使用Tensorflow識別車牌

由Matthew Earl (mailto:blog@matthewearl.com) 在2016年5月6日創建 。 在reddit上進行討論！（111分/ 17評論） (https://www.reddit.com/r/programming/comments/4i4j5x/how_i_wrote_an_automatic_license_plate/)

介紹

在過去的幾周中，我一直在研究深度學習，尤其是卷積神經網絡 (https:// en.wikipedia.org/wiki/Convolutional_neural_network)。最近一次傑出的論文是Google的“ 街景視圖中 (http:// static.googleusercontent.com/media/research.google.com/en//pubs/archive/ 42241.pdf)的 多位數數字識別” (http:// static.googleusercontent.com/media/research.google.com/en//pubs/archive/ 42241.pdf)。本文介紹了一種使用單個端到端神經網絡從街景圖像中提取門牌號的系統。然後，作者繼續說明如何將同一網絡應用於以人為層面的準確性破壞Google自己的CAPTCHA系統。

為了獲得實踐神經網絡的實踐經驗，我決定設計一種系統來解決類似的問題：自動車牌識別（如果您在美國，則可以自動車牌識別）。我這樣做的原因有三點：

- 我應該能夠使用與Google論文相同（或相似）的網絡架構：谷歌架構在解決驗證碼方面表現出同樣出色的表現，因此可以合理地假設它在讀取車牌上也表現良好。在我學習CNN的過程中，擁有一個已知的良好網絡體系結構將大大簡化事情。
- 我可以輕鬆生成培訓數據。訓練神經網絡的主要問題之一是需要大量標記的訓練數據。為了正確地訓練網絡，通常需要成千上萬個帶有標籤的訓練圖像。幸運的是，英國車牌的相關統一性意味著我可以綜合訓練數據。
- 好奇心。傳統的ANPR系統依靠 (https://en.wikipedia.org/wiki/ Automatic_number_plate_recognition#Algorithms)手寫算法進行印版定位，歸一化，分割，字符識別等。因此，這些系統往往需要成千上萬行。有趣的是，我能用最少的特定領域知識以較少的代碼量開發出一個出色的系統。

對於這個項目，我使用了Python，TensorFlow (https://www.tensorflow.org/)， OpenCV (http://opencv.org/)和NumPy (http://www.numpy.org/)。源代碼 可在此處獲得 (https://github.com/matthewearl/deep-anpr)。

輸入，輸出和窗口

為了簡化生成訓練圖像並減少計算需求，我決定我的網絡將在128x64灰度輸入圖像上運行。

之所以選擇128x64作為輸入分辨率，是因為它足夠小，可以使用適度的資源在合理的時間內進行訓練，但又足夠大，可以使車牌更加可讀：

圖片信用

為了在更大的圖像中檢測車牌號，在各種比例下使用滑動窗口方法：

圖片信用

右側的圖像是神經網絡看到的128x64輸入，而左側則顯示原始輸入圖像上下文中的窗口。

對於每個窗口，網絡應輸出：

- 在輸入圖像中出現車牌的概率。（在上面的動畫中顯示為綠色框）。
- 數字在每個位置的概率，即 對於7個可能位置中的每個位置，它應該返回36個可能字符之間的概率分佈。（對於該項目，我假設車牌號正好包含7個字符，大多數英國車牌號就是這種情況）。

當且僅在以下情況下，才認為板存在：

- 印版完全落在圖像範圍內。
- 印版的寬度小於圖像寬度的80％，印版的高度小於圖像高度的87.5％。
- 印版的寬度大於圖像寬度的60％，或者印版的高度大於圖像高度的60％。

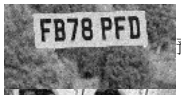
使用這些數字，我們可以使用一次滑動窗口，該窗口一次移動8個像素，並放大 $\sqrt{}$ 在兩次縮放級別之間切換，並確保不會錯過任何板塊，同時不會對任何單個板塊產生過多的匹配項。確實發生的所有重複項都將在後處理步驟中合併（稍後說明）。

