Furniture classification using images and Deep Learning

Thiago Moretto thiago@moretto.eng.br

1. Definition

1.1 Project Overview

Automated product classification became a pretty common and one of the hardest task for big marketplace sites (Fensel et al., 2001). One of their task is to aggregate thousands of products, using information given from several sellers, eventually with misleading information, from distinct languages, incomplete data and so on. Along with the product's textual meta-data provided by the sellers, images can be also used to classify these products, improving the quality of product categorization and matching.

Image recognition task has gained power with the advent of Deep Learning techniques and powerful hardwares available (Krizhevsky et al., 2012; Abadi et al., 2016), making an expensive task affordable again. This project aims to solve the problem of product classification using the technique, at least part of the problem, by using images. In this particular case, the interest is to classify a small set of products' subcategories of a specific main category, the furnitures.

1.2 Problem statement

The goal of this project is to create a product category recognizer from images, as mentioned, some kind of furniture. Given an image of a product, the solver must outputs the most probably category of the item being shown in the image.

The approach taken to solve the problem is to use an annotated dataset of these products, training a classifier using Deep Learning techniques. Roughly speaking, the tasks involved are the following:

- Preprocess images, resizing, normalizing and/or extracting features splitting them into train, test and validation sets.
- Train the classifier to output the probabilities of each category given a image.
- Prepare a simple algorithm that uses the classifier and given an unseen image output the class of the product present on it.

It's expected that the final model outputs the class of the product with a reasonable certainty.

1.3 Metrics

In order to measure the quality and the certainty about the answer that the model will provide, some metrics must be used. Since the dataset provided is balanced, as metric, **accuracy** will be used.

The accuracy will be measured after training as:

$$acc(y, \hat{y}) = \frac{1}{n_{samples}} \sum_{i=0}^{n_{samples}-1} 1(\hat{y}_i = y_i)$$

2. Analysis

2.1 Data exploration

Images are based on the dataset provided and used in the iMaterialist Challenge (Furniture) at FGVC5 (Kaggle, 2018), a Kaggle Competition. Originally, the published dataset provided by the competition has more than 180000 images for training. Images are classified into 128 possible classes.

For the purpose of this project just part of the classes and images were used, this is due to budget and computing power restrictions. Datasets of the top-5 classes and images were prepared. It was generated three datasets varying in size, one with 1000, other with 2500 and last one with 5000 images. Models shown here were trained with the 5000 images dataset.

2.2 Exploratory visualization

The 5 classes present in the dataset are: Mug, Cup (of Glass), Pillow, Desk and Table lamp. Figure 1 shows examples of pictures of each of these classes.

(a) Mugs (b) Cups

(c) Pillows (d) Desks

(e) Table lamps

Figure 1: Example of pictures of each class

Figure 2 shows the t-SNE clustering plot. It was applied a t-SNE (perplexity = 20) to see whether there is evident pattern in data or not. Unfortunately, due the complexity of the images, the plot didn't show too much. There are some areas of concentration like table lamps and desks but of the other classes t-SNE could not clearly separate them.

(a) t-SNE of 1000 images (b) t-SNE of 2500 images

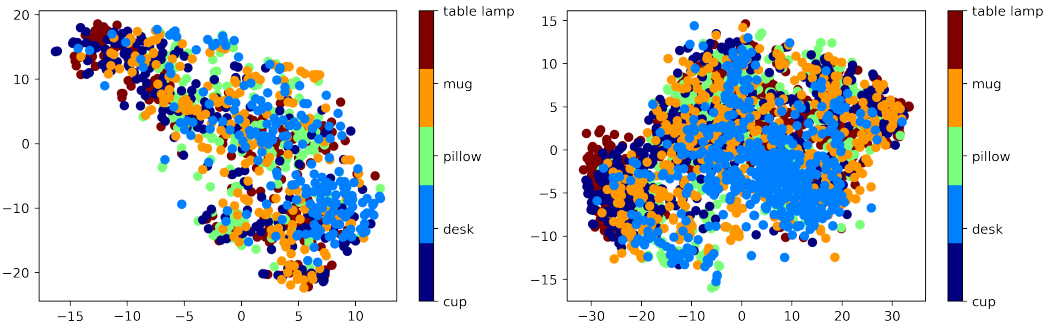


Figure 2: t-SNE plots (perplexity = 20)

