EXERCISE – 2

Scraping images from IKEA:

Programming language: Python

Libraries used: BeautifulSoup, requests

There are many subcategories inside a category like Childrens beds, Double beds, Single beds, etc and the task is to download all the images from the subcategories and put them inside the main category. I am providing just the main category link to my scrapper eg. www.ikea.in/beds and the program will fetch the subcategory links from the master page and it will visit each link and download all the images of that category. An option to download images upto n pages(where n can be configured, I choose 3) of a subcategory is provided.

Some subcategories contained even more subcategories inside them. Eg. Loft beds and bunk beds under beds. Some of the subcategories had no data in it. Code to handle such cases is written.

I have collected data for 4 classes: beds, chairs, tables, tablewares. Each image has a resolution of 300x300 (images with multiple resolution were available upto 1400x1400).

Next Step: Perform Classification

There are different techniques for image classification like usage of SVM, Logistic Regression, Decision Trees, Random Forest and CNN. CNN is widely used for image classification because it can generalize better and give better results on unseen data.

I choose to select a model trained on imagenet dataset and finetune it for my given 4 classes because imagenet dataset comprises of 1000 classes and so the model has learnt a lot of different features and it is easy for the model to adapt to the new classes. A good accuracy is obtained after training for few epochs only.

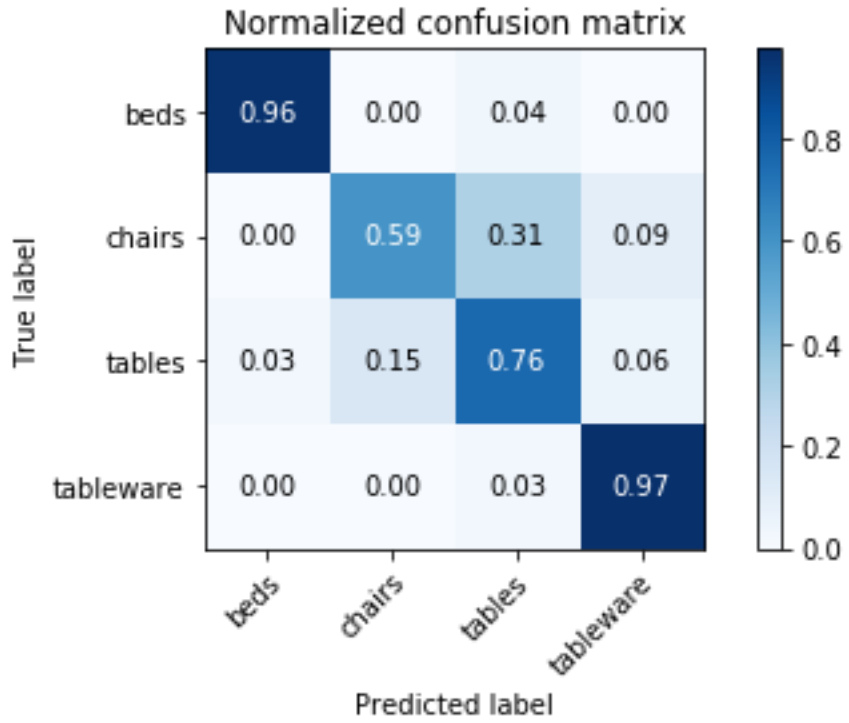
I have selected the model excluding the final layer, added 2 fully connected layers and final output layer of 4 neurons. First I tried with freezing all the layers except the newly added layers and trained the network. After that I un-freezed the last 4

layers before the newly added layers and trained the model which improved the models accuracy. Lower layers of the CNN extract very high level features like edges, corners, etc and as they are already trained on a large dataset, it doesn't need to be retrained. Also training only a few layers makes the process faster.

Total images were split into 10% test , 20% validation(later added some images manually from the train set so around 30% val images) and rest of the images for training.