合成圖像

為了訓練任何神經網絡，必須提供一組訓練數據以及正確的輸出。在這種情況下，這將是一組128x64圖像以及預期的輸出。這是為此項目生成的培訓數據的說明性示例：

- 

預期產出 `HH41RFP 1` 。

- 

預期產出 `FB78PFD 1` 。
- 

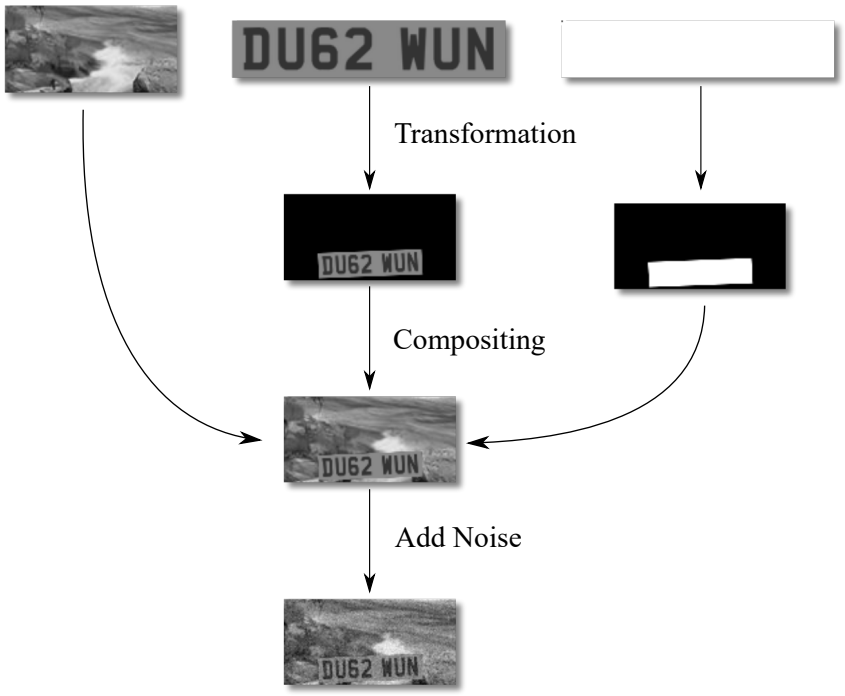
預期產出 `JW01GAI 0` 。（板被部分截斷。）
- 

預期產出 `AM46KVG 0` 。（盤子太小。）
- 

預期產出 `XG86KIO 0` 。（盤子太大。）
- 
預期產出 `XH07NYO 0` 。（完全沒有盤子。）

預期輸出的第一部分是網絡應輸出的數字。第二部分是網絡應輸出的“存在”值。對於標記為不存在的數據，我在括號中包含了解釋。

生成圖像的過程如下所示：



文本和印版顏色是隨機選擇的，但是文本必須比印版暗一些。這是為了模擬現實世界的照明變化。最後添加了噪聲，不僅考慮了實際的傳感器噪聲，而且還避免了網絡過於依賴清晰定義的邊緣（如在散焦輸入圖像中看到的那樣）。

具有背景很重要，因為這意味著網絡必須學會識別車牌的範圍而不會“作弊”：例如，如果使用黑色背景，則網絡可能會學會基於非黑色來識別車牌位置，這顯然會不適用於汽車的真實圖片。

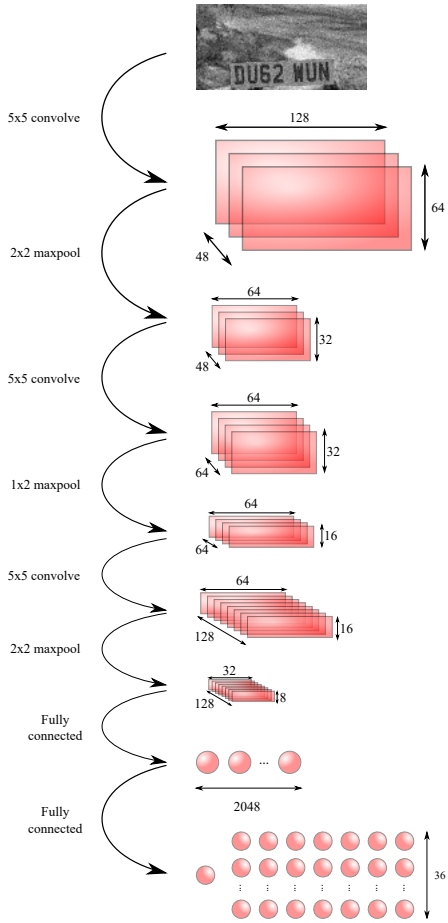
背景來自SUN數據庫 (http:// vision.cs.princeton.edu/projects/2010/SUN/)，該數據庫 (http:// vision.cs.princeton.edu/projects/2010/SUN/)包含超過100,000張圖像。重要的是，圖像的數量很大，以避免網絡“記住”背景圖像。