All datasets provided in this project are perfectly balanced, they have been pre-processed and prepared to have the same amount of images of each class (check the README for links to download). During the training of the model, images are resized to 224x224 with 3 channels.

2.3 Algorithms and Techniques

This project used a Convolutional Neural Network (CNN), as it is a state-of-the-art architecture to solve this kind of problem, that involves finding patterns in images. CNN is a special kind of Neural Network, and usually it is applied to visual imagery problems. In part, CNN are good in this kind of task due its spatial invariant ability, which means that can recognize patterns that can be translated, shifted or had this shape deformed.

This spatial invariant ability that makes CNN to performs so well for images, can be assigned to the characteristic of the convolution layer, that plays the role to identify the high level feature of an image, such as edges, curves, straight edges and colors. To achieve this capacity, the convolution layer uses filters (also known as convolution kernel), with a predefined size, sliding through the whole image from top-left, passing the input to activation layer a thus producing a 2-dimensional activation map.

One of the approaches taken in this project is the known Transfer Learning, roughly speaking, it is like picking part of already trained (the weights) Neural Network and using in a new one, training the rest for a new task. The part taken were exactly the convolutional layers, using it as *feature extractor*, to exactly identify what was already mentioned about CNNs, the high level feature of an image, such as edges, curves, straight edges and colors.

The benefits of the Transfer Learning approach is the reuse of an previous and expensive effort used to build these convolutional layers. They are expensive to build from scratch. Best known CNNs trained to recognize a wide of range of objects from images, usually, took several hours, days or even weeks to be trained, even in powerful and dedicated hardwares.

So, the model that uses the Transfer Learning approach built for this project was significantly cheaper to train, and once it uses of a good pre-trained model, it still performs really well.

2.4 Benchmark

Something between 80-90% of accuracy it is the objective to reach by the final solution. To serve also as benchmark and as a baseline, a model was built and trained from scratch (including the Convolutional Layers). The vanilla model. The hypothesis is that the model that uses another pre-trained model as feature extractor outperform the vanilla model. Same dataset was be used in both cases for fair comparison.

3. Methodology

3.1 Data preprocessing

For the Transfer Learning approach, images were resized to 224x224 (required by pre-trained model) and *preprocess_input* method (of Keras and Tensorflow) used to adequate the input to the shape required by the pre-trained model. For the benchmark model (Vanilla), images were also resized to 224x224, input normalized and converted into tensors.

No data augmentation technique was made. During the tests, using the half (2500 images) of the entire of the dataset (5000 images) showed similar results, indicating that augmentation wasn't strictly necessary to make the final model reach the initial expectations. The vanilla model might get some improvement using the technique, but as long as it is just the baseline model, no much effort was made in order to improve it for now.

