفتح ملف makefile من مجلد darknet الذي قمنا بتنزيله في الخطوة السابقة، ثم سنجري عدّة تعديلات لإخبار أداة CMake أننا نريد أن نقوم ببناء darknet، مع GPU، cuDNN، و OpenCV، عن طريق تعديل القيم صفر من هذا الملف واسناد القيمة 1، عوضاً عنها، كما في الصورة التالية:
- بنفس الطريقة التي قمنا من خلالها بتنصيب OpenCV عن طريق CMake سنقوم بتنصيب darknet، ولكن الخطوة الأولى هي القيام بتحميل ملف الأوزان الموافق لخوارزمية YoloV4، وسنقوم بتحميل هذا

```

1 GPU=1
2 CUDNN=1
3 CUDNN_HALF=0
4 OPENCV=1
5 AVX=0
6 OPENMP=0
7 LIBSO=0
8 ZED_CAMERA=0
9
10 # set GPU=1 and CUDNN=1 to speedup on GPU
11 # set CUDNN_HALF=1 to further speedup 3 x times (Mi
12 # set AVX=1 and OPENMP=1 to speedup on CPU (if erro
13

```

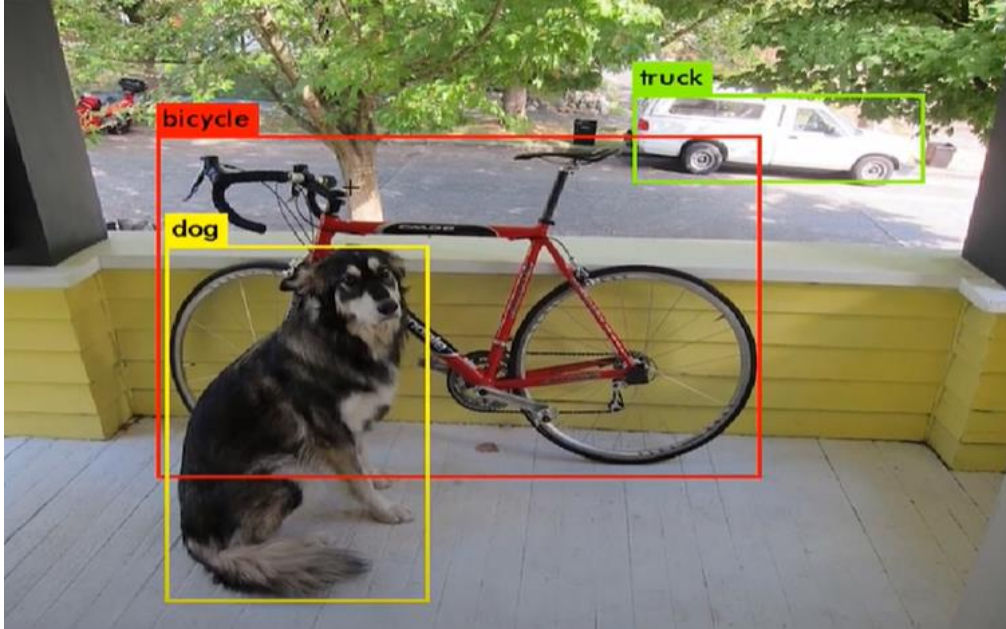
الشكل 35: التعديلات التي سنقوم بإجرائها على ملف Cmakefile من مجلد darknet

الملف من الرابط:

https://github.com/AlexeyAB/darknet/releases/download/darknet_yolo_v3_optimal/yolov4.weights

ثم سنقوم بتحديد المجلد darknet كـ source، والمجلد build كـ build ثم سنقوم باختيار الـ platform، وإجراء أول عملية configure، وبعد انتهاء هذه العملية ستظهر لدينا المربعات الحمراء كما في حالة تنصيب OpenCV، لذا سنقوم بالبحث عن كل من ENABLE_CUDA، ENABLE_CUDNN، ENABLE_OPENCV، وسنضع إشارة لتفعيل كل من هذه الخيارات، ثم سنقوم بإجراء ثاني عملية configure، بعد انتهاء هذه العملية لن نحصل على أي مربعات حمراء لذا يمكننا مباشرة أن نقوم بتوليد المشروع الموافق لبناء darknet، بعد الانتهاء من عملية التوليد سنلاحظ أن مجلد الـ build الذي قمنا بتحديدده يحتوي على المشروع الموافق لذا سنقوم بفتح الملف darknet.sln باستخدام visual studio 2019، بعد تغيير النمط إلى Release سنقوم ببناء الملف ALL_BUILD، وأخيراً سنقوم ببناء الملف INSTALL، وبهذه الطريقة نكون قد بنينا darknet على منصة Windows، ومن خرج المرحلة السابقة لدينا الملف darknet.exe وهو الملف الذي سنستخدمه للكشف عن الأغراض أو للتدريب.