The images of beds classes were almost half as compared to other 3 classes so I used flipping to augment the data and balance the classes. But the overall training/val images were still less so Augmentation techniques like rotation, width_shift and height_shift were used at runtime using ImageDataGenerator of Keras.

The confusion matrix(normalized) is:



This matrix is calculated on the test data. A test accuracy of 82.57% is achieved.

As seen from the confusion matrix most of the beds and tables are correctly classified. Most of the misclassification was from the chairs data.

Many of the images from chairs dataset were as below:

These are from the dining sets subclass of the chair class. As it has a table, the image is being misclassified as belonging to table class which can also be observed from the confusion matrix that 31% of the chairs are being classified as tables. This also holds true for table class up to a certain extent:

Model HyperParameters:

Model used: MobileNet. Because it's small and has less number of parameters to learn. Since we have a limited dataset, we don't need a large model. I also tried the training with a bit larger models like Xception and InceptionV3 which has higher accuracy benchmarks on Imagenet dataset but got almost same accuracy on my dataset so I decided to proceed with MobileNet as the training was faster on it.

Optimizer: Adam

Learning Rate: 1e-5

Num epochs: 50

Loss: categorical_crossentropy

Files: train.ipynb – create the model and train the dataset.

train_test_val_split.ipynb – split into train/test/val and augment beds class.

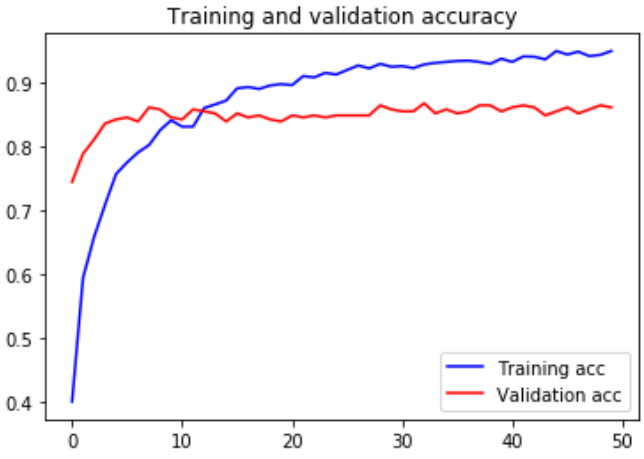
scrap_data.ipynb – to scrap the data from ikea.

Results:

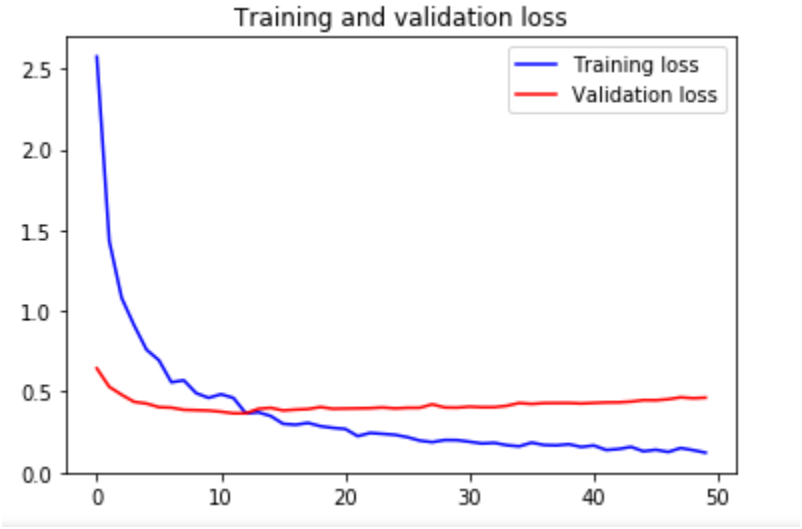
Training accuracy: 94.96%

Validation accuracy: 86.12%

Test accuracy: 82.57%



Accuracy Curve



Loss Curve

Possible Improvements:

Pre-processing the data even more like removing less relevant images.

Augmenting the data to increase the count of training/validation data.