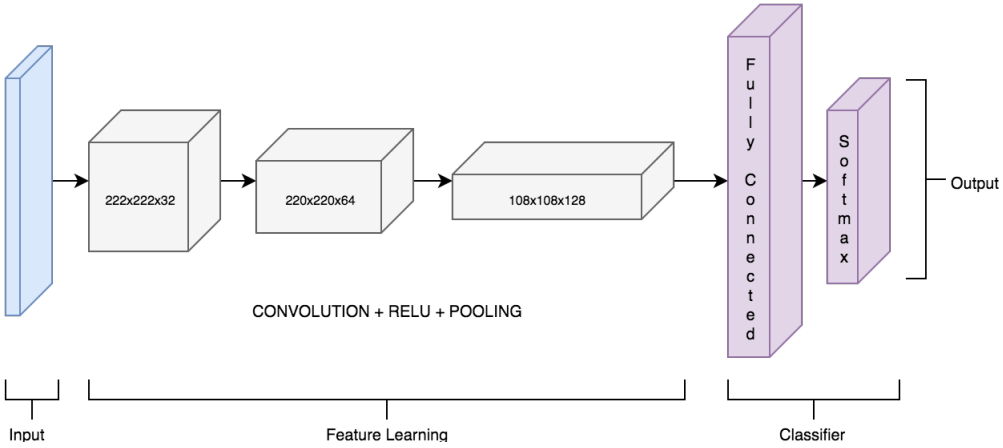
3.2 Implementation

As mentioned, two kind of models were built for this project. Below you will find the implementation details, as well as the architecture of each one. In both cases, some utilities classes were implemented to simplify training and test procedures, as well as to compare the models easily. You might see them in the source code (*utilities* module), they are the Trainer, TrainedModel (encapsulates the trained model providing some methods to interact with to it), the TrainPlotter and TrainTestDataset.

3.2.1 Vanilla model, the benchmark model

One model built to solve the problem is what we call, the vanilla model, a pretty common Convolutional Neural Network architecture without any use of a pre-trained model as part of it, or anyother sophisticated component. It has 2D-Convolutional layers with ReLU as activation functions, MaxPooling was used in between to reduce the resolution and complexity, at the end a GlobalAveragePooling layer that also flattens the 2D structure into 1D input. A fully-connected structure as a classifier with the output layer values being squashed using *Softmax*. The figure 3 shows an overview of its architecture.

Figure 3: Vanilla basic architecture



One challenge faced by building from scratch was to build and train the model efficiently without wasting too much computational resources, managing the training time and, good use of the train, test and validation sets in such way to maintain a sustainable training procedure and controlling overfitting. Optimizers were changed and tested, at the end, the *Adam* was selected as final.

The size and deepness of the model was controlled to maintain the memory usage below this limit, at the same time, maintaining the model complex enough the extract the patterns from data to perform well on its objective.

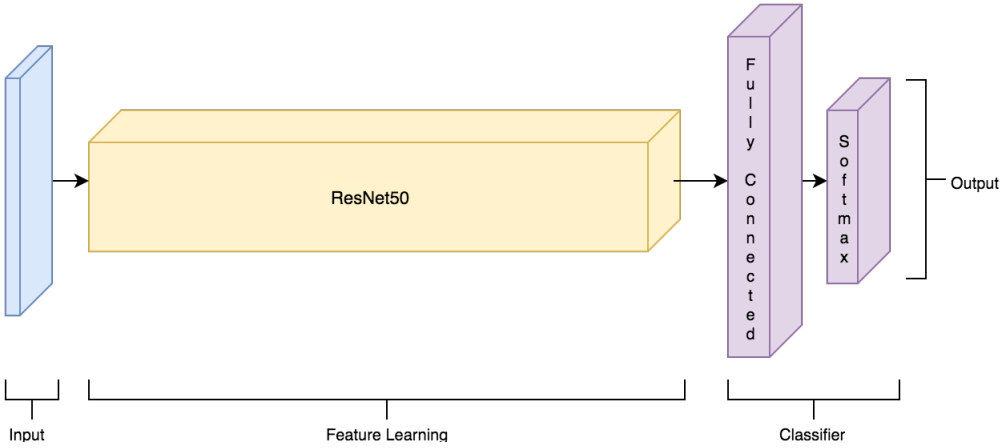
Originally, the initial intention of this project was to use a pre-trained model, due to historical and remarkable results gotten using the strategy in this kind of problem. The vanilla model serves here as a benchmark for the solution based on pre-trained model.

3.2.2 Transfer learning-based model

The main model of this project was based on transfer learning techniques, which consists in using part of a pre-trained model to achieve good quality results faster and without so much computational resources. The chosen pre-trained model was the ResNet50 (Microsoft, 2018). One reason of choosing ResNet50 at first, it was due a previous personal experience with it and because of its complexity, the model does not requires too much computational resources to achieve good quality results.

ImageNet challenge ResNet50's weights were taken as pre-trained model. Using a pre-trained model resulted in the final architecture that is pretty simple (excluding the internal architecture of the ResNet50 itself). This seems to another good reason to use pre-trained models, the resulting architecture of trainable part of the entire model can be simple enough to train fast. The figure 4 shows the resulting architecture.

Figure 4: Transfer-learning based basic architecture



As shown in the diagram, the model basically is part ResNet50 (Convolutional Layers), with the final classification model replaced to achieve the objective of this project. The trainable model receives as a input features extracted by the ResNet50. The trainable model is basically Fully-Connected layers with *Dropout* and *Softmax* as activation of output layer.

One of initial tasks faced to build this model was to prepare the dataset. The trainable part of the whole model it is just fully-connected layers (as mentioned earlier). This means that it is necessary to extract the features of images from the dataset first, these features are commonly known as *bottleneck features*. So, part of the implementation was to create methods to extract these features, and to speed up the process later, the *bottleneck features* of the images were saved to the disk for caching purposes.

The *bottleneck features* extraction is coordinated by *generate_bottleneck_features* method. Images are loaded and resized to 224x224x3, features are extracted and the output reshaped into 2048-size vectors, turning it possible to be fed into a fully-connected model that cannot work with multi-dimensional input directly.

After the bottleneck extraction complete, the training procedure was prepared in similarly way of the made with the vanilla model, using common utilities classes, the Trainer, TrainedModel and so on.

3.3 Refinement

One issue found during training the model using ResNet50 was an huge overfitting problem. Initially, the final classifier was just fully-connected layers, and as input, the features extracted using the ResNet50. In an attempt to reduce this gap or to make the model more generic, a *Dropout* layer was added between two dense layers of the classifier and fine tuned.

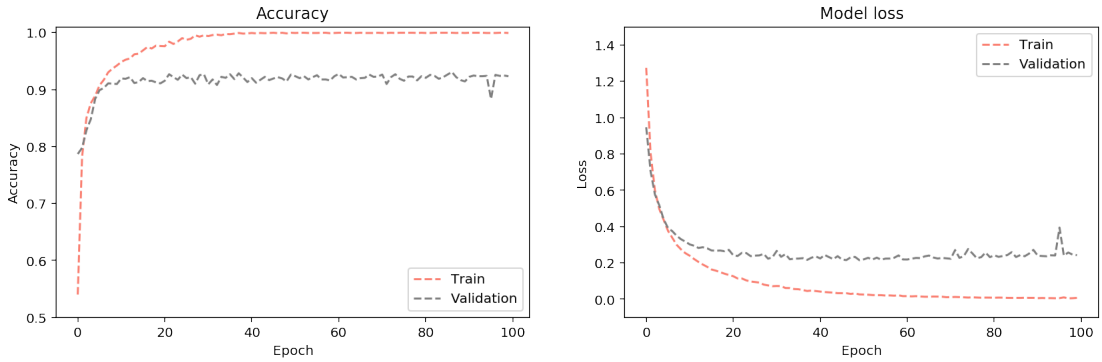
Adding a *Dropout* reduced the training accuracy, as well as the validation a bit, but the gap between the two curves was reduced, indicating a robust model. As *Dropout* refers to ignoring some neurons activations during the training, it ended up affecting the overall accuracy, as long as the number of epochs was maintained the same. This not means that the model is worse, it might just need more epochs for training.

As our intention here is to produce a generic and robust model to classify furniture of images from internet or from products' catalogs, it should be generic enough and resilient

to random changes of the input to make it useful. Other refinements are welcome in the future as improvements to reduce even more this gap as well as the overall quality.