應用於印版（及其蒙版）的變換是基於隨機滾動，俯仰，偏航，平移和縮放的仿射變換。根據可能會看到車牌的範圍來選擇每個參數允許的範圍。例如，偏航允許的變化比側傾變化大得多（您更可能看到汽車轉彎而不是側彎）。

生成圖像的代碼相對較短（約300行）。可以在gen.py中 (https://github.com/matthewearl/deep-anpr/blob/master/gen.py)讀取。

網絡

這是使用的網絡架構：



有關CNN構建基塊的摘要，請參見維基百科頁面 (https://en.wikipedia.org/wiki/ Convolutional_neural_network)。上面的網絡實際上是基於Stark等人的這篇論文 (https://vision.in.tum.de/_media/ spezial/bib/stark-gcpr15.pdf)的，因為它比Google的論文提供了有關所使用架構的更多細節。

輸出層有一個節點（如左圖所示）用作狀態指示器。其餘的編碼特定號碼牌的概率：如圖所示，每一列對應於號碼牌中的數字之一，每個節點給出存在相應字符的概率。例如，第2列第3行中的節點給出第二個數字是a的概率 `c` 。

按照深度神經網絡的標準，除了輸出層以外，其他所有層都使用ReLU激活 (https://en.wikipedia.org/wiki/Rectifier_(neural_networks))。存在節點具有S型激活，通常用於二進制輸出。其他輸出節點跨字符使用softmax（即，使每一列中的概率加起來為1），這是對離散概率分佈進行建模的標準方法。

定義網絡的代碼在model.py中 (https://github.com/matthewearl/ deep-anpr/blob/master/model.py)。

損失函數是根據標籤和網絡輸出之間的交叉熵來定義的。為了獲得數值穩定性，使用 `softmax_cross_entropy_with_logits` (https://www.tensorflow.org/versions/ r0.8/api_docs/python/nn.html#softmax_cross_entropy_with_logits)和 將最終層的激活函數匯總到交叉熵計算中 `sigmoid_cross_entropy_with_logits` (https://www.tensorflow.org/versions/ r0.8/api_docs/python/nn.html#sigmoid_cross_entropy_with_logits)。有關交叉熵的詳細，直觀的介紹，請參閱 Michael A. Nielsen的免費在線書籍中的 (http:// neuralnetworksanddeeplearning.com/)本節 (http:// neuralnetworksanddeeplearning.com/chap3.html#the_cross-entropy_cost_function)。 (http:// neuralnetworksanddeeplearning.com/)

使用nVidia GTX 970進行培訓（train.py (https://github.com/matthewearl/deep-anpr/blob/master/ train.py)）大約需要6個小時，而培訓數據是由CPU上的後台進程即時生成的。

輸出處理

為了真正地檢測和識別輸入圖像中的車牌號，如窗口部分所述，將類似於上述的網絡應用於各種位置和比例的128x64窗口。

該網絡與訓練中使用的網絡不同，後兩層是卷積的而不是完全連接的，並且輸入圖像可以是任何大小，而不是128x64。這個想法是，可以將特定比例的整個圖像饋入該網絡，從而生成一個在每個“像素”處具有存在/字符概率值的圖像。這裡的想法是相鄰的窗口將共享許多卷積特徵，因此將它們捲入同一網絡可避免多次計算相同的特徵。

可視化輸出的“狀態”部分會產生以下內容：

圖片信用

此處的框是網絡檢測到大於99%的概率存在車牌的區域。高閾值的原因是要考慮到訓練中引入的偏差：大約一半的訓練圖像包含車牌，而在現實世界中，汽車的牌照圖像要少得多。因此，如果使用50%閾值，則檢測器容易出現誤報。

為了應對明顯的重複，我們對輸出應用了一種非最大抑制形式：

圖片信用

這裡使用的技術首先將矩形分組為重疊的矩形，並為每組輸出：

- 所有邊界框的交集。
- 與出現概率最高的組中的框相對應的許可證號。

這是應用於本文頂部圖像的檢測器：

圖片信用

Whoops, the *R* has been misread as a *P*. Here's the window from the above image which gives the maximum presence response:

Image credit

On first glance it appears that this should be an easy case for the detector, however it turns out to be an instance of overfitting. Here's the *R* from the number plate font used to generate the training images:

Note how the leg of the *R* is at a different angle to the leg of the *R* in the input image. The network has only ever seen *R*'s as shown above, so gets confused when it sees *R*'s in a different font. To test this hypothesis I modified the image in GIMP to more closely resemble the training font:

And sure enough, the detector now gets the correct result:

The code for the detector is in detect.py (https://github.com/matthewearl/ deep-anpr/blob/master/detect.py).

Conclusion

I've shown that with a relatively short amount of code (~800 lines), its possible to build an ANPR system without importing any domain-specific libraries, and with very little domain-specific knowledge. Furthermore I've side-stepped the problem of needing thousands of training images (as is usually the case with deep neural networks) by synthesizing images on the fly.

On the other hand, my system has a number of drawbacks:

1. It only works with number plates in a specific format. More specifically, the network architecture assumes exactly 7 chars are visible in the output.
2. It only works on specific number plate fonts.
3. It's *slow*. The system takes several seconds to run on moderately sized image.

The Google team solves 1) by splitting the higher levels of their network into different sub-networks, each one assuming a different number of digits in the output. A parallel sub-network then decides how many digits are present. I suspect this approach would work here, however I've not implemented it for this project.

I showed an instance of 2) above, with the misdetection of an *R* due to a slightly varied font. The effects would be further exacerbated if I were trying to detect US number plates rather than UK number plates which have much more varied fonts. One possible solution would be to make my training data more varied by drawing from a selection of fonts, although it's not clear how many fonts I would need for this approach to be successful.

The slowness (3)) is a killer for many applications: A modestly sized input image takes a few seconds to process on a reasonably powerful GPU. I don't think its possible to get away from this without introducing a (cascade of) detection stages, for example a Haar cascade (https://en.wikipedia.org/ wiki/Viola%E2%80%93Jones_object_detection_framework), a HOG detector (https:// en.wikipedia.org/wiki/Histogram_of_oriented_gradients), or a simpler neural net.

It would be an interesting exercise to see how other ML techniques compare, in particular pose regression (http://vision.ucsd.edu/~pdollar/files/papers/ DollarCVPR10pose.pdf) (with the pose being an affine transformation corresponding with 3 corners of the plate) looks promising. A much more basic classification stage could then be tacked on the end. This solution should be similarly terse if an ML library such as scikit-learn (http:// scikit-learn.org/) is used.

In conclusion, I've shown that a single CNN (with some filtering) *can* be used as a passable number plate detector / recognizer, however it does not yet compete with the traditional hand-crafted (but more verbose) pipelines in terms of performance.

Image Credits

(https://commons.wikimedia.org/wiki/ File:Proton_Saga_EV_at_the_RAC_Future_Car_Challenge_2011,_U.K.jpg)Somaditya Bandyopadhyay的 (http://www.flickr.com/people/16836099@N08/)原始" Proton Saga EV"圖像 (https://commons.wikimedia.org/wiki/ File:Proton_Saga_EV_at_the_RAC_Future_Car_Challenge_2011,_U.K.jpg)是 在 Creative Commons Attribution-Share Alike 2.0 Generic許可下獲得許可的 (https:// creativecommons.org/licenses/by-sa/2.0/deed.en)。

(https://commons.wikimedia.org/wiki/ File:Google_Street_View_Car_near_Howden,_UK_2.JPG)Reedy (https:// commons.wikimedia.org/wiki/User:Reedy)原始的" Google Street View Car"圖片 (https://commons.wikimedia.org/wiki/ File:Google_Street_View_Car_near_Howden,_UK_2.JPG)是根據 知識共享署名-相同方式分享 3.0未遷移許可證獲得許可的 (https:// creativecommons.org/licenses/by-sa/3.0/deed.en)。

討論reddit！（111分/ 17評論）
(https://www.reddit.com/r/programming/comments/4i4j5x/how_i_wrote_an_automatic_license_plate/)

←較舊 (/2016/01/22/portrait-timelapse/) 博客索引 (/) 較新→ (/2018/06/28/smb-level-extractor/)