Machine Learning Engineer Nanodegree

Detecting if there's a car in an image or not (Binary image classification)

Laith Al Rachwani

December 25th, 2017

I. Definition

Project Overview

Image Processing has been a big trend recently, especially in the world of machine learning, where we can use pictures to extract important features and data from them. Perhaps a good example on image processing is the new facial recognition feature apple implemented in its latest iPhone X, where you can unlock your phone using the facial detection system integrated within the phone, this is done through image recognition using Machine learning approaches such as Deep learning, you can find the article here which talks about apple using machine learning approaches in order to implement these features, you can read more about it here: https://www.wired.com/story/apples-neural-engine-infuses-the-iphone-with-ai-smarts/

Another possible example on image processing is google's "Search by image feature" where you can provide google a picture and based on that picture provided google will give you similar pictures, i could not find an article that says this is done through image processing, however i think such an important feature like this can be implemented through image processing and machine learning approaches.

And during studying image processing i realized that we can use image processing for many other incredible things such as extracting important features from pictures and even image classification, were we can classify images based on its contents, for example we may have pictures of various animals, we can classify each picture to the animal it has, so if the picture has a cat in it it will be classified under that "cats" class, and if it has a dog it will be classified under the "dogs" class and so on. all of this is done

using machine learning techniques and a very effective approach for this is deep learning.

What I will be doing in this project is classifying an image into either an image that has a car in it or an image that does not have a car, where we are going to to be provided with an image and using our algorithm we built, we are going to predict the probability that there's a vehicle in the image provided, this is called binary image classification, where we only have two classes, a "cars" class which contains pictures that have a car and a "non-cars" class which contains pictures that do not have a car in them.

al

Problem Statement

As mentioned above, the problem we are trying to solve is a binary image classification problem, where we have labeled data from two classes, and we would like to train a model using this labeled data so that our model can learn what a picture with a car is like and what a picture that does not have a car is like, so that in the future when we input to our model a picture that it has never seen before, the model will predict the label for that picture, the prediction will be a probability which is going to be to turn a probability of the chance that there's a car in the picture, so after training our model we could feed it pictures to predict the class of that picture and we get is a percentage (the probability multiplied by 100) that there's a car in this picture.

The approach i will be using in order to solve this problem is a deep learning approach, where i will be building a deep CNN that will be trained using our labeled data. After training as mentioned above, we can feed our CNN pictures and our CNN will predict to which class it belongs.

Metrics

The metric i will be using for this project is the accuracy metric. The accuracy in general means that how many predictions that we did were correct out of all of the predictions we made. Here by accuracy, we are going to see if our CNN actually

classifies an image into its correct class. So out of all the classified images, how many of them were actually classified correctly.

The reason behind that choice is that accuracy is a good indicator of how good the network is performing during the training, and it's a good metric for such a binary image classification problem.

II. Analysis

Data Exploration

The datasets I will be using for this project are two datasets, the first dataset I will be using is provided by stanford.edu which can found through this link: http://ai.stanford.edu/~jkrause/cars/car_dataset.html

The dataset contains pictures of various car pictures which are taken from many different angles, I will be using each picture in the dataset as an example to teach my model (which is a CNN in this case) how a car looks like, here are some examples of pictures taken from the dataset, which are pictures varying in size, the size of pictures in this dataset vary in size, I will resize each picture to a size of 150x150

another dataset I will be using is a dataset called '**The LabelMe-12-50k dataset'** which can be found through this link: http://www.ais.uni-bonn.de/download/datasets.html.

This dataset contains random pictures from many classes, of course I am going to exclude any picture that falls under the car class because I will be using the pictures in this dataset as examples of pictures that don't have a car in them, the pictures in this dataset all have a size of 224x224 and will be resized to a size of 150x150, here are some examples of pictures taken from this datasets:


The number of training pictures I will using is 8979 pictures, around half of these pictures will be pictures that have a car in them and the other half will be the examples of pictures that don't have a car in them, and i will use 20% of 8979 pictures as validation data, so that we can see how well our CNN doing during training, so 1796 pictures will be for validation and the rest are going to be for training.