Figure 5 shows the train history after 100 epochs using the model without a *Dropout* layer, the 5 shows the same version just with the dropout added.

(a) Without dropout after 100 epochs



(b) With dropout (0.3 of drop probability) after 100 epochs

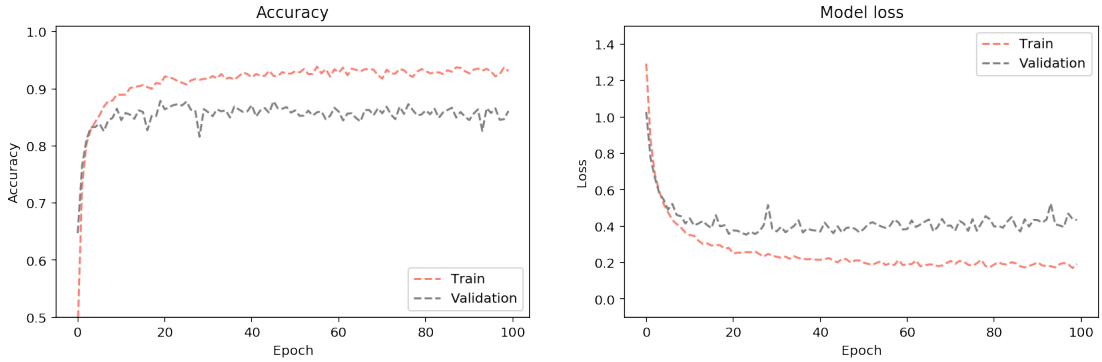


Figure 5: Comparing the addition of a dropout layer

4. Results

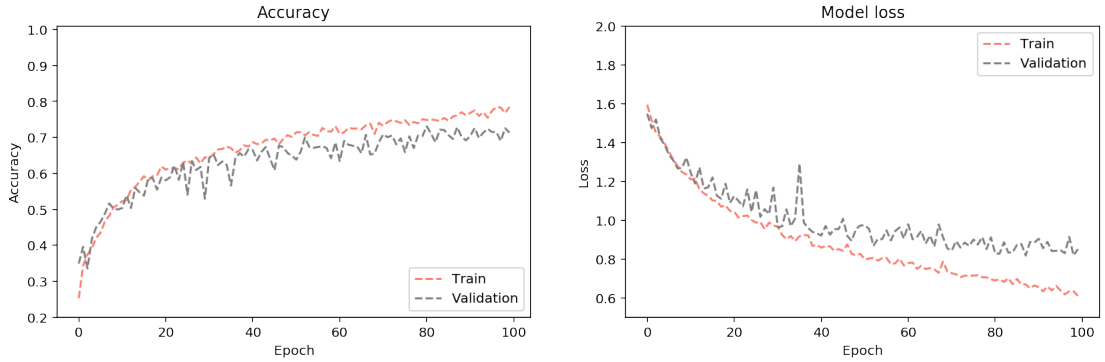
4.1 Model Evaluation and Validation

In this section both models are evaluated and their results discussed.

4.1.1 Vanilla model, the benchmark model

The final version of the vanilla model showed an accuracy close to 70% on test set (10% of the dataset) after 100 epochs using 16 as batch size. Of the total of 5000 images, 3250 were exclusively for training, 501 for testing as unseen images, and 1249 for validation. The Figure 6 shows the training/validation accuracy and loss over epochs.

