Machine Learning Nanodegree

Capstone Proposal

Thiago Moretto

July, 08, 2018

Furniture classification

1. Domain background

Object classification/recognition from images is a suitable task for Machine Learning techniques nowadays, especially with Deep Learning techniques [1]. The performance achieved using Deep Learning for object classification task is impressive. Several uses of these techniques can be seen in consumer products (e.g. Google's products) [2].

The leverage of Deep Learning comes from the evolution of GPUs (Graphical Processing Units), helping the academia and industry to advance the techniques, exploring deeper and complex Neural Networks [1]. Even GPU vendors are building dedicated and powerful hardware to speed up the evolution of Deep Learning models [3].

The problem presented here is to classify some categories of products from images, in this case, furnitures. With the advent of online marketplaces (like part of the business of Amazon, Walmart and others), the classification of products becomes a hard task since several sellers that sells the same or similar products, give to the marketplaces poor or few information of the products they are selling, making it hard to marketplace to group products together and/or deduplicate some records [4].

Using images of the products might help on this hard task to correct classify, and possible, deduplicate records of the products on those databases. Making the customer experience smarter and better.

1.1. Motivation

The motivation behind this decision is a personal interest in this kind of problem, especially when dealing with Deep Learning and images. Also, I've worked to a e-commerce marketplace before so I've faced this kind of problem there. Unfortunately I ain't working with this anymore but the interest remains live.

2. Problem statement

In this project, we'll use Deep Learning to classify the furniture present in an image. This project will be based on the dataset provided and used in the iMaterialist Challenge (Furniture) at FGVC5 [5], a Kaggle Competition.

The problem can be defined in one statement as: Given an image, classify the type of furniture present on it. Once the problem is to classify images of these kind of products and potentially used to classify, automatically, thousand of images, it's suitable to say that we should use a computer to do this task.

Even though, we, humans, are good at image classifications, doing this for thousand of images in a matter of few minutes using people, it is practically an impossible task or at least a lot of people will be necessary to do this in parallel. Thus, automating this kind of task is very important for certain businesses a can save a lot of resources.

The solution I propose is to create a classification model using a pre-trained Convolutional Network to extract features from images and the final classification model itself, trained for the purpose.

3. Datasets and inputs

Originally, the published dataset provided by the competition has more than 180000 images for training. Images are classified into 128 possible labels.

For the purpose of this project just part of the labels and images may be used, this is due to budget and computing power restrictions. Datasets of the *top-5* classes and images were prepared and will be used to build the classification model. The datasets are balanced since we are selecting a limited amount of images and selecting the almost the same number of each class, reducing issues when dealing with imbalanced datasets.

Three datasets were prepared (they just vary in amount of images), one small with 1000 images (200 images for each class) [~200MB], other with 2500

images (500 each class) [~500MB] and the last one with 5000 images (1000 each class) [~900MB]. The images vary in dimensions, they are not standardized, and they have color layers. The images are 8-big sRGB.

Example of images of the dataset:



The original dataset (from the competition) is available as *json-format* file and inside it there are URLs to download the images, some of these images are not available anymore. The images were already downloaded of my part, so it will be provided (as part of this project) a gzipped version of each dataset part as well as any other resource to run this project.

The dataset will be split into trained set, validation set and test set. The proportion of each will be defined later.

4. Solution statement

The suitable Deep Learning model for this task is a Convolutional Neural Network. Unfortunately, training those models from scratch are expensive in terms of computing power and time. Instead, the approach chosen here is to solve this problem using a pre-trained model. Using a pre-trained model is common in industry to speed up the process.

The solution will be made with a pre-trained CNN model to extract features from the images and a fine-tuned classification model (a Fully-Connected Neural Network).

5. Benchmark Model

One of the possible benchmark are the compare the solution against solutions the same problem trained from scratch (without any pre-trained model), a vanilla

model. The vanilla model will serve as baseline of the improvement we can get using transfer learning of a pre-trained model.

The CNN architecture that will be used as vanilla model, is describe more in details in *Project Design* section, but basically will one with multiple **Convolutional Layers** and a **Fully-Connected** at the end.

6. Evaluation Metrics

Since the dataset is balanced, in order to evaluate the results, accuracy will be chosen as final metric. The choice of pre-trained model will be based on resource-constraint and quality of results using this metric. In the *Project Design* the candidates of pre-trained model are specified.

7. Project Design

This section describe a bit about how the vanilla model will be build and trained, as well as the classification model with a pre-trained CNN. At the end is discussed about fine-tuning.

7.1. Vanilla model

For the vanilla model, the images will resized to 244x244 and pixels, normalized. To fair comparison between the models, both of them will use the same dataset, without any modification.

The planned vanilla model to classify images will be designed with a sequence of **Convolutional Layers** with **Max Pooling** and **Dropout** (if need, as regularizer) in between. The final classification layer can be a typical **Fully-connected**. This kind of architecture I've worked with in the past and usually performs reasonably well on this kind of task.