Exploratory Visualization

Since our datasets consist entirely of pictures, we are going to visualize how a picture is like, a picture is usually a grid and each cell within that grid has a value between 0 and 255 as the following:

The original picture

47	58	69	80	1	12	23	34	45
57	68	79	9	11	22	33	44	46
67	78	8	10	21	32	43	54	56
77	7	18	20	31	42	53	55	66
6	17	19	30	41	52	63	65	76
16	27	29	40	51	62	64	75	5
26	28	39	50	61	72	74	4	15
36	38	49	60	71	73	3	14	25
37	48	59	70	81	2	13	24	35

A small part of the picture's grid

We can see above our grid right next to the picture which is a small part from our picture (the original grid representing our picture would have more columns and rows), note how every pixel in the picture is saved as a number, and that each number is a value between 0 and 255, where the closer the number is to 0 means the color of that pixel is closer to black, and the closer the pixel to 255 means the color of that pixel is closer to white.

Algorithms and Techniques

As we have already mentioned, our model for this problem will be Convolutional Neural network, the CNN will be trained using our labeled data, the input for our CNN is a 150x150x1 image, our images by default will have three channels, for the red, green and the blue but we will be converting each picture to a grayscale image which means we will have only one channel and we will be resizing each picture to a size of 150x150 so the input will be a 150x150 image.

Another technique i used is the fit_generator() function to train our CNN, where i implemented a function that generates our data in real time and directly feeds it to our CNN, which will be better for our memory.

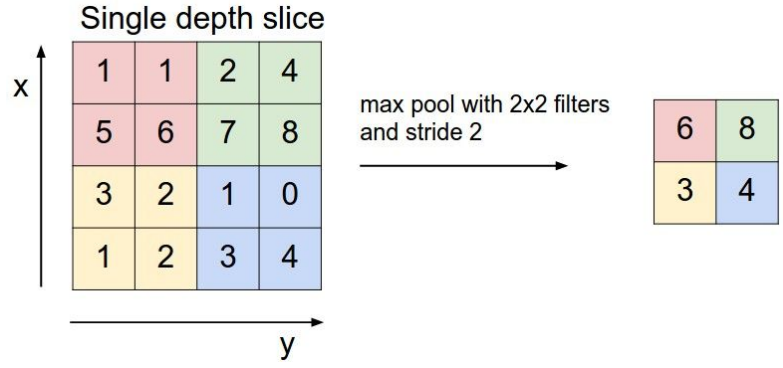
The output layer of our CNN will have a sigmoid activation function, which outputs a value between 0 and 1.

Let's get more in depth in our CNN, as mentioned in a section above, CNNs have proven to be extremely efficient when it comes to image processing, the Convolutional Neural Network consists of many components, such as Convolutional Layers, the first convolutional layer in my model has an input shape of 150x150x1 which is the size of

of each image we have in our datasets after we resized them to that size, in order to specify the size of the input we use the `input_shape` parameter to specify the shape of the input, another thing we have to define in our convolution layer is the size of the convolutional window, which is also sometimes referred to as the kernel, which is a 2D grid of a size usually around 2x2 or 3x3, that is going to slide across our input in order to find features or extract information from the input, we also define the number of these kernels within our convolutional layer, we can have tens or hundreds of these filters in a convolutional layer, in my model the number of filters in the first convolutional layer is 32, where we are going to have 32 filters, so our `filters` parameter is going to be set to 32 and each one of these filter is going to have a size of 3x3, each filter is responsible for extracting certain features or finding a certain pattern within the image. So for our first convolutional layer we have 32 filters each of size of 2x2.

please note that the input of each convolutional layer will be the output of its previous convolutional layer, in case the convolutional layer was not the first one in our model.

Another important component we used in our CNN is something called a pooling layer, which usually takes a convolutional layer as an input. we may need to use many number of filters in our convolutional layers in order to find more patterns within our inputs, so we may end up with a bigger number of filters which means a bigger number of parameters will need to be used which makes our CNN more prone to overfitting, pooling layers are used to solve this issue, in our model i used a Max Pooling layer which will also have a size of height and width just like our convolutional window, the max pooling window will slide through the input and pick the maximum value among the window of cells it's currently on, that will decrease the dimensions of our filters heavily which means less parameters which means our CNN becomes less prone to overfitting, perhaps some visualization would make things clearer:

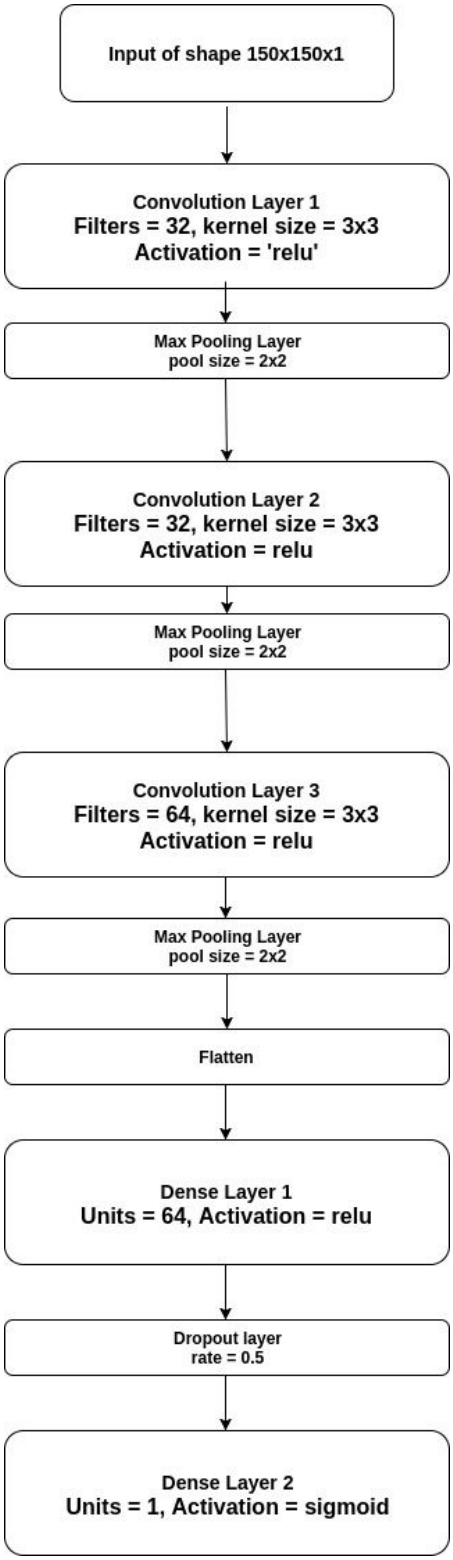


Notice in the picture above we have a max pooling window of size 2x2 and a stride of 2 (the stride means how many steps we will be taking in each step) which means we will

have a window of size 2x2 sliding through our input and in every 2x2 section it goes through, it picks the highest value, so in our example above, we went from having an input 4x4 to an input 2x2.

In our model, after 3 Convolutional layers (and of course after using a max pooling layer after each convolutional layer) we have a flattening layer which flattens its input to a vector, this vector will be passed through two fully connected layers, after the first fully connected layer i used a dropout layer using a rate of 0.5, what a dropout layer does is that it drops units randomly at a certain rate, in our a case a rate of 0.5 was applied. dropout layers help our CNN to generalize better and overfit less. after our dropout layer i added a dense layer which will only have one node with a ‘Sigmoid’ activation function, our final node will output a result between 0 and 1 indicating the probability of having a car within the image.

In the next page is a diagram i made to illustrate our model:



A diagram illustrating our model, the diagram was built using the online tool: https://www.draw.io/

Benchmark

I think GoogLeNet model is a good benchmark for my model, GoogLeNet has achieved incredible results when it comes to image processing, the accuracy reach around 95% for multiclass image processing, the accuracy of my model on the validation set has reach 95%, so i think my model is quite close to the benchmark mentioned, which means maybe with some more training data our model could reach an accuracy close to these powerful models.

	Top 1 accur.	Top 5 accur.	Final loss
Baseline SVM	0.031		55.48
Baseline ConvNet	0.062		42.73
CaffeNet fine-tuned	0.447		2.62
CaffeNet partial-train	0.418		2.61
CaffeNet full-train	0.417		3.21
CaffeNet scratch	0.005		5.31
GoogLeNet fine-tuned	0.774	0.943	1.25
GoogLeNet partial-train	0.775	0.943	1.28
GoogLeNet full-train	**0.800**	**0.951**	1.09
GoogLeNet scratch	0.347	0.636	6.42
VGGNet fine-tuned	0.789	0.942	**1.01**
VGGNet scratch	0.008	0.031	5.51

Some models and their accuracy

III. Methodology

Data Preprocessing

Preprocessing is always an important step to take before we start training our model, on the things i did was converting each picture from an RGB to a grayscale picture which reduces the input space quite a lot, since we convert from a 3-dimensional input space to a 2-dimensions.

Another important preprocessing step is normalizing our pictures, which means we convert each value in our grid to a value between 0 and 1, In general it's good practice to work with normalized input.