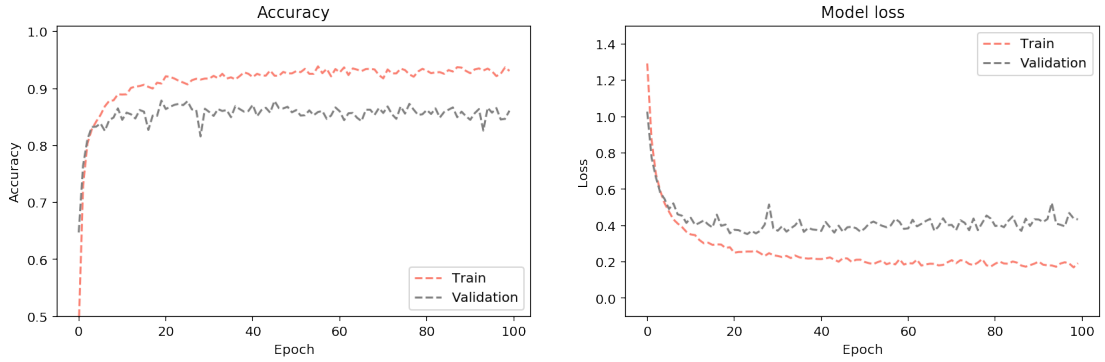
Figure 6: Vanilla model training/validation accuracy and loss curves



4.1.2 Transfer learning-based model

The final version of the main model, the transfer-learning-based, showed an accuracy about 83% on test set, about 13 p.p. above the benchmark model in almost all tests. One issue found was a slightly overfit. In order to reduce and control the overfitting, a dropout layer as a regularizer was added. The figure 7 shows the training/validation accuracy and loss over epochs.

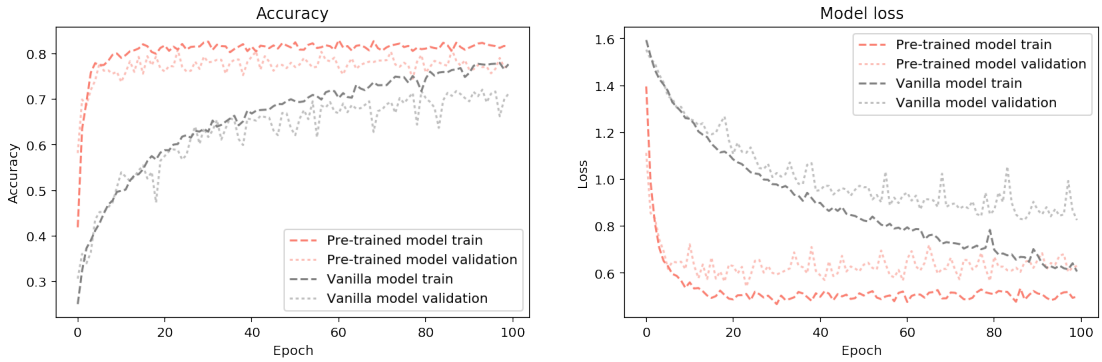
Figure 7: Transfer learning (ResNet50) training/validation accuracy and loss curves



4.1.3 Comparing models

The Figure 8 shows the training curves for both models. Both of them were trained using the same dataset and initially, 100 epochs. It's possible to see that the pre-trained model achieve its best performance in few epochs and stay stable later, without so much improvement. The vanilla model reduces the gap later.