6- يمكننا الآن إجراء عملية الكشف عن الأغراض، وذلك لأننا قمنا بتحميل الأوزان المدربة بتعليمه مشابحة للتعليم المستخدمة في نسخة Yolov4 باستخدام Google colab، وتتم هذه العملية باستخدام التعليم التالية:
./darknet.exe detect cfg/yolov4.cfg yolov4.weights data/dog.jpg
ويكون خرج هذه التعليم بالشكل التالي:



الشكل 36: خرج عملية الـ detection لـ darknet مع الأوزان المدربة مسبقاً و Yolov4

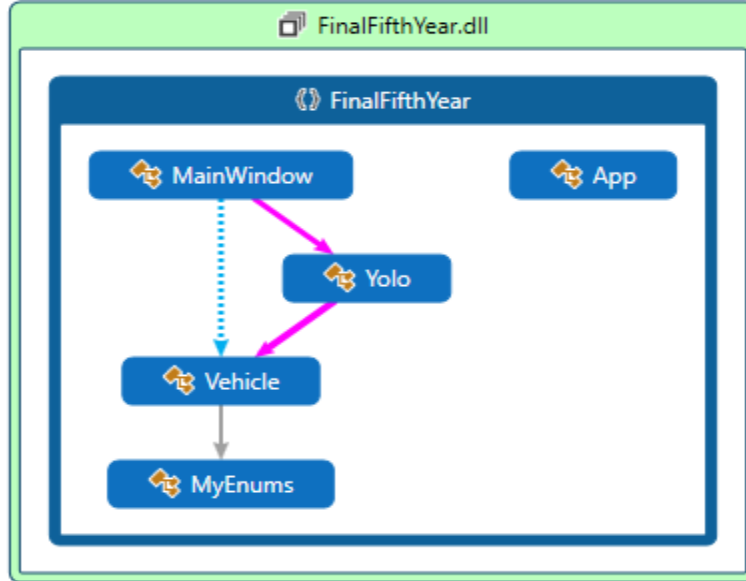
7- بعد أن قمنا بتنصيب darknet على حاسبنا أصبح بإمكاننا بسهولة القيام بعملية التدريب، ولكن علينا أولاً أن نقوم بتحميل الـ dataset، وإجراء الـ configuration بطريقة مماثلة لما قمنا به في الفقرة 4.2.4 [تحميل الـ Dataset إلى بيئة colab](#). بداية على الـ dataset التي قمنا بالتدريب عليها على الـ colab، أي نفس مراحل عملية إجراء الـ configuration، ثم نقوم بتنفيذ التعليم التالية:

```
./darknet.exe detector train data/obj.data cfg/yolov4-custom.cfg yolov4.conv.137 -  
dont_show -map
```

8- وستقوم هذه التعليم بإجراء التدريب والاختبار لأننا قمنا بإضافة الـ flag (map) الذي يقوم برسم منحني التابع الـ loss و الـ map بدلالة عدد الـ iterations، وبهذا تبدأ بسهولة عملية التدريب، التي أخذت وقتاً كبيراً حوالي 48 ساعة، وبعد انتهاء هذه العملية سنحصل على الأوزان الجديدة للنموذج model، وتواجد هذه الأوزان ضمن المجلد backup ضمن مجلد darknet، ويمكن أن نقوم بعملية الكشف عن الأشكال بنفس طريقة المذكورة سابقاً عند التعامل مع Google Colab.