And finally we resized each picture to a size of 150x150 which heavily decreased the number of parameters in our model, since i downsized the size of the pictures, so i downsized the size of the input in our model.

Implementation

The approach we did here is a deep learning learning approach, where trained a CNN to learn what a car looks like and what a non car looks like, the CNN was implemented using the keras module, keras is a powerful tool when implementing CNN in python.

Our CNN process each picture it receives, the pictures are labeled with 0 or 1, 0 meaning there's no car in them and 1 means there is a car in them, our CNN through it's hidden layers learns how a car looks like and how a car does not look like, of course our CNN will consist of several hidden layers, each hidden layer will be responsible for detecting a certain feature within an image, for example the first layer might be responsible for detecting lines and simple shapes and the second layer is responsible for detecting more complex features and shapes within the image like maybe circles and more.

Our convolutional layers were then followed by a dense layer and a dropout layer, using a dropout layer will help the CNN generalize more and overfit less, so i followed my dense layer with a dropout layer at 0.5.

The output was only one neuron, since it's a binary classification problem and our labels were only 0 and 1 so one output was enough to output the score of the possibility if there's a car in this picture or not.

One of the complications i faced was the large number of parameters because i resized each picture to a size of 250x250 at first, but then by decreasing that size to 150x150 i was able to achieve a model with way less parameters that needed less computational power.

Also, when training my CNN i didn't fit the dataset directly to the CNN, i used the fit_generator() function, and using a function i implemented called gen_batches() i was able to feed my CNN in real time without having to worry about consuming so much memory during the training.

Refinement

One of the things that helped me overcome a problem was using the fit_generator() function keras provides for training my model, instead of just using my entire dataset directly and exhaust my memory, i was able to generate data in real time and feed it to my CNN directly which helped me during the training.

Another thing that helped my model was increasing the number of epochs for the training process, due to my limited computational power, i was trying to use a small number of epochs such as 3 epochs which yielded an accuracy of 0.89, however i increased the number of epochs to 20 and the accuracy increased all the way to approximately 0.95

IV. Results

Model Evaluation and Validation

The model i have chosen for this problem is a Convolutional Neural Network, CNN's have proven to be extremely powerful and reliable when it comes to image processing, the input of our model is a 150x150x1 image, which is a grayscale image, and as for the output it's a single floating number, indicating the probability whether than image fed into the CNN has a car in it or not.

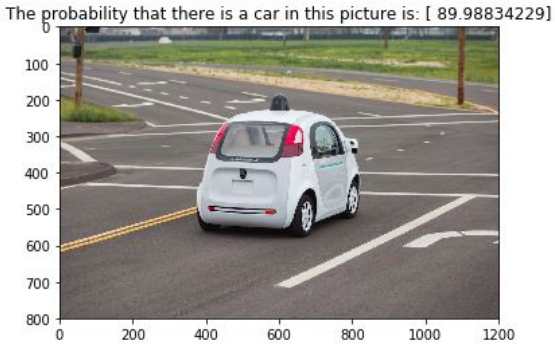
The model has been tested with various inputs indeed and it has yielded great results, i used random pictures from the internet has car in them and pictures that don't have a car and the results were pretty good, here is a look:

I tried using a picture of me as an input and it got a score of 1.17 out of 100 which is pretty good since the score indicates the possibility of having a car in the picture.

Another input i tried is a picture of a car, that's not present in our training set and has been taken from google search:

And the score here is 99.99 which is pretty good as well! Since it's a picture of a car and it has yielded a high score so our CNN is indeed efficient for such a problem, since these pictures are pictures that the CNN has never seen before and it has yielded such good results, i think it can be trusted as a good model for this problem.

Here's another example of a car that looks a little different from the regular car, yet the model still had a good accuracy when predicting it:



We got a score here of 89.9 which is pretty good, i also noticed that my training set does not have pictures of cars on the road like this, most of the pictures are professionally taken while this picture is just a regular picture of a car on a street

One last example i would like to talk about which i personally found interesting is that i tried using a picture of a crashed car and the results were quite impressive

We got a score of 99.9, all these pictures are pictures that our CNN has never seen before and they do look somehow different from the training set, however our CNN was still able to generalize well which proves that our model is actually robust.