(a) After 100 epochs



(b) After 300 epochs

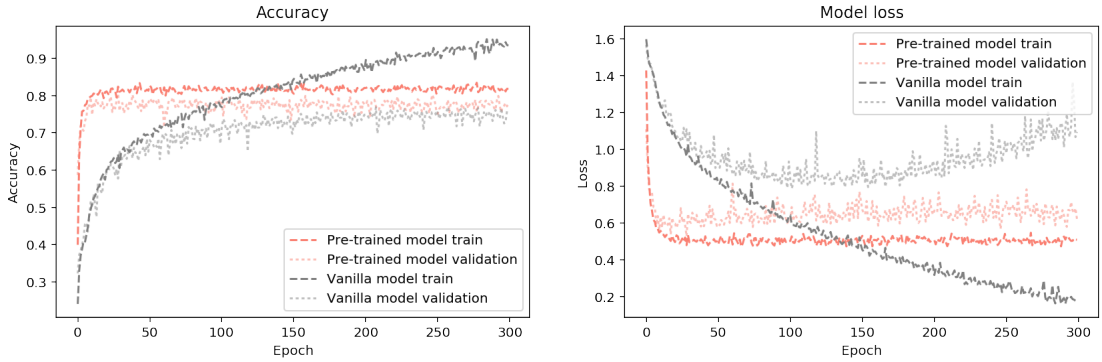


Figure 8: Comparing models' performance

Giving more epochs to the vanilla model in a tentative to outperform the transfer learning based model, did not work totally. In fact, it showed a better training metrics, but the validation did not improved at the same rate, as seen in Figure 8, the validation loss started to get worse after 150 epochs. The overfitting problem hit the vanilla model just by increasing the number of epochs. This might be a indication that the model could not generalizes well and could be too complex for the problem.

4.2 Justification

The final version of the main model, the transfer learning based, showed an accuracy close to 83% in overall, about 13 p.p. above the benchmark model. The advantages of this model that it consumed much less time to train and resources once it used the pre-trained model. It met at least one expectation that was to surpass the vanilla model, in terms of quality as well as the resources expended (time and computational).

As seen in confusion matrix in 9, *Cups* and *Mugs* are classes where the model makes the most of the errors, this looks to be reasonable once the objects shapes are pretty similar. The main difference between them, at least in our dataset, is that the *Cup* is mainly made of glass, transparent.

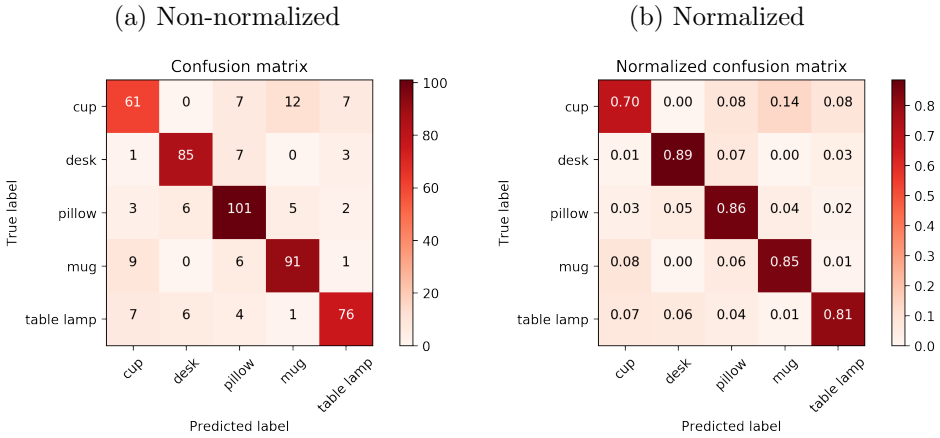


Figure 9: Confusion matrix of test set on transfer-learning based model

Comparing the results using the confusion matrix of the transfer learning based model shown in figure 9 and the confusion matrix of the Vanilla model shown in figure 10, it is possible state that the transfer learning based model outperformed the Vanilla model.

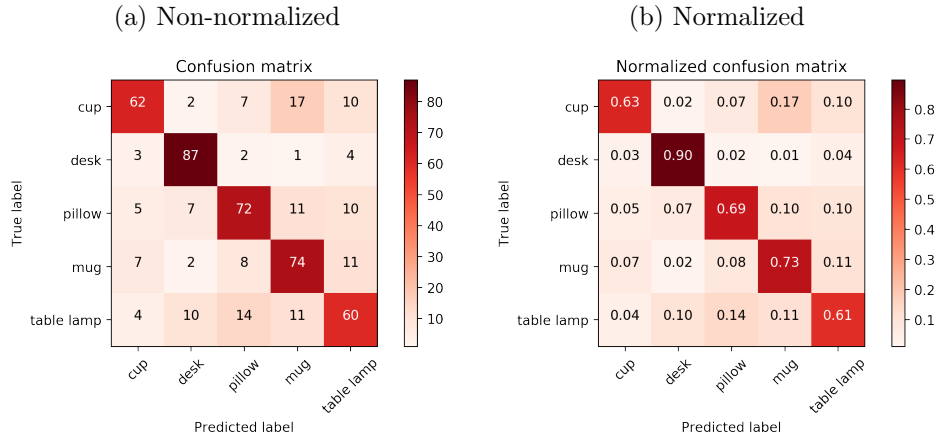


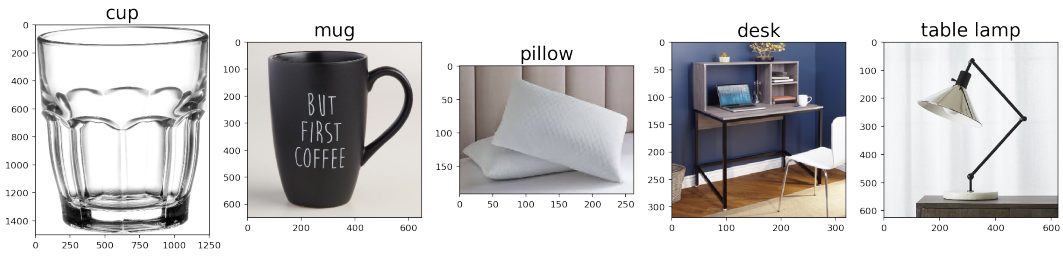
Figure 10: Confusion matrix of test set on Vanilla model

5. Conclusion

5.1 Free-Form Visualization

The final model based on transfer learning techniques was able to recognize the furniture on images from internet. It correctly classified the products as seen in figure 11.

Figure 11: Classified images from internet using ResNet50-based model



5.2 Reflection

The entire process of building this end-to-end solution was:

- Find an interesting problem to solve and publicly available data.
- Prepare the dataset.
- Explore the dataset and check it up
- Create and train and benchmark model.
- Create and train the final model itself.
- Prepare to use the trained models to classify unseen images from internet.

Building the models and doing a properly training was the hardest part. Initially was because of the huge amount of resources it usually needs, also, sometimes the model was too complex or too simple to find the patterns on data. Training, controlling the overfit and convergence also was a bit complicated at first, it is a endless task. However, once a model starting converging, improving the accuracy and loss reducing, it just matter of time and a proper fine tuning to make it better. Of course, you can always improve the model even more. There is not clearly a end, you have to figure out when it is good enough for your purpose.

5.3 Improvement

There are some improvements that can be made to improve the whole project. One that I believe is to properly use Keras generators to reduce memory usage and thus making possible to use more images. Unfortunately after upgrading Keras to make use o new API, it ended up changed some results, totally unexpected, and also some bugs appeared. Living on the edge is hard. Downgrade was necessary to avoid messing entire project.

Other possible improvement, it is to explore more the vanilla model by using Data Augmentation and/or sophisticated image pre-processing to improve the model result quality, although the Vanilla-version wasn't the main model and just served as benchmark, the results of the Vanilla model showed, at least for me, interesting and promising. Putting more effort on it can be rewarding.

References

Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: a system for large-scale machine learning. In *OSDI*, volume 16, pages 265–283, 2016.

Dieter Fensel, Ying Ding, Borys Omelayenko, Ellen Schulten, Guy Botquin, Mike Brown, and Alan Flett. Product data integration in B2B e-commerce. *IEEE Intelligent Systems*, 16(4):54–59, 2001.

Kaggle. iMaterialist Challenge (Furniture) at FGVC5, 2018. URL `https://www.kaggle.com/c/imaterialist-challenge-furniture-2018`.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

Microsoft. Deep Residual Networks, 2018. URL `https://github.com/KaimingHe/deep-residual-networks`.