5.3.3 بناء التطبيق النهائي

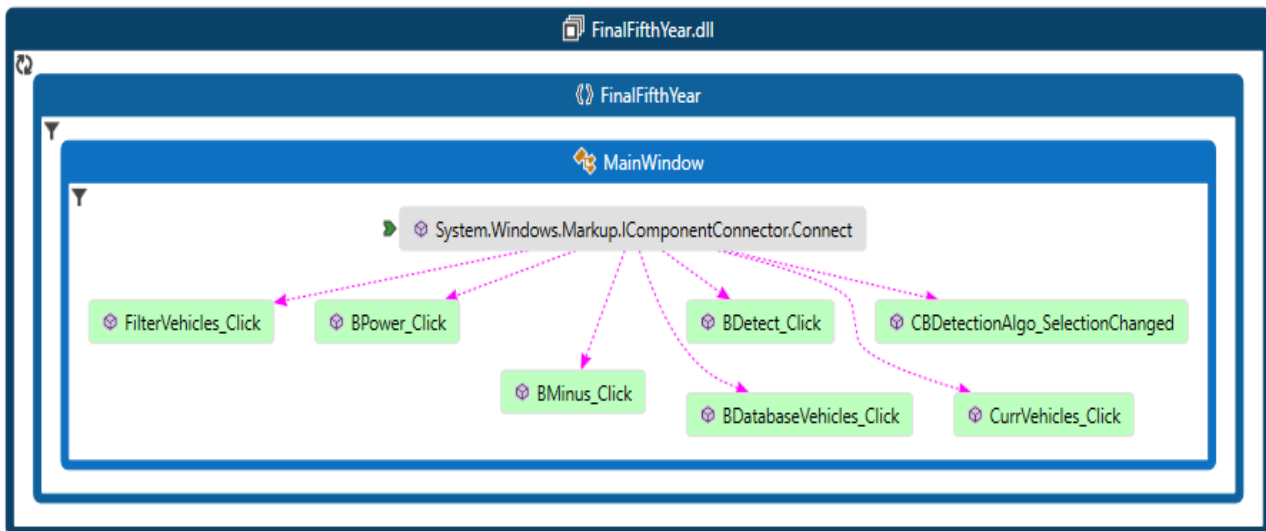
بعد أن حصلنا على الأوزان الجديدة، يمكننا البدء ببناء التطبيق النهائي، وكنا قد ذكرنا سابقاً أن هذا التطبيق هو من النمط WPF، ويتألف هذا التطبيق من الصفوف التالية:



الشكل 37: الصفوف التي يتألف منها التطبيق النهائي.

:MainWindow

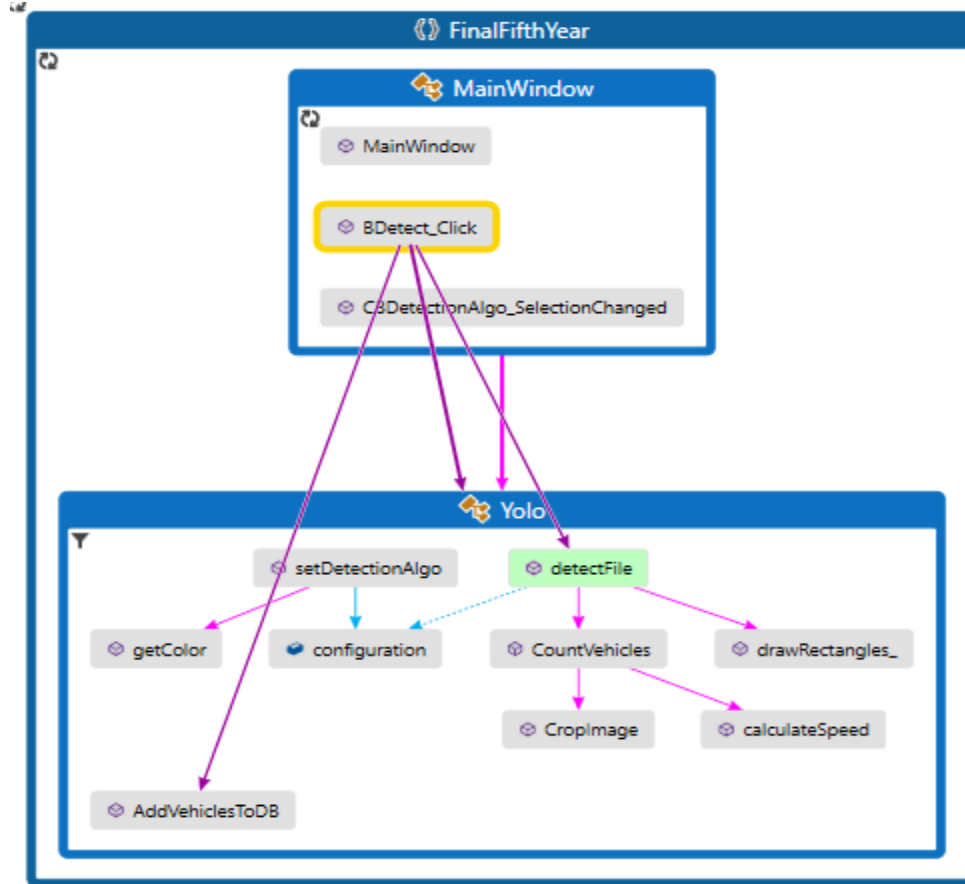
وهو الصف الذي يتم فيه معالجة الأحداث كلها بحيث، يحتوي على تنجيز لكل التوابع التي يقدمها مشروعنا، ويحتو هذا الصف على التوابع التالية:



الشكل 38: توابع الصف MainWindow.

:Yolo

وهو الصف الأساسي في المشروع، ويقوم بكامل العمليات الأساسية من ضمنها عملية التعرف على السيارات، والعمليات المختلفة عليها من عد لهذا السيارات، وحساب سرعتها، وأخيراً عملية إضافتها إلى قاعدة المعطيات التي قمنا بتصميمها، ويتألف هذا الصف من التوابع التالية:



الشكل 39: الصف Yolo، مع التوابع التي يتألف منها، وتوضيح كيف يتم استدعائها.

التابع detectFile()

يقوم هذا التابع بعملية الكشف عن السيارات بالاستعانة بالمكتبة Alturos.Yolo التي قمنا بالحدديث عنها في الفقرة 3.11، وينص هذا التابع على ما يلي:

```

public void detectFile()
{
    vehicles.Clear();
    using (var yoloWrapper = new YoloWrapper(configuration))
    {
        var i = 0;
        using (var video = new VideoCapture(inputFilePath))
        {
            var yoloTracking = new YoloTracking(video.FrameWidth, video.FrameHeight);
            fps = video.Fps;
            while (video.Grab())
            {
                using (var img = video.RetrieveMat())
                {
                    if (img != null)
                    {
                        try
                        {
                            //convert from Mat to byte Array because yolo needs this type to work
                            byte[] imageData = img.ToBytes();
                            //detecting will give us yoloitems
                            var items = yoloWrapper.Detect(imageData)
                                .Where(v => AcceptedTypes.Any(s => (v.Type).Equals(s))).ToArray();
                            //tracking will give us yoloTrackingitem this means that we
                            //also have an objectId associated with each object
                            var trackingItems = yoloTracking.Analyse(items);
                            CountVehicles(img, trackingItems, i);
                            //var editedImage = drawRectangles(img, items);
                            var editedImage = drawRectangles_(img, trackingItems);
                            //this way it will start showing the results faster
                            Cv2.ImShow("image2", editedImage);
                            Cv2.WaitKey(1);
                            editedImage.Dispose();
                        }
                        catch
                        {
                            var st = "frame error";
                        }
                    }
                    i++;
                }
            }
            Cv2.DestroyAllWindows();
        }
    }
}


```

بحيث يتم في هذا التابع تعريف YoloWrapper باستخدام الـ Configuration التي يتم تحديدها من الواجهة البرمائية، ونظراً لكون الـ instance من YoloWrapper كبيرة جداً لأنها تحتوي على تعريف للشبكة العصبونية بأوزانها، وتأخذ جزءاً كبيراً من ذاكرة العمل نلاحظ أن التابع يحتوي العديد على تعريف للـ objects بطريقة using، وذلك لأن هذه الطريقة تمتاز بكونها تقوم

بتدمير ال object عند انتهاء تعليمة ال using، كما أنها تزيله من الذاكرة متفوقة على تعليمة dispose التي قد لا تقوم بهذا الأمر، بعد تعريف ال yolowrapper، قمنا باستخدام المكتبة OpenCvCsharp لخلق VideoReader، واستخدمنا هذا ال object لقراءة الفيديو ثم القيام بتقطيعه إلى frames، ثم قمنا بتعريف YoloTracker لقيام بملاحقة الأغراض التي تمكن Yolo من الكشف عنها، وأخيراً قمنا باستدعاء التابع countVehicles الذي يقوم بعد السيارات، ثم استدعينا التابع drawRectangles الذي يقوم برسم المربعات المحيطة بالأشكال التي تم الكشف عنها، ملاحظتها، وكما استعنا بالمكتبة OpenCvCsharp لعرض الصور الناتجة، وأخيراً قمنا بالتخلص من الصور بعد عرضها حتى نحافظ على ذاكرة العمل، لأن عدم القيام بهذه الخطوة سيجعل أداء البرنامج يتأثر عندما يكون الفيديو طويل، وبهذا الشرح نلاحظ أن هذا التابع من أهم التوابع المنجزة في البرنامج...