Justification

The results were quite good judging from the examples provided above, and we reached an accuracy on the validation set around 0.95, perhaps with more training data points and datasets that actually have pictures of cars driving on a regular street in

different conditions and we could achieve an even more robust model, as we mentioned in the section above our model was able to generalize really well, so i do believe that our model's solution is significant enough to have solved the problem.

My model had an accuracy around 0.95 on the validation set and googLeNet has around the same accuracy, so when comparing my model to the benchmark model mentioned above, we can see that it reaches the benchmark accuracy. googLeNet is a powerful model and i think i am really satisfied with the results since i have reached the benchmark results.

V. Conclusion

Free-Form Visualization

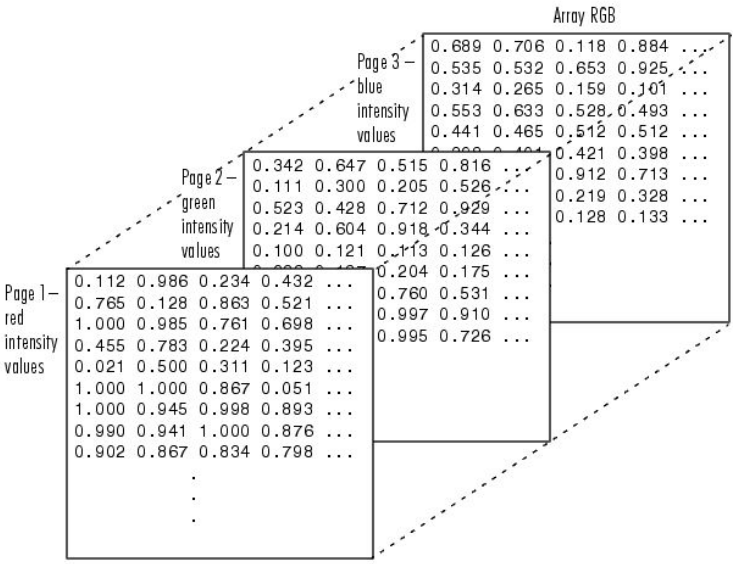
Before feeding my CNN with the training data, i had to do some preprocessing for the pictures, perhaps one of the preprocessing steps that i think needs visualization is converting each picture to a grayscale picture, here is an example taken from our training set:

Original Image

Image after converting to gray

It's also important to mention that when we have the original image before the conversion to grayscale, the image is represented as a 3-Dimensional array, as if we

have 3 two dimensional arrays, each one of these 3 two dimensional array represents a channel out of the three channels; red, green and blue like this:



While grayscale images are just represented as a regular 2D grid, we can see by converting an RGB image into grayscale, we decrease the size of the image, thus decreasing the complexity of our model, if we were to use colored images with our model, we would have increased the complexity and since we have around 8000 pictures we would have increased it heavily and the number of parameters needed to train the CNN might have increased a lot as well.

Reflection

The problem in general was classification problem based on image processing, I have heard so much about this image processing before I even got into machine learning and I was happy that I had to work with something like that, more specifically we are trying to detect if a picture has a car in it or not, using a deep learning approach we were able to train our model, and our CNN was able to learn how a vehicle looks like.

I think one of the most difficult aspect i faced during this project is the limited computational powers i have, and i think this problem arises a lot whenever we use a CNN since they require a big computational power.

However, one of the most interesting things I found during this project is working more with keras, since it's a very powerful tool to use and i was able to explore some of its various functions.

I think the final model and solution does fit my expectations, however i do look to expand this model in the future, hopefully when i can afford a more powerful platform or an amazon aws instance, i will be able to build more complex models that will more complex tasks

Improvement

Our model perhaps could be improved by using a more complex CNN, there are many other complex CNN's that I was not able to train due to the limited computational power, so adding additional hidden layers and building more complex models could actually make my model even better.

Another thing I have found to be extremely helpful in image processing in general is computer vision techniques, Many filters could be applied to our images before training which can make the objects within the images more distinct.

Perhaps another thing that could improve our CNN is having more data during training which could make our CNN generalize even better for unseen examples since it has learned even more and more features of how a car looks like and how it doesn't.

References

http://cs231n.stanford.edu/reports/2015/pdfs/lediurfinal.pdf

http://ai.stanford.edu/~jkrause/cars/car_dataset.html

http://www.ais.uni-bonn.de/download/datasets.html.

https://stackoverflow.com/questions/47889999/expected-convolution2d-input-1-to-have-4-dimensions-but-got-array-with-shape-1

https://www.mathworks.com/help/matlab/math/multidimensional-arrays.html