The following Keras model summary shows an example of this architecture and, of course, the version that will be used might differ of this shown here.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_21 (Conv2D)           (None, 222, 222, 32)      896
_________________________________________________________________
max_pooling2d_21 (MaxPooling (None, 222, 222, 32)      0
_________________________________________________________________
dropout_5 (Dropout)          (None, 222, 222, 32)      0
```

```
_________________________________________________________________
conv2d_22 (Conv2D)           (None, 220, 220, 64)      18496
_________________________________________________________________
max_pooling2d_22 (MaxPooling (None, 110, 110, 64)      0
_________________________________________________________________
dropout_6 (Dropout)          (None, 110, 110, 64)      0
_________________________________________________________________
conv2d_23 (Conv2D)           (None, 108, 108, 128)     73856
_________________________________________________________________
max_pooling2d_23 (MaxPooling (None, 108, 108, 128)     0
_________________________________________________________________
dropout_7 (Dropout)          (None, 108, 108, 128)     0
_________________________________________________________________
global_average_pooling2d_8 ( (None, 128)               0
_________________________________________________________________
dense_9 (Dense)              (None, 5)                 645
=================================================================
Total params: 93,893.0
Trainable params: 93,893.0
Non-trainable params: 0.0
```

In order to improve the quality of the model, data augmentation can be considered. Although it is almost sure that a vanilla model trained with few resources is certain that will perform worse that a model that uses a pre-trained ConvNet, exploring its potential can be rewarding.

Summing up, the vanilla model will receive a tensor of shape (batch_size, 244, 244, 3) and will output a vector of shape 5 which the probabilities of each class based on an given image. As an activation of output layer, the Softmax will be used.

7.2. Transfer learning

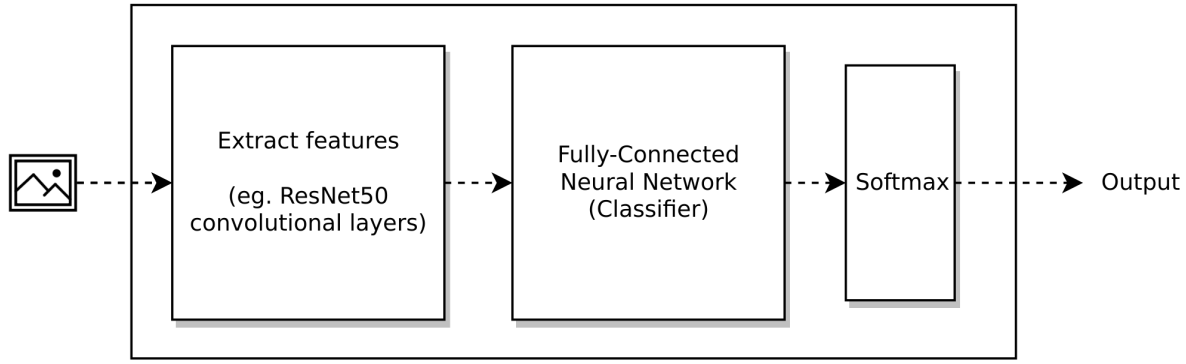
In this scenario, I am going to use a pre-trained model to extract the bottleneck features from the images. The images will be resized to 244x244 and bottleneck features extracted and saved to disk (caching purposes). At this moment there is no plan to, for example, crop or apply any transformation in images. But, during the workflow, we can include this step to improve the results.

The candidates as pre-trained model would be *Resnet50* and *InceptionV3*, they are available in Keras and ready to use. To speed up the process the bottleneck features extracted will be saved to the disk and will be available for download later to the reviewers.

The classification model will be created and prepared to consume as an input the features extracted using the pre-trained CNN. A **fully-connected** layer with

Softmax as activation function of the output layer. A **Dropout** layer can be used to help in generalization as well as to reduce overfitting problems.

Once having a model trained, a typical and simplified final structure of a algorithm to perform the classification is shown below:



7.3. Fine-tuning

For both cases, where applicable, a fine-tuning on training and in the models to improve the quality of the results will be made. For example, switching the optimizer and their parameters, like switching from **Adam** to **SGD** or **RMSprop**.

Adjusting parameters like **learning rate**, **batch size** and **momentum.** Or the parameters available for each optimizer that might lead to optimization. An alternative to fine tune the parameters is by applying a grid search of something similar to explore the parameters automatically.

One useful tool to explore the training and results is the **TensorBoard**. This tool can be used to help to understand the training procedure and potential issues and optimization opportunities.

References

[1] Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." *Advances in neural information processing systems*. 2012.

[2] Abadi, Martín, et al. "Tensorflow: a system for large-scale machine learning." *OSDI*. Vol. 16. 2016. Jianmin C., Zhifeng C., Andy D. et al.

[3] https://developer.nvidia.com/deep-learning

[4] Fensel, Dieter, et al. "Product data integration in B2B e-commerce." IEEE Intelligent Systems 16.4 (2001): 54-59.

[5] https://www.kaggle.com/c/imaterialist-challenge-furniture-2018