Computer Vision and Tourist Attractions

Carlos Ronchi CARLOSHVRONCHI@GMAIL.COM
Universidade Federal do Paraná - UFPR

Abstract

Computer Vision has been achieving great advancements with the new state-of-the-arts algorithms, such as convolutional neural networks and its variants. With all these possibilities it's been possible to use images for cancer diagnosis (Litjens et al., 2016) and recognizing places in geral (Zhou et al., 2014). This current work intends to use convolutional neural networks and other models from machine learning to identify and predict some tourist attractions from the city of Curitiba, Brazil. This can be useful for tourists visiting the city and blind people who wants to know more about their surround.

1. Introduction

The growing development of artificial inteligence (AI) allowed us to use images in the context of real world applications. One subfield of AI called Computer Vision, works towards developing technics in order to solve problems arising from images and videos for example.

With such development, one can try to improve people's life using these algorithms. In that sense, one will implement convolutional neural networks and softmax regression to obtain a classifier of tourist atractions in Curitiba, which are the five following: The Cathedral Basilica Minor of Our Lady of Light, Botanical Garden, Oscar Niemeyer Museum, Wire Opera House, Paiol Theater.

2. Softmax Regression

Given a dataset $\{(x^{(i)}, y^{(i)})\}_{i=1}^{m}$ with labeled data, that means $x^{(i)} \subset \mathbb{R}^n$ and $y^{(i)} \in \{1, \ldots, k\}$ in the case of a multi-class problem. We want to try to predict a $\hat{y}_i$ such that it's close to $y^{(i)}$ given $x^{(i)}$ and a parameter $\theta \in \mathbb{R}^n$, which we do not know a prior. Instead of taking the sigmoid function which handles only the binary classification at once, consider the softmax function given in Equation (1). In other words, given a test input x, we want our hypothesis to estimate the probability that $p(y = j|x)$ for each value $j = 1, \ldots, k$.

$$\begin{aligned} m_\theta(x) &= \begin{bmatrix} p(y^{(i)} = 1|x^{(i)}; \theta) \\ p(y^{(i)} = 2|x^{(i)}; \theta) \\ \vdots \\ p(y^{(i)} = k|x^{(i)}; \theta) \end{bmatrix} \\ &= \frac{1}{\sum_{j=1}^{k} e^{\theta_j^T x^{(i)}}} \begin{bmatrix} e^{\theta_1^T x^{(i)}} \\ e^{\theta_2^T x^{(i)}} \\ \vdots \\ e^{\theta_k^T x^{(i)}} \end{bmatrix}, \end{aligned} \tag{1}$$

where $\theta_1, \ldots, \theta_k \in \mathbb{R}^n$ are the parameters of our model. Furthermore, notice that the sum $\frac{1}{\sum_{j=1}^{k} e^{\theta_j^T x^{(i)}}}$ normalizes the hypothesis vector, so that when one sums up the entries it'll give 1.

When using logistic regression, we tried to minimize the Equation (2)

$$\begin{aligned} f(\theta) = -\sum_{i=1}^{m} \Big[& y^{(i)} \log\left(m_\theta(x^{(i)})\right) \\ & + (1 - y^{(i)}) \log\left(1 - m_\theta(x^{(i)})\right) \Big], \end{aligned} \tag{2}$$

with respect to the variable θ. But since we changed our hypothesis function, we should consider the function given in Equation (3) to be minimized.

$$J(\theta) = -\sum_{i=1}^{m} \sum_{j=1}^{k} 1_{y^{(i)}=j} \log\left(\frac{e^{\theta_j^T x^{(i)}}}{\sum_{l=1}^{k} e^{\theta_l^T x^{(i)}}}\right), \tag{3}$$

where $1_{y^{(i)}=j}$ is the indicator function, that means, if $y^{(i)} = j$, then $1_{y^{(i)}=j} = 1$, otherwise $1_{y^{(i)}=j} = 0$. Note that if we consider $k = 2$ and using some tricks we obtain logistic regression back.

Final project for CM116.

Moreover, observe that θ is a matrix, whose column vectors are the parameters $\theta_1, \ldots, \theta_k$ we are trying to adjust. In order to minimize the function $J(\theta)$, we can use gradient descent methods, such as the steepest descent gradient, stochastic gradient descend, adam and many others.

Taking derivatives, one can show that the gradient is

$$\nabla_{\theta_j} J(\theta) = \sum_{i=1}^{m} \left[x^{(i)} \left(1_{y^{(i)}=j} - p(y^{(i)} = j | x^{(i)}) \right) \right], \quad (4)$$

where

$$p(y^{(i)} = j | x^{(i)}) = \frac{e^{\theta_j^T x^{(i)}}}{\sum_{l=1}^{k} e^{\theta_l^T x^{(i)}}}.$$

And for example, using the gradient descent we would have the following rule for minimizing our function.

$$\theta_j = \theta_j - \alpha \nabla_j J(\theta) \quad \forall j = 1, \ldots, k.$$

3. Deep Learning

Using neural networks with several stacked layers is called Deep Learning. These have been proven in the last few years to be very promising in the tasks of computer vision (Krizhevsky et al., 2012), (Zhou et al., 2014).

Convolutional neural networks are a kind of neural networs with more operations instead of just using the vanilla *input* → *activation* → *output*. There are two fundamentals operations, the *convolution*, hence convolutional neural networks and *pooling*. The convolution layer takes as input an matrix, which can be 3d or not, and apply a certain number of filters (operations) over this matrix trying to find patterns. One such example is imagine one has a image of a square, and we apply the convolution with 4 filters of a certain size. They might find patterns, which will be in this case the four edges of a square. Roughly speaking, these filters are operating through matrix multiplications and sum of numbers.

The pooling can be thought as a max pooling and average pooling, the well know poolings. A max pooling of size s will take a sample of your image (as a matrix) and transform this square into a number, the maximum of the sample. And so one does over the whole image. The average pooling does the same, but instead of taking the maximum element in the sample we take the average.

4. Data

Data has been collected web scrapping *Google Images*. The Table 1 shows the number of images collected and cleaned for the use of the algorithms.

Table 1. Number of images from Tourist attractions.

Tourist Attraction	Number of Images
Cathedral	100
Botanical Garden	171
Museum	92
Wire Opera	157
Paiol Theater	73

Figure 1. Cathedral

The pre-processing step for the images consisted in resizing the images to 224px × 224px.

Figure 2. Botanical Garden

5. Implementation

It has been used the API from Google, *TensorFlow* (Abadi et al., 2015) to implement every algorithm.

The algorithms implemented were the softmax regression and two kinds of convolutional neural networks. All implementations were made using Python and are

available at Github.

For the deep learning models, two architectures were used, namely, a vanilla one given in TensorFlow's website and the other was based in the article (Krizhevsky et al., 2012). The architectures are shown in Table 2

Table 2. Architecture of the convolutional neural networks

ConvNet Configuration	
Model 1	**Model 2**
4 weight layers	7 weight layers
Input (224x224 RGB Image)	
conv5-32	conv11-64
maxpool	
conv6-32	conv5-192
maxpool	
FC1024	conv3-384 conv3-256 conv3-256
	maxpool
	FC4096 FC4096
Out	
Softmax	

Here *conv3-384* means that we are applying a convolutional layer using 384 filters each of size 3×3.

For the minimization of all functions it was used the Adam Optimizer (Kingma and Ba, 2014), whose routine is implemented in TensorFlow, and a mini batch rule for calculating the gradient, since the computer could not handle all the training examples at the same time. The mini batch sizes varied between 10 to 220. It is important to notice that the activation function between the layers was ReLU, which is given on Equation (5)

$$f(x) = \max(0, x). \tag{5}$$

For training the model it was used 80% of the data and 20% used for calculating the accuracy of the model.

6. Results

Below in Table 3 one can find the results for every model and with the number of classes used for that specific model. The values in each sell represent the accuracy of the model calculated on the test set.

As can be seen above, the first model did not perform so well, this might be happening because it is a shallow neural network, thus can not learn many parameters fully. In light of that, it was chosen that the architecture of model 1 was no to not use for further training new models.

Table 3. Accuracy for every model for every number of classes.

#Classes	Model 1	Model 2	Softmax
2	65,38%	88,46%	75%
3	-	80%	69,23%
4	-	71,84%	65,05%
5	-	75,83%	63,33%

Notice that the Softmax and Model 2 achieved almost the same accuracy, eventhough the running time for the Model 2 was much bigger (around 2 hours), while for the softmax was under 10min.

In the case of five classes, the perfomance of Model 2 was much better than the softmax, mainly because it can learn more parameters, it's more deep and complex than a softmax model.

When Classes = 3, observe the Figure 3, as one can notice, the softmax error (upper plot) became less noisy with fewer steps than alexnet does, this might happens since alexnet is deeper and more complex in order to learn.

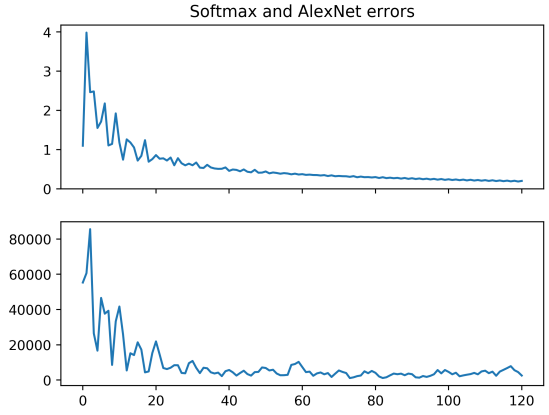

Figure 3. Erros calculated on mini-batchs during training. Softmax error (upper image) and alexnet error (lower image).

7. Conclusion

The results obtained showed that CNN's can achieve better results than softmax regression. Furthermore, it might be possible to train this model with more classes given more data. The models learnt well enough considering that data was scarce.

8. Future work

This work can be extended using newer convolutional neural networks, such as VGG16 (Simonyan and Zisserman, 2014), Inception by Google (Szegedy et al., 2014) and GAN (Goodfellow et al., 2014).

Another way for improving the results is adding preprocessing steps, such as taking the mean from the training set, autoencoders, which are neural networks with one small hidden layer and the output has the same size as input. Other way of improving it is using the CNN's as feature extractors and linking them as inputs to a linear SVM.

Images are very important in this process, so to further improve the results it is necessary to obtain more data, perhaps through a web pipeline where people upload pictures and open sourcing this project. Other way would be artificially increasing the dataset using data augmentation.

References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., & Zheng, X. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems.

Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., & Bengio, Y. (2014). Generative adversarial networks.

Kingma, D. P. & Ba, J. (2014). Adam: A method for stochastic optimization.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In: *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, & K. Q. Weinberger, ed., pages 1097–1105. Curran Associates, Inc.

Litjens, G., Sánchez, C. I., Timofeeva, N., Hermsen, M., Nagtegaal, I., Kovacs, I., van de Kaa, C. H., Bult, P., van Ginneken, B., & van der Laak, J. (2016). Deep learning as a tool for increased accuracy and efficiency of histopathological diagnosis. *Scientific Reports*.

Simonyan, K. & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., & Rabinovich, A. (2014). Going deeper with convolutions.

Zhou, B., Lapedriza, A., Xiao, J., Torralba, A., & Oliva, A. (2014). Learning deep features for scene recognition using places database. *Advances in Neural Information Processing Systems 27 (NIPS)*.