قاعدة المعطيات:

إن قاعدة المعطيات التي نقوم بالتخزين فيها بسيطة جداً بحيث تتألف هذه القاعدة من جدول واحد فقط، وهو الجدول Vehicle الموضح في الشكل التالي:

Vehicle	
	Id
	CreationDate
	VehicleType
	Speed
	Status
	ImageURL
	VideoTitle
	Color

الشكل 40: تصميم قاعدة المعطيات التي تتألف من الجدول **Vehicle**، مع توضيح ال attributes التي يتألف منها هذا الصف.

يتم في هذا الصف تخزين السيارات التي نقوم بالكشف عنها، مع معلومات أساسية عنها، وهذه المعلومات هي:

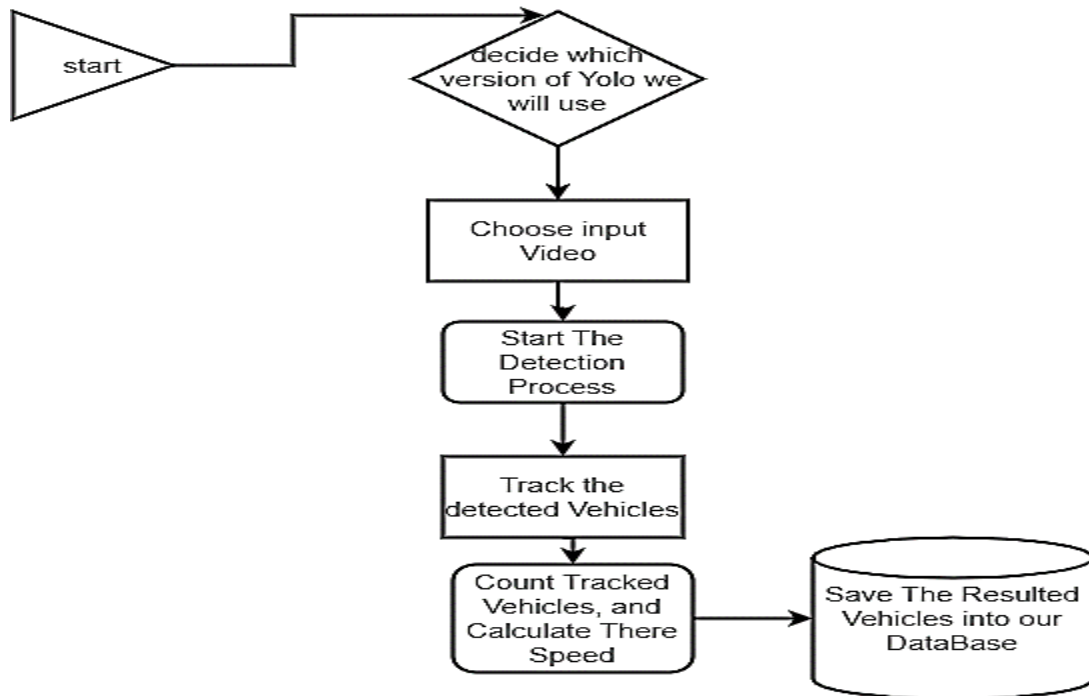
- **VehicleType** أو نوع السيارة وهو عبارة عن string، ويأخذ إحدى القيم التالية (car, bus, truck...).
- **Speed** أو سرعة السيارة وهو من النوع float، ويخزن فيه سرعة السيارة التي قمنا بإيجادها من الفيديو المدخل.
- **ImageURL**، ويخزن في هذا العمود المسار الذي يحتوي على صورة السيارة، التي استطعنا الحصول عليها من التابع cropImage.

- CreationDate، ويتم في هذا العمود تخزين الوقت الذي تم فيه استكشاف هذه السيارة ويفيد هذا العمود عند القيام بعملية الفلترة لمعرفة السيارات التي مرت على الطريق خلال فترة زمنية محددة.
- وغيرها من الاعمدة الأخرى التي تخزن معلومات معرفة لكل سيارة من السيارات، وفيما يلي نجد صورة كمثال عن طبيعة البيانات المخزنة في قاعدة المعطيات.

	Id	CreationDate	VehideType	Speed	Status	ImageURL	VideoTitle
1	37C682BE-04FC-41AB-A18E-0098FC814EEE	2020-09-05 10:43:07.1330000	bus	771.428588867188	0	C:\Users\anwar\source\repos\FinalFifthYear\Fin...	highway.mp4
2	9B5B9E9D-5C2D-48D1-87FB-0138FAFDE2E4	2020-09-08 04:34:52.6870000	car	13.6395494470322	0	C:\Users\anwar\source\repos\FinalFifthYear\Fin...	neww.mp4
3	21AD563F-77FA-4F0C-9D2D-01A3DB8CC535	2020-09-05 10:43:07.1300000	car	771.428588867188	0	C:\Users\anwar\source\repos\FinalFifthYear\Fin...	highway.mp4

الشكل 41: مثال عن طريقة تخزين البيانات المعرفة للسيارة في قاعدة المعطيات.

إن السيناريو الأساسي المطلوب من النظام هو التالي:

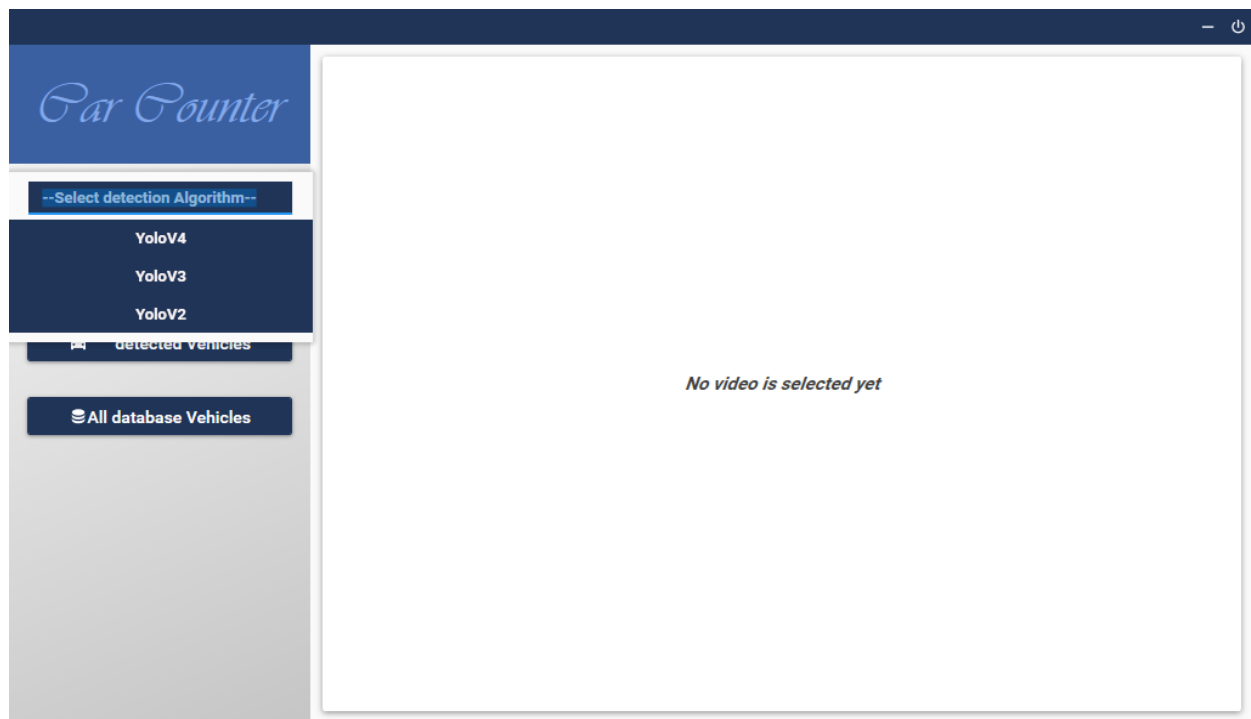


الشكل 42: السيناريو الرئيسي للنظام.

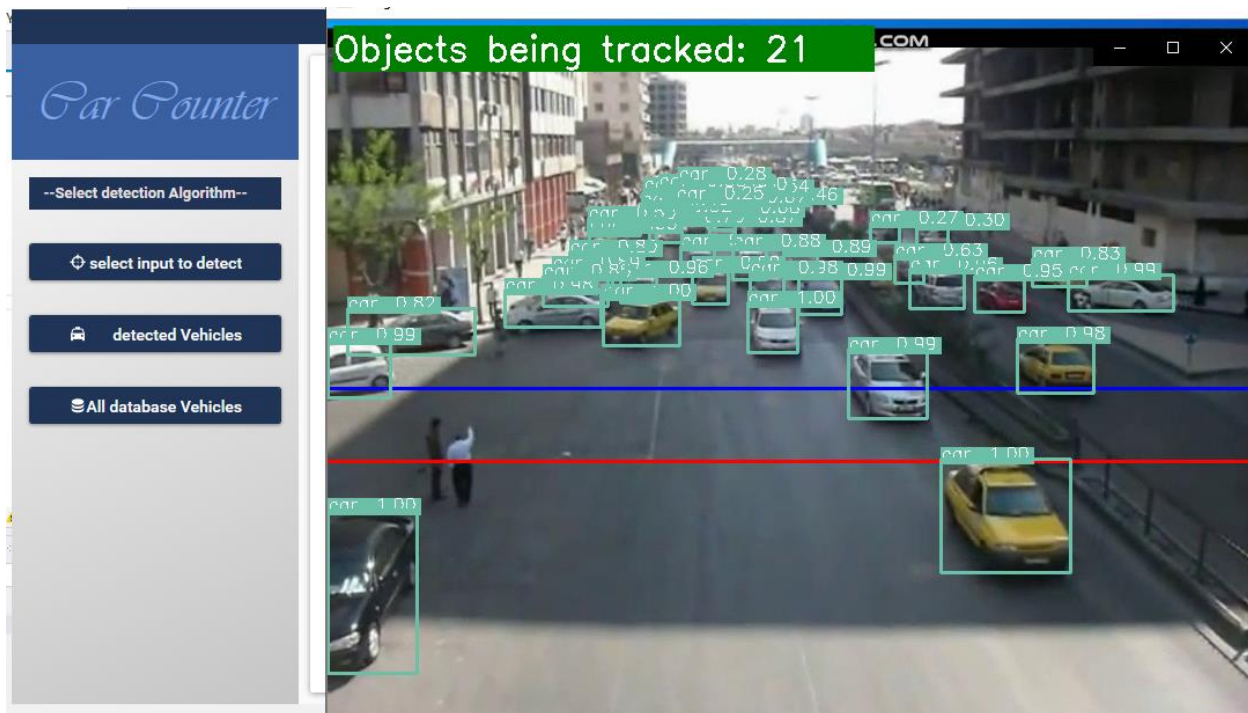
ولتحقيق هذا السيناريو قمنا بخلق الواجهة التالية التي يمكن من خلالها معالجة كافة العمليات المطلوب من النظام تحقيقها، بالشكل التالي:



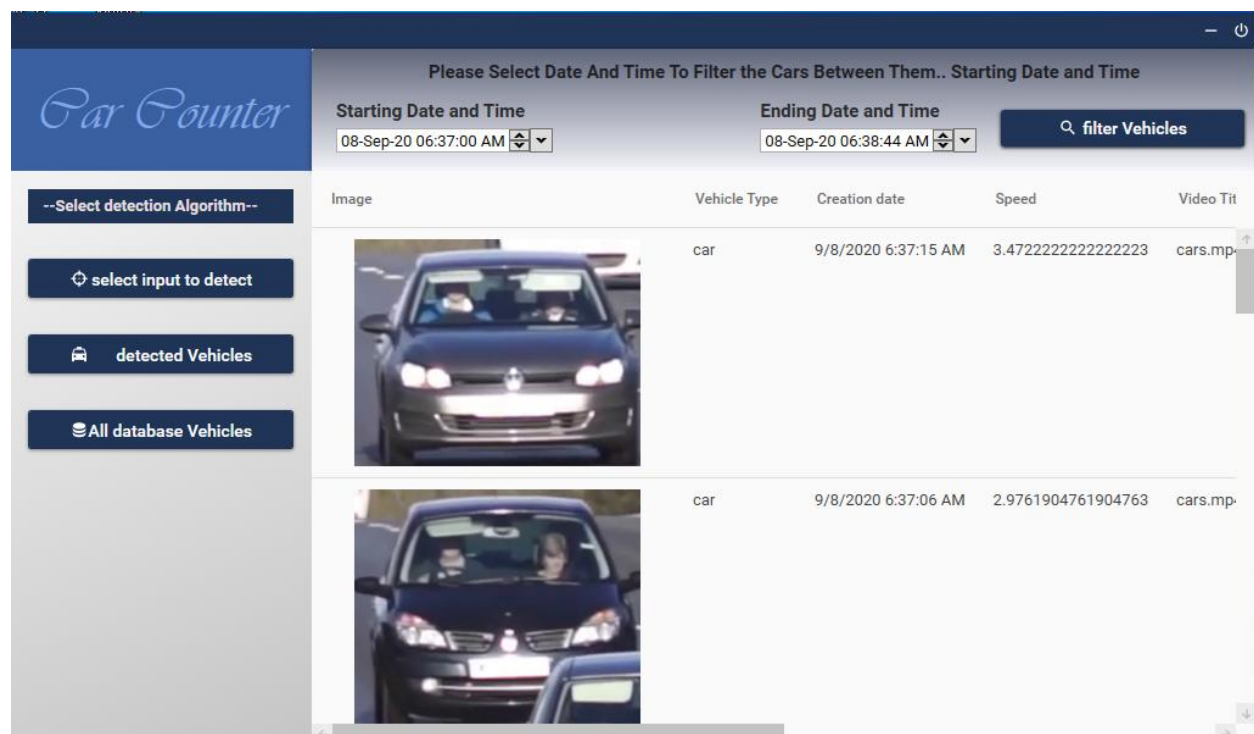
الشكل 44: الواجهة الأساسية للتطبيق.



الشكل 43: واجهة اختيار أحد النسخ من خوارزمية Yolo.



الشكل 46: واجهة عملية الكشف



الشكل 45: واجهة عرض السيارات من قاعدة المعطيات وتطبيق فلترة عليها.

الخاتمة:

Bibliography .6

- [1] S. Ren, K. He, R. Girshick and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," in *Neural Information Processing Systems (NIPS)*, Montreal, Quebec, Canada., 2015.
- [2] K. He, X. Zhang and J. Sun, "Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition," in *13th European Conference on Computer Vision (ECCV)*, Zurich, Switzerland, 2014.
- [3] R. Girshick, "Fast R-CNN," in *IEEE International Conference on Computer Vision (ICCV)*, 2015.
- [4] J. Redmon, S. Divvala, R. Girshick and A. Farhadi, "You Only Look Once:," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Las Vegas, NV, USA, 2016.
- [5] J. Redmon and A. Farhadi, "YOLO9000: Better, Faster, Stronger," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Honolulu, 2017.
- [6] J. Redmon and A. Farhadi, "YOLOv3: An Incremental Improvement," in *the IEEE conference on computer vision and pattern recognition*, Honolulu, Hawaii., 2018.
- [7] "Introduction to Residual Networks - GeeksforGeeks," GeeksforGeeks, 2020. [Online]. Available: <https://www.geeksforgeeks.org/introduction-to-residual-networks/>. [Accessed 2 9 2020].
- [8] A. Bochkovski, C.-Y. Wang and H.-Y. M. Liao, "YOLOv4: Optimal Speed and Accuracy of Object Detection," in *arXiv:2004.10934v*, 2020.
- [9] J. Redmon, *Darknet: Open Source Neural Networks in C*, \url{http://pjreddie.com/darknet/}, 2013 -- 2016.
- [10] "Material Design In XAML," 2020. [Online]. Available: <http://materialdesigninxaml.net>. [Accessed 2 9 2020].
- [11] P. D. Sheriff, "Why Use WPF?," *CODE magazine*, no. 2009- November/December.
- [12] "roboflow," [Online]. Available: <https://roboflow.ai/>. [Accessed 19 8 2020].
- [13] J. Solawetz, "Roboflow vehicles-openImages dataset," june 2020. [Online]. Available: <https://public.roboflow.ai/object-detection/vehicles-openimages>. [Accessed 19 8 2020].
- [14] L. Wen, D. Du, Z. Cai, Z. Lei, M. Ching Chang, H. Qi, j. Lim, M. Hsuan Yang and S. Lyu, "{UA-DETRAC:} {A} New Benchmark and Protocol for Multi-Object Detection and Tracking," *Computer Vision and Image Understanding*, 2020.
- [15] S. a. C. M.-C. a. D. D. a. L. W. a. W. Y. a. D. C. M. a. C. P. a. S. A. a. M. B. a. C. D.-H. a. o. Lyu, "UA-DETRAC 2018: Report of AVSS2018 \& IWT4S challenge on advanced traffic monitoring," in *2018 15th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*, 2018.

- [16] S. a. C. M.-C. a. D. D. a. W. L. a. Q. H. a. L. Y. a. W. Y. a. K. L. a. H. T. a. D. C. M. a. o. Lyu, "UA-DETRAC 2017: Report of AVSS2017 \& IWT4S Challenge on Advanced Traffic Monitoring," in *Advanced Video and Signal Based Surveillance (AVSS), 2017 14th IEEE International Conference on*, 2017.
- [17] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu and A. C.Berg, "SSD: Single Shot MultiBox Detector," 2016.