Requirements:

