Text Detection from an Image

A Project
Presented By
Department of Computer Science
State University of New York Polytechnic Institute
At Utica/Rome
Utica, New York

In partial fulfillment
of the requirements of the
Master of Science Degree

By
Piyush Jain Goda

December 2020
DECLARATION

I declare that this project is my work and has not been submitted in any form for another degree or diploma at any university or other institute of tertiary education. Information derived from the published and unpublished work of others has been acknowledged in the text and a list of references is given.

Piyush Jain Goda

Piyush Jain Goda
SUNY POLYTECHNIC INSTITUTE

DEPARTMENT OF COMPUTER SCIENCE

Approved and recommended for acceptance as a project in partial fulfillment of the requirements for the degree of Master of Science in Computer and information science

Piyush Jain Goda

Piyush Jain Goda
Date : 12/11/2020

____________________________
Dr. Bruno R. Andriamanalimanana
Advisor

____________________________
Dr. Jorge E. Novillo

____________________________
Dr. Scott Spetka
ABSTRACT

Recently, a variety of real-world applications have triggered a huge demand for techniques that can extract textual information from images and videos. Therefore, image text detection and recognition have become active research topics in computer vision. The current trend in object detection and localization is to learn predictions with high capacity deep neural networks trained on a very large amount of annotated data and using a high amount of processing power. In this project, I have built an approach for text detection using the object detection technique. Our approach is to deal with the text as objects. We use an object detection method, YOLO (You Only Look Once), to detect the text in the images. We frame object detection as a regression problem to spatially separated bounding boxes and associated class probabilities. YOLO, a single neural network, that predicts bounding boxes and class probabilities directly from full images in one evaluation. Since the whole detection pipeline is a single network, it can be optimized end-to-end directly on detection performance. The MobileNet pre-trained deep learning model architecture was used and modified in different ways to find the best performing model. The goal is to achieve high accuracy in text spotting. Experiments on standard datasets ICDAR 2015 demonstrate that the proposed algorithm significantly outperforms methods in terms of both accuracy and efficiency.
ACKNOWLEDGEMENTS

I take this opportunity with much pleasure to thank all the people who have helped me through the course of our journey towards this report.

I sincerely thank my project advisor Dr. Bruno Andriamanalimanana for his guidance, help, and motivation. Apart from the subject of my project, I learned a lot from him, I am sure it will be useful to me in different stages of my life. I would like to express my gratitude to him for providing the necessary information regarding the project and for his constant guidance, supervision, kind co-operation, and invaluable support in all aspects.

I would like to express my sincere thanks to Dr. Jorge E. Novillo and Dr. Scott Spetka for being on the project review committee. I have always enjoyed your classes.

Finally, I would like to express my heartfelt thanks to God, my parents, and colleagues for their blessings and constant support.

Piyush Jain Goda
# Table of Contents

1. Introduction                                       1
2. Background
   2.1 Computer Vision                                 2
   2.2 Object Detection                                3
   2.3 Convolution Neural Network                      4
   2.4 How does CNN work?                             5
      2.4.1 Convolution                                5
      2.4.2 Applying Activation Function               5
      2.4.3 Pooling                                    6
      2.4.4 Flattening                                 6
      2.4.5 Full connection                            6
   2.5 Pre-trained Models                              7
   2.6 Algorithms for Object detection                 7
      2.6.2 Fast R-CNN                                 8
      2.6.3 Faster R-CNN                               9
      2.6.4 YOLO (You Only Look Once)                   9
      2.6.5 Single Shot Detector (SSD)                 10
3. Design                                             12
   3.1 Overview                                        12
   3.2 Tools and Technology Used                       12
   3.3 Dataset                                         12
   3.4 Preprocessing Task                              12
   3.5 Model                                           13
   3.6 YOLO Algorithm                                  13
      3.6.1 The Predictions Vector                      14
      3.6.2 Architecture                               14
   3.7 The Loss Function                               15
      3.7.1 Classification Loss                         16
      3.7.2 Localization Loss                           16
      3.7.3 Confidence Loss                             17
      3.7.4 Total Loss                                  17
   3.8 Non-maximal suppression                          18
4 Implementation                                     19
   4.1 Dataset                                         19
   4.2 Preprocessing                                    19
   4.3 The Model                                       21
5 Conclusion                                          24
   5.1 Accomplishments and lessons learned              24
   5.2 Limitation                                      24
   5.3 Future work                                     24
Recently, text detection and recognition became pragmatic research topics with quick advancement and growth in the world of computer vision. The widespread of smartphones and digital cameras provide us with an enormous number of images that contain text. Text in natural images gives precious information about the content of the image that helps in a lot of applications like automatic sign reading, navigation, language translation, license plate reading, content-based image search, many vision-based applications. For example, in blind navigation assistance systems, it is considered a useful component of navigation devices when it successfully recognizes the text on the street signs and shows directions for blind people.

Texts in images, contain valuable information, and provide cues about images. So, the computer needs to understand the images. It is a complex task to detect text from the images for many reasons like different types of text patterns like font size, style, orientations, colors, background outlier like the text characters. Due to the circumstances, there are some challenges faced during the detection of objects like:

- Positioning: the object in the image can be positioned in various aspects.
- Rotation: the object can be in various aspects of the image.
- Occlusion: some part of the object in the image may not be visible.
- Scale: the size of the object may vary.

The above are some challenges that should be considered while developing an object detection system. Based on the attributes it is difficult to accurately conduct text detection and thus leads to image classification and recognition problems.

Deep learning theory has been well researched and developed in image processing and pattern recognition. This paper attempts to convert the multi-classification problem of target detection into binary classification (background and text) and apply it to text detection in images. Based on the YOLO model, the model introduced multiple technologies such as an anchor mechanism in Fast-R-CNN. With the advantage of maintaining the original speed, the accuracy was improved. At the same time, YOLO also simplified the model and facilitated the optimization of the model.
2.1 Computer Vision

Computer vision technology is one of the most promising areas of research within artificial intelligence and computer science. It offers tremendous advantages for businesses in the modern era. It is a multidisciplinary field that could broadly be called a subfield of artificial intelligence and machine learning, which may involve the use of specialized methods and make use of general learning algorithms.[1]

The goal of computer vision is to understand the content of digital images. Typically, this involves developing methods that attempt to reproduce the capability of human vision. Understanding the content of digital images may involve extracting a description from the image, which may be an object, a text description, a three-dimensional model, and so on. In many computer vision applications, the major task is to recognize things in images. For example:

- Object Classification: What broad category of object is in this image?
- Object Verification: Is the object in the image?
- Object Detection: Where are the objects in the image?
- Object Recognition: What objects are in this photograph and where are they?

To find the text in an image, firstly, we must detect the object in an image.
2.2 Object Detection

Object detection is a computer vision technique that allows us to identify and locate objects in an image or video. With this kind of identification and localization, object detection can be used to count objects in an image and determine and track their precise locations, all while accurately labeling them. [2] Object detection is commonly confused with image recognition, so before we proceed, we must clarify the distinctions between them.

![Figure 2.2: Differences between Image recognition and Object Detection](image)

Image recognition assigns a label to an image. A picture of a dog receives the label “dog”. A picture of two dogs still receives the label “dog”. Object detection, on the other hand, draws a box around each dog and labels the box “dog”. The model predicts where each object is and what label should be applied. In that way, object detection provides more information about an image than recognition.

Object detection can be broken down into machine learning-based approaches and deep learning-based approaches. In more traditional ML-based approaches, computer vision techniques are used to look at various features of an image, such as the color histogram or edges, to identify groups of pixels that may belong to an object. These features are then fed into a regression model that predicts the location of the object along with its label.

On the other hand, deep learning-based approaches employ convolutional neural networks (CNNs) to perform end-to-end, unsupervised object detection, in which features do not need to be defined and extracted separately.

Deep learning-based object detection models typically have two parts.

- An encoder takes an image as input and runs it through a series of blocks and layers that learn to extract statistical features used to locate and label objects.
• Outputs from the encoder are then passed to a decoder, which predicts bounding boxes and labels for each object. The simplest decoder is a pure regressor. The regressor is connected to the output of the encoder and predicts the location and size of each bounding box directly.

In object detection and recognition, researchers have used deep learning for learning features directly from the image pixels, which are more effective than the manual features. Recently deep learning-based algorithms remove the manual features extraction methods and directly use features extracting methods from the original images.

There are various deep learning models that are used for object detection such as RCNN, Fast RCNN, Faster RCNN, YOLO, SSD, etc. These models are originated from a deep learning algorithm Convolution Neural Network (CNN), which is mainly used to detect features in an image.

2.3 Convolution Neural Network

A Convolutional Neural Network (CNN) is a multilayered neural network with a special architecture to detect complex features in data. CNN’s have been used in image recognition, powering vision in robots, and for self-driving vehicles.[3]

A Convolutional Neural Network (ConvNet/CNN) is a Deep Learning algorithm that can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image, and be able to differentiate one from the other. The role of the ConvNet is to reduce the images into a form that is easier to process, without losing features which are critical for getting a good prediction.

Figure 2.3: Example for CNN sequence for classifying handwritten digits
Once CNN is built, it can be used to classify the contents of different images. All we must do is feed those images into the model. Let us understand some concepts of CNNs and the steps of building one.

2.4 How does CNN work?

Images are made up of pixels. Each pixel is represented by a number between 0 and 255. Therefore, each image has a digital representation which is how computers can work with images.

2.4.1 Convolution

A convolution is a combined integration of two functions that shows you how one function modifies the other.[4]

\[
(f * g)(t) \overset{\text{def}}{=} \int_{-\infty}^{\infty} f(\tau)g(t - \tau) \, d\tau = \int_{-\infty}^{\infty} f(t - \tau)g(\tau) \, d\tau.
\]

Figure 2.4: Convolution Function

There are three important items to mention in this process:

- The input image is the image being detected.
- the feature detector is a matrix, usually 3x3. A feature detector is also referred to as a kernel or a filter.
- the feature map.

Intuitively, the matrix representation of the input image is multiplied with the feature detector to produce a feature map, also known as an activation map. This step aims to reduce the size of the image and make processing faster and easier. Some of the features of the image are lost in this step. The main features of the image that are important in image detection are retained. These features are the ones that are unique to identifying that specific object.

2.4.2 Applying Activation Function

In this step, we apply the activation function to increase non-linearity in CNN. It is just a function that you use to get the output of the node. It is also known as a transfer function. Images are made of different objects that are not linear to each other. Without applying this
function, the image classification will be treated as a linear problem while it is a non-linear one. There are many activation functions such as Sigmoid, Tanh, ReLU, Leaky ReLU, etc.

2.4.3 Pooling

A problem with the output feature maps is that they are sensitive to the location of the features in the input. One approach to address this sensitivity is to downsample the feature maps. This has the effect of making the resulting down-sampled feature maps more robust to changes in the position of the feature in the image, referred to by the technical phrase “local translation invariance.”

Pooling layers provide an approach to downsampling feature maps by summarizing the presence of features in patches of the feature map. There are different types of pooling, for example, max pooling and average pooling. Max pooling works by placing a matrix of 2x2 on the feature map and picking the largest value in that box. The 2x2 matrix is moved from left to right through the entire feature map picking the largest value in each pass. These values then form a new matrix called a pooled feature map. This helps reduce overfitting, which would occur if the CNN is given too much information, especially if that information is not relevant in classifying the image.

2.4.4 Flattening

Once the pooled featured map is obtained, the next step is to flatten it. Flattening involves transforming the entire pooled feature map matrix into a single column which is then fed to other layers of the neural network for processing.

2.4.5 Full connection

After flattening, the flattened feature map is passed through other layers of the neural network. This step is made up of the input layer, the fully connected layer, and the output layer. The output layer is where we get the predicted classes. The information is passed through the network and the error of prediction is calculated. The error is then backpropagated through the system to improve the prediction. The final figures produced by the neural network do not usually add up to one. However, these figures must be brought down to numbers between zero and one, which represent the probability of each class. This is the role of the SoftMax function.
2.5 Pre-trained Models

Deep convolutional neural network models may take days or even weeks to train on very large datasets. A way to short-cut this process is to re-use the model weights from pre-trained models that were developed for standard computer vision benchmark datasets, such as the ImageNet image recognition tasks. Top-performing models can be downloaded and used directly or integrated into a new model for computer vision problems. The rapid developments in computer vision, and by extension – image classification have been further accelerated by the advent of transfer learning. To put it simply, transfer learning allows us to use a pre-existing model, trained on a huge dataset, for our tasks. Consequently, reducing the cost of training new deep learning models and since the datasets have been vetted, we can be assured of the quality.[4]

There are many pre-trained models for image classification that are state-of-the-art (SOTA) and are widely used in the industry as well. They are EfficientNet, InceptionV3, MobileNet, MobileNetV2, ResNet50, VGG16, etc.

Why should I use a pre-trained model?

- The issue with not using one is that depending on what experience you have, you will spend a serious amount of time training your model from scratch. You will have to do plenty of calculations and experiments to build a proper CNN architecture. Think about how many design questions you will have to sort out: How many layers do I need? What about pooling? Do I put in stacks? Moreover, the complexity of the data set will weigh in as well. [5]
- You might not have a data set that is large enough so that your model can generalize well enough and you might not have the computational resources for that either.
- Keep in mind that ImageNet has 1000 classes, so the pre-trained models have been trained to work on a lot of different things.
- The hard work of optimizing the parameters has already been done for you, now what you have to do is fine-tune the model by playing with the hyperparameters so, in that sense, a pre-trained model is a life-saver.

2.6 Algorithms for Object detection

The problem of identifying the location of an object in an image is called localization. Predicting the location of the object along with the class is called Object Detection. In place of predicting the class of object from an image, we now must predict the class as well as a rectangle called bounding box containing that object. To detect all kinds of objects in an image, we want our algorithm to be able to classify and localize all the objects in an image,
not just one. So, the idea is, just crop the image into multiple images and run CNN for all the cropped images to detect an object.

Object Detection is modeled as a classification problem where we take windows of fixed sizes from input images at all the possible locations to feed these patches to an image classifier. Each window is fed to the classifier which predicts the class of the object in the window. Hence, we have to know both the class and location of the objects in the image. Well, there are a few more problems. How do you know the size of the window so that it always contains the image?

As you can see that the object can be of varying sizes. To solve this problem an image pyramid is created by scaling the image. Idea is that we resize the image at multiple scales and we count on the fact that our chosen window size will completely contain the object in one of these resized images. Most commonly, the image is down sampled until a certain condition typically a minimum size is reached. On each of these images, a fixed size window detector is run. It is common to have as many as 64 levels on such pyramids. Now, all these windows are fed to a classifier to detect the object of interest. This will help us solve the problem of size and location.

There is one more problem, aspect ratio. A lot of objects can be present in various shapes like a sitting person will have a different aspect ratio than a standing person or sleeping person. We shall cover this a little later in this post. There are various methods for object detection like RCNN, Fast RCNN, Faster-RCNN, YOLO, SSD, etc. Below, is a brief description of the algorithms that detect the object in an image. [6]

2.6.1 Region-based Convolution Neural Network (R-CNN)

Since we had modeled object detection into a classification problem, success depends on the accuracy of classification. It was impossible to run CNNs on so many patches generated by the sliding window detector. CNN’s were too slow and computationally very expensive. R-CNN solves this problem by using an object proposal algorithm called Selective Search which reduces the number of bounding boxes that are fed to the classifier to close to 2000 region proposals. The selective search uses local cues like texture, intensity, color, and/or a measure
to generate all the possible locations of the object. Now, we can feed these boxes to our CNN based classifier. Remember, a fully connected part of CNN takes a fixed-sized input so, we resize all the generated boxes to a fixed size and feed to the CNN part. Hence, there are 3 important parts of R-CNN:

- Run Selective Search to generate probable objects.
- Feed these patches to CNN, followed by SVM to predict the class of each patch.
- Optimize patches by training bounding box regression separately.

### 2.6.2 Fast R-CNN

Fast RCNN uses the ideas from SPP-net and RCNN and fixes the key problem in SPP-net i.e. they made it possible to train end-to-end. To propagate the gradients through spatial pooling, it uses a simple back-propagation calculation which is very similar to max-pooling gradient calculation with the exception that pooling regions overlap and therefore a cell can have gradients pumping in from multiple regions.

In Fast R-CNN, they added the bounding box regression to the neural network training itself. So, now the network had two heads, classification head, and bounding box regression head. This multitask objective is a salient feature of Fast R-CNN as it no longer requires training of the network independently for classification and localization. These two changes reduce the overall training time and increase accuracy.

### 2.6.3 Faster R-CNN

The slowest part in Fast RCNN was a selective search or edge boxes. Faster R-CNN replaces selective search with a very small convolutional network called Region Proposal Network to generate regions of interest. To handle the variations in aspect ratio and scale of objects, Faster R-CNN introduces the idea of anchor boxes. We apply bounding box regression to improve the anchor boxes at each location. So, RPN gives out bounding boxes of various sizes with the corresponding probabilities of each class. The varying sizes of bounding boxes can be passed further by apply Spatial Pooling just like Fast-RCNN. The remaining network is similar to Fast-RCNN.

Faster-RCNN is 10 times faster than Fast-RCNN with a similar accuracy of datasets like VOC-2007. That’s why Faster-RCNN has been one of the most accurate object detection algorithms. Here is a quick comparison between various versions of R-CNN.

| Speed comparison |
|------------------|-----------------|-----------------|
| Test time per image (with proposals) | 50 seconds | 2 seconds | 0.2 seconds |
| (Speedup) | 1x | 25x | 250x |
| mAP (VOC 2007) | 66.0 | 66.9 | 66.9 |

Figure 2.6: Speed comparison of R-CNN, Fast R-CNN, and Faster R-CNN
So far, all the methods discussed handled detection as a classification problem by building a pipeline where first object proposals are generated and then these proposals are sent to classification/regression heads. However, there are a few methods that pose detection as a regression problem. Two of the most popular ones are YOLO and SSD. These detectors are also called single shot detectors.

2.6.4 YOLO (You Only Look Once)

For YOLO, detection is a simple regression problem that takes an input image and learns the class probabilities and bounding box coordinates. YOLO divides each image into a grid of $S \times S$ and each grid predicts $N$ bounding boxes and confidence. The confidence reflects the accuracy of the bounding box and whether the bounding box contains an object. YOLO also predicts the classification score for each box for every class in training. You can combine both the classes to calculate the probability of each class being present in a predicted box. We have run our image on CNN only once. Hence, YOLO is super-fast and can be run in real-time. Another key difference is that YOLO sees the complete image at once as opposed to looking at only generated region proposals in the previous methods. So, this contextual information helps in avoiding false positives. However, one limitation for YOLO is that it only predicts 1 type of class in one grid hence, it struggles with very small objects. We discuss YOLO further in the following section.

2.6.5 Single Shot Detector (SSD)

Single Shot Detector achieves a good balance between speed and accuracy. SSD runs a convolutional network on input image only once and calculates a feature map. Then, we run a small $3 \times 3$ sized convolutional kernel on this feature map to predict the bounding boxes and classification probability. SSD also uses anchor boxes at various aspect ratios similar to Faster-RCNN and learns the off-set rather than learning the box. To handle the scale, SSD predicts bounding boxes after multiple convolutional layers. Since each convolutional layer operates at a different scale, it can detect objects of various scales.

![Figure 2.7: The performance metric for Fast R-CNN, Faster R-CNN, YOLO, and SSD](image-url)
Currently, Faster R-CNN is the choice if you are fanatic about the accuracy numbers. However, if you are strapped for computation, SSD is a better recommendation. Finally, if accuracy is not too much of a concern but you want to go super-fast, YOLO will be the way to go.
3.1 Overview

In this project, we design a model to detect text in an image using YOLO (You Only Look Once). Therefore, the input of this project is the image which may or may not contain text. Then we preprocess the image so that the image size is compatible with the model we had developed to detect a text on that image and we run the image with our model to get an output as an image that highlighting the text content in that image.

3.2 Tools and Technology Used

The algorithm is developed in a Python programming language. So, I have used Jupyter Notebook as a platform to develop the model. While for the image classification, I have used Keras with TensorFlow as a backend. Keras being a deep-learning framework provides high-level features to train machine learning models. TensorFlow is an open-source framework provided by Google. It can be considered as a pioneer in the research being conducted by individual artificial intelligence enthusiasts. The majority of tech companies utilize the TensorFlow framework as it provides robust and scalable functionalities and tools to create and train machine learning models.[7] While Keras is a high-level framework that can use TensorFlow to perform various tasks ranging from text processing to training models.

3.3 Dataset

ICDAR 2015 Incidental Text (IC15) is Challenge 4 of the ICDAR 2015 Robust Reading Competition.[8] This challenge features incidental scene text images taken by Google Glasses without taking care of positioning, image quality, and viewpoint. Consequently, the dataset exhibits large variations in text orientation, scale, and resolution, making it much more difficult than previous ICDAR challenges. The dataset consists of two folders, one consists of images and the other is the ground truth of the images. "Ground truth" refers to information collected on location. Ground truth allows image data to be related to real features and materials on the ground.[9]

3.4 Preprocessing Task

Data pre-processing or data cleansing is a crucial step before building the model. Image pre-processing is the term for operations on images at the lowest level of abstraction. The aim of pre-processing is an improvement of the image data that suppresses undesired distortions or enhances some image features relevant for further processing and analysis tasks. Some
Examples of data pre-processing include outlier detection, missing value treatments, and remove unwanted or noisy data.

Images in the dataset had differing sizes, therefore images had to be resized before being used as input to the model. Therefore, we down sampled the images to a fixed resolution of $512 \times 512$. Accordingly, the ground truth of the images is modified as well.

3.5 Model

To develop the required algorithm, the algorithm uses a pre-trained model namely MobileNetV2. MobileNetV2 is one of the families of neural network architectures released by Google to be used on machines with limited computing power, like mobile devices. They strive to provide state of the art accuracy while requiring as little memory and computing power as possible. This makes them a very fast family of networks to use for image processing.[10]

MobileNets i.e., MobileNetV1 and MobileNetV2 achieve this performance by reducing dramatically the number of learnable parameters, which also makes them faster and easier to train compared to more traditional networks. MobileNetV2 is a significant improvement over MobileNetV1 and pushes the state of the art for mobile visual recognition including classification, object detection, and semantic segmentation. For instance, for detection, when paired with Single Shot Detector Lite, MobileNetV2 is about 35 percent faster with the same accuracy as MobileNetV1.[11]

3.6 YOLO Algorithm

To detect the objects in an image, the YOLO algorithm is used in this project. YOLO (You Only Look Once) is a state-of-the-art object detection architecture.[12]

YOLO is a deep learning model that can predict object classes and location. It belongs to the group of classification algorithms. YOLO uses a single neural network that predicts bounding boxes and class probabilities directly from full images in one evaluation. Since YOLO makes predictions with a single network evaluation, YOLO is extremely fast. It is more than 1000x faster than R-CNN and 100x faster than Fast R-CNN.

So, to put it simply, you take an image as input, pass it through a neural network that looks similar to a normal CNN, and you get a vector of bounding boxes and class predictions in the output.
3.6.1 The Predictions Vector

The first step to understanding YOLO is how it encodes its output. The input image is divided into an S x S grid of cells. For each object that is present on the image, one grid cell is said to be responsible for predicting it. That is the cell where the center of the object falls into.

Each grid cell predicts ‘B’ bounding boxes as well as ‘C’ class probabilities. The bounding box prediction has 5 components: (x, y, w, h, confidence).

- The (x, y) coordinates represent the center of the box, relative to the grid cell location. These coordinates are normalized to fall between 0 and 1.
- The (w, h) box dimensions are also normalized to [0, 1], relative to the image size.
- The confidence prediction represents the IOU between the predicted box and any ground truth box. The concept of Intersection over Union (IOU) is frequently used as an evaluation metric to measure the accuracy of an object localizer.

Each grid cell predicts C conditional class probabilities, Pr(Classi | Object). These probabilities are conditioned on the grid cell containing an object. We only predict one set of class probabilities per grid cell, regardless of the number of boxes B. At test time we multiply the conditional class probabilities and the individual box confidence predictions,

\[ Pr(Classi | Object) \times Pr(Object) \times IOU = Pr(Classi) \times IOU \]

which gives us class-specific confidence scores for each box. These scores encode both the probability of that class appearing in the box and how well the predicted box fits the object.

The network only predicts one set of class probabilities per cell, regardless of the number of boxes B. That makes S x S x C class probabilities in total. Adding the class predictions to the output vector, we get a S x S x (B * 5 +C) tensor as output.[13]

3.6.2 Architecture

Changes to loss functions for better results is interesting. Two things stand out:

- Differential weight for confidence predictions from boxes that contain objects and boxes that do not contain an object during training.
- Predict the square root of the bounding box width and height to penalize error in small objects and large objects differently.
Fast YOLO uses a neural network with fewer convolutional layers and fewer filters in those layers. Other than the size of the network, all training and testing parameters are the same between YOLO and Fast YOLO.

YOLO predicts multiple bounding boxes per grid cell. At training time we only want one bounding box predictor to be responsible for each object. We assign one predictor to be “responsible” for predicting an object based on which prediction has the highest current IOU with the ground truth. This leads to specialization between the bounding box predictors. Each predictor gets better at predicting certain sizes, aspect ratios, or classes of objects, improving overall recall.

### 3.7 The Loss Function

YOLO predicts multiple bounding boxes per grid cell. To compute the loss for the true positive, we only want one of them to be responsible for the object. For this purpose, we select the one with the highest IOU (intersection over union) with the ground truth. This strategy leads to specialization among the bounding box predictions. Each prediction gets better at predicting certain sizes and aspect ratios.[14]

YOLO uses the sum-squared error between the predictions and the ground truth to calculate the loss. The loss function composes of:

- The classification loss
- The localization loss (errors between the predicted boundary box and the ground truth).
- The confidence loss (the objectness of the box).
3.7.1 Classification Loss

If an object is detected, the classification loss at each cell is the squared error of the class conditional probabilities for each class:

\[
S^2 \sum_{i=0}^{S^2} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2
\]

Where,

\[
\hat{p}_i(c)
\]

denotes the conditional class probability for class c in cell i

“\(\mathbb{1}_{\text{obj}}\) = 1 if an object appears in cell i, otherwise 0

3.7.2 Localization Loss

The localization loss measures the errors in the predicted boundary box locations and sizes. We only count the box responsible for detecting the object.

\[
\lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{\text{obj}} (\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2
\]

Where “\(\mathbb{1}_{\text{obj}}\) = 1, if an object is present in grid cell ith and the jth bounding box predictor is “responsible” for that prediction, otherwise 0.

\(\lambda_{\text{coord}}\) increase the weight for the loss in the boundary box coordinates.

YOLO predicts the square root of the bounding box width (w) and height (h) instead of the width and height. Besides, to put more emphasis on the boundary box accuracy, we multiply the loss by \(\lambda_{\text{coord}}\). Default value \(\lambda_{\text{coord}}\) is 5.
### 3.7.3 Confidence Loss

The confidence loss is given as below:

\[
\sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{obj} (C_i - \hat{C}_i)^2 + \lambda_{noobj} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{noobj} (C_i - \hat{C}_i)^2
\]

Where \(\mathbb{1}_{obj} = 1\), If an object is present in grid cell \(i\)th and the \(j\)th bounding box predictor is “responsible” for that prediction, otherwise 0. \(\mathbb{1}_{noobj}\) is the complement of \(\mathbb{1}_{obj}\). \(\lambda_{noobj}\) weights down the loss when detecting background.

The first part of the confidence loss function is if an object is detected in the box, (measuring the objectness of the box). And the other part is if an object is not detected in the box. Most boxes do not contain any objects. This causes a class imbalance problem, i.e. we train the model to detect background more frequently than detecting objects. To remedy this, we weight this loss down by a factor \(\lambda_{noobj}\). The default value for \(\lambda_{noobj}\) is assigned as 0.5.

### 3.7.4 Total Loss

The final loss adds localization, confidence, and classification losses together.

\[
\lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{obj} \left[ (x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right] \\
+ \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{obj} \left[ (\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2 \right] \\
+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{obj} (C_i - \hat{C}_i)^2 \\
+ \lambda_{noobj} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{noobj} (C_i - \hat{C}_i)^2 \\
+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{noobj} \sum_{c \in \text{classes}} (\hat{p}_i(c) - \hat{\hat{p}}_i(c))^2
\]
3.8 Non-maximal suppression

The objects in the image can be of different sizes and shapes, and to capture each of these perfectly, the object detection algorithms create multiple bounding boxes. (left image). Ideally, for each object in the image, we must have a single bounding box. The purpose of non-max suppression is to select the best bounding box for an object and reject or “suppress” all other bounding boxes. The Non-maximal suppression takes two things into account. [15]

- The objectiveness score is given by the model.
- The overlap or IOU of the bounding boxes.

The non-max suppression will first select the bounding box with the highest objectiveness score. And then remove all the other boxes with high overlap. This step gets rid of anomalous detections of objects. However, even after such filtering, we end up with many boxes for each object detected. But we only need one box. This bounding box is calculated using Non-max suppression. Non-max suppression makes use of a concept called “intersection over union” or IOU. It takes as input two boxes and calculates the ratio of the intersection and union of the two boxes.
Chapter 4
Implementation

4.1 Dataset

ICDAR 2015 is used as a dataset for this project. The data folder consists of images that contain 461 image files and the ground truth that contains information of that 461 images. The link to the data is https://drive.google.com/file/d/1ObrV9pbH_-LBGblodWgB6W4dtQIoTTH6/view [16]

4.2 Preprocessing

Jupyter Notebook is used to implement the code. Firstly, we import the required python libraries like Keras, matplotlib, sklearn, TensorFlow, etc.

```python
In [1]: import matplotlib.pyplot as plt
   ...: import os
In [2]: import numpy as np
In [3]: import cv2
   ...: from sklearn.model_selection import train_test_split
In [4]: from keras import backend as K
   ...: from keras.callbacks import ModelCheckpoint
   ...: from keras.optimizers import Adam
   ...: from keras.layers import *
   ...: from keras.applications import MobileNetV2
   ...: import tensorflow as tf
   ...: from keras.models import Model
   ...: from keras.models import model_from_json
```

Now, we declare variables to load the data and assign the path of that data.
In the following cell, we initialize image variables grid height, grid width, image width, and image height.

```python
In [4]:
X_final = []
Y_final = []
grid_h = 16
grid_w = 16
img_w = 512
img_h = 512
```

Now, the images in the dataset consist of different sizes and noisy images. So, the below code resizes the images to 512 x 512 dimensions. The ground truth coordinates are processed to form a matrix of dimensions as (grid height, grid width, 1, 5).

```python
In [5]:
for i in range(len(train_image_paths[2])):
    new_file = train_image_paths[2][i]
    x = cv2.read(train_image_paths[2][i])
    x_h, x_w = x.shape[0], x.shape[1]
    img_resized = cv2.resize(x, (512, 512))
    X_final.append(x)
    Y_final.append(y)

for i in data:
    file_names = file_data.split('.')[0]
    x_h, x_w = x.shape[0], x.shape[1]
    x = np.zeros((x_h, x_w, 1, 5))
    for j in range(len(x)):
        y = y + x[j, y]
        y = y + x[j, y]
        y = y + x[j, y]
        y = y + x[j, y]
        y = y + x[j, y]
        y = y + x[j, y]
        y = y + x[j, y]
        y = y + x[j, y]
        y = y + x[j, y]
        y = y + x[j, y]
```

20
The coordinate of the images is found in the above code is stored as a NumPy array.

4.3 The Model

Now, we create the model taking the image as an input which is an array. Using the “MobileNetV2” pre-trained model, we create the model. The convolution neural network is built with 3 x 3 kernel is used. ReLU and Leaky ReLU are used as an activation function. Adam optimizer is used to update weights based on training data.

We define loss function to the model. As I stated earlier the default value $\lambda_{\text{coord}} (l_{\text{coords}})$ is 5 and the $\lambda_{\text{noob}} (l_{\text{noob}})$ is 0.5. With the help of the true and predicted width and height values we find the classification loss, localization loss, confidence loss and the total loss. The loss function is given as below:

Classification Loss:

\[
\text{p_loss_absent} = \text{K.sum(K.square(p_pred - p_true)*noobs)}
\]
\[
\text{p_loss_present} = \text{K.sum(K.square(p_pred - p_true))}
\]

Localization Loss:

\[
\text{w_loss} = \text{K.sum(K.square(K.sqrt(w_pred) - K.sqrt(w_true)))*coords}
\]
\[
\text{h_loss} = \text{K.sum(K.square(K.sqrt(h_pred) - K.sqrt(h_true)))*coords}
\]
wh_loss = w_loss + h_loss

Confidence Loss:

x_loss = K.sum(K.square(x_pred - x_true)*coords)

yy_loss = K.sum(K.square(yy_pred - yy_true)*coords)

xy_loss = x_loss + yy_loss

Total Loss:

total_loss = p_loss_absent + p_loss_present + xy_loss + wh_loss

Now, we load the weights, model, and train the data.
Finally, we predict the bounding boxes around the text in the image using the model we defined and Intersection over Union (IOU).
Chapter 5

Conclusion

5.1 Accomplishments and lessons learned

An efficient model is developed in this research work that generates the text that presents in the given image. As part of this project research, I developed a deep understanding of object detection and requirements for creating an object detection model from an image. Identification of objects is achieved through YOLO. The implementation of a YOLO model is much more recommended because it is faster and the complexity is also less compared to the other models. By incorporating proper loss functions, the detector can predict either rotated rectangles or quadrangles for text regions, depending on specific applications. The majority challenge is to improve the speed and accuracy of text detection.

5.2 Limitation

Since our model learns to predict bounding boxes from data, it struggles to generalize to objects in new or unusual aspect ratios or configurations. Our model also uses relatively features for predicting bounding boxes since our architecture has multiple downsampling layers from the input image. Finally, while we train on a loss function that approximates detection performance, our loss function treats error the same in small bounding boxes versus large bounding boxes. A small error in a large box is generally negligible but a small error in a small box has a much greater effect on IOU.

5.3 Future work

This project is just a demo of what artificial intelligence has to do in terms of machine learning and deep learning, and what we can provide the machines to perform better in terms of artificial intelligence. This project can be extended by not only detecting the text but also recognizing the text from the image and video. The algorithms proposed by several large organizations have allowed not just to detect text but, also extract it. This lead to having a lot of research in this area recently by tech leads such as Google and Facebook. Industrial-strength, high-accuracy, real-time text detection and recognition system are incredibly difficult to achieve. Possible directions for future research include adapting the geometry formulation to allow direct detection of curved text, integrating the detector with a text recognizer, and extending the idea to general object detection. We are greedy to look for better methodologies to improve our work.
import numpy as np
import os
import tensorflow as tf
from scipy.io import loadmat
import cv2
import matplotlib.pyplot as plt

def decode_to_boxes(output, ht, wd):
    img_ht = ht
    img_wd = wd
    threshold = 0.5
    grid_h, grid_w = output.shape[:2]
    final_boxes = []
    scores = []
    for i in range(grid_h):
        for j in range(grid_w):
            if output[i, j, 0, 0] > threshold:
                temp = output[i, j, 0, 1:5]
                x_unit = ((j + (temp[0]))/grid_w)*img_wd
                y_unit = ((i + (temp[1]))/grid_h)*img_ht
                width = temp[2]*img_wd*1.3
                height = temp[3]*img_ht*1.3

                final_boxes.append([x_unit - width/2, y_unit - height/2, x_unit + width/2, y_unit + height/2])
scores.append(output[i,j,0,0])
return final_boxes,scores

def iou(box1,box2):
    x1 = max(box1[0],box2[0])
    x2 = min(box1[2],box2[2])
    y1 = max(box1[1],box2[1])
    y2 = min(box1[3],box2[3])
    inter = (x2 - x1)*(y2 - y1)
    fin_area = area1 + area2 - inter
    iou = inter/fin_area
    return iou

def non_max(boxes , scores , iou_num):
    scores_sort = scores.argsort().tolist()
    keep = []
    while(len(scores_sort)):
        index = scores_sort.pop()
        keep.append(index)
        if(len(scores_sort) == 0):
            break

    iou_res = []
    for i in scores_sort:
        iou_res.append(iou(boxes[index] , boxes[i]))
    iou_res = np.array(iou_res)
filtered_indexes = set((iou_res > iou_num).nonzero()[0])

scores_sort = [v for (i,v) in enumerate(scores_sort) if i not in filtered_indexes]

final = []
for i in keep:
    final.append(boxes[i])
return final

def decode(output, ht, wd, iou):
    boxes, scores = decode_to_boxes(output, ht, wd)
    boxes = non_max(boxes, np.array(scores), iou)
    return boxes

Preprocessing.ipynb

import numpy as np
import pandas as pd
import cv2
import os
import tqdm
from scipy.io import loadmat
import matplotlib.pyplot as plt
image_path = 'C:\Users\godap\Text-detection\Data\images\'

for new_file in tqdm.tqdm(os.listdir(gt_path)):
    name_split = new_file.split('.
    image_name = name_split[0][3:]
    image_name = image_name + '.jpg'
if 'gt' in new_file:
    image_name = name_split[0][3:]
    image_name = image_name + '.jpg'
    path_img = os.path.join(image_path, image_name)
    train_image_paths.append(path_img)
    train_gt_paths.append(os.path.join(gt_path, new_file))

X_final = []
Y_final = []
grid_h = 16
grid_w = 16
img_w = 512
img_h = 512
for z in tqdm.tqdm(range(len(train_image_paths))):
    new_file = train_image_paths[z]
    x = cv2.imread(train_image_paths[z])
    x_sl = 512/x.shape[1]
    y_sl = 512/x.shape[0]
    img = cv2.resize(x,(512,512))
    X_final.append(img)
    i = " "

if 'img' in new_file:
    i = ", "
Y = np.zeros((grid_h,grid_w,1,5))
file = train_gt_paths[z]
name = open(file, 'r')
data = name.read()
data = data.split("\n")
data = data[:-1]

for li in data:
    temp_list = []
    file_data = li.split(i)
    s = file_data[4]
    bb = file_data[:4]
    x = int(bb[0]) * x_sl
    x = int(bb[2]) * x_sl
    y = int(bb[1]) * y_sl
    y = int(bb[3]) * y_sl
    w = (xmax - xmin)/img_w
    h = (ymax - ymin)/img_h
    x = ((xmax + xmin)/2)/img_w
    y = ((ymax + ymin)/2)/img_h
    x = x * grid_w
    y = y * grid_h
    Y[int(y),int(x),0,0] = 1
    Y[int(y),int(x),0,1] = x - int(x)
    Y[int(y),int(x),0,2] = y - int(y)
    Y[int(y),int(x),0,3] = w
    Y[int(y),int(x),0,4] = h
    Y_final.append(Y)

X = np.array(X_final)
X_final = []
Y = np.array(Y_final)
Y_final = []
X = (X - 127.5)/127.5
np.save('C:\Users\godap\Text-detection\Data\X.npy',X)
Model.ipynb

import keras
import cv2
from Utils import *
from keras import backend as K
from sklearn.model_selection import train_test_split
from keras.callbacks import ModelCheckpoint
from keras.optimizers import Adagrad
from keras.layers import *
from keras.applications import MobileNetV2
import tensorflow as tf
import numpy as np
from keras.models import Model
from keras.models import model_from_json
import matplotlib.pyplot as plt
import os

X = np.load('C:\\Users\\godap\\Text-detection\\Data\\X.npy')
Y = np.load('C:\\Users\\godap\\Text-detection\\Data\\Y.npy')
print(X.shape, Y.shape)

X_train, X_val, Y_train, Y_val = train_test_split(X, Y, train_size=0.75, shuffle=True)
X = []
Y = []

def save_model(model):
    model_json = model.to_json()
    with open('C:\\Users\\godap\\Text-detection\\model\\text_detect_model.json', "w") as json_file:
json_file.write(model_json)

def load_model(strr):
    json_file = open(strr, 'r')
    loaded_model_json = json_file.read()
    json_file.close()
    loaded_model = model_from_json(loaded_model_json)
    return loaded_model

def yolo_model(input_shape):
    inp = Input(input_shape)

    model = MobileNetV2(input_tensor=inp, include_top=False, weights='imagenet')
    last_layer = model.output

    conv = Conv2D(512, (3,3), activation='relu', padding='same')(last_layer)
    conv = Dropout(0.4)(conv)
    bn = BatchNormalization()(conv)
    lr = LeakyReLU(alpha=0.1)(bn)
    conv = Conv2D(128, (3,3), activation='relu', padding='same')(lr)
    conv = Dropout(0.4)(conv)
    bn = BatchNormalization()(conv)
    lr = LeakyReLU(alpha=0.1)(bn)
    conv = Conv2D(5, (3,3), activation='relu', padding='same')(lr)
    final = Reshape((grid_h, grid_w, classes, info))(conv)
    model = Model(inp, final)
    return model

#optimizer

opt = Adam(lr=0.0001, beta_1=0.9, beta_2=0.999, epsilon=1e-08, decay=0.0)

#checkpoint

checkpoint = ModelCheckpoint('C:\Users\godap\Text-detection\text_detect.h5',
monitor='val_loss', verbose=1, save_best_only=True, mode='min', period=1)
def yolo_loss_func(y_true, y_pred):
    # y_true : 16,16,1,5
    # y_pred : 16,16,1,5
    l_coords = 5.0
    l_noob = 0.5
    coords = y_true[:, :, :, :, 0] * l_coords
    noobs = (-1 * (y_true[:, :, :, :, 0] - 1) * l_noob)
    p_pred = y_pred[:, :, :, :, 0]
    p_true = y_true[:, :, :, :, 0]
    x_true = y_true[:, :, :, :, 1]
    x_pred = y_pred[:, :, :, :, 1]
    yy_true = y_true[:, :, :, :, 2]
    yy_pred = y_pred[:, :, :, :, 2]
    w_true = y_true[:, :, :, :, 3]
    w_pred = y_pred[:, :, :, :, 3]
    h_true = y_true[:, :, :, :, 4]
    h_pred = y_pred[:, :, :, :, 4]
    p_loss_absent = K.sum(K.square(p_pred - p_true) * noobs)
    p_loss_present = K.sum(K.square(p_pred - p_true))
    x_loss = K.sum(K.square(x_pred - x_true) * coords)
    yy_loss = K.sum(K.square(yy_pred - yy_true) * coords)
    xy_loss = x_loss + yy_loss
    w_loss = K.sum(K.square(K.sqrt(w_pred) - K.sqrt(w_true)) * coords)
    h_loss = K.sum(K.square(K.sqrt(h_pred) - K.sqrt(h_true)) * coords)
    wh_loss = w_loss + h_loss
    loss = p_loss_absent + p_loss_present + xy_loss + wh_loss
    return loss

#load and save model
input_size = (img_h, img_w, channels)
model = yolo_model(input_size)
print(model.summary())
save_model(model)
model.load_weights('C:\\Users\\godap\\Text-detection\\text_detect.h5')
model.compile(loss=yolo_loss_func, optimizer=opt, metrics=['accuracy'])
hist = model.fit(X_train, Y_train, epochs=30, batch_size=4, callbacks=[checkpoint], validation_data=(X_val, Y_val))
model = load_model('C:\\Users\\godap\\Text-Detection\\model\\text_detect_model.json')
model.load_weights('C:\\Users\\godap\\Text-detection\\text_detect.h5')

def predict_func(model, inp, iou, name):
    ans = model.predict(inp)
    boxes = decode(ans[0], img_w, img_h, iou)
    img = ((inp + 1)/2)
    img = img[0]
    for i in boxes:
        i = [int(x) for x in i]
        img = cv2.rectangle(img, (i[0], i[1]), (i[2], i[3]), color=(0, 255, 0), thickness=2)
    plt.imshow(img)
    plt.show()
    cv2.imwrite(os.path.join('Results', str(name) + '.jpg'), img*255.0)

rand = np.random.randint(0, X_val.shape[0], size = 5)
for i in rand:
    predict_func(model, X_val[i:i+1], 0.5, i)

for i in os.listdir('Test'):
    img = cv2.imread(os.path.join('Test', i))
img = cv2.resize(img,(512,512))

img = (img - 127.5)/127.5

predict_func(model , np.expand_dims(img,axis= 0) , 0.5 , 'sample')
References

[14] https://medium.com/@amrokamal_47691/yolo-yolov2-and-yolov3-all-you-want-to-know-7e3e92dc4899

Figure references

[2.1] https://machinelearningmastery.com/what-is-computer-vision/
[2.2] https://www.fritz.ai/object-detection/
[2.7] https://www.oreilly.com/library/view/reinforcement-learning-with/9781788835725/786aae81-77a7-437e-9a75-64925d7940ca.xhtml
[3.1] https://towardsdatascience.com/yolo-you-only-look-once-3dbdbb608ec4
[3.2] https://hackernoon.com/understanding-yolo-f5a74bbc7967
[3.3] https://hackernoon.com/understanding-yolo-f5a74bbc7967
[3.4] https://hackernoon.com/understanding-yolo-f5a74bbc7967
[3.5] https://towardsdatascience.com/yolo-you-only-look-once-3dbbb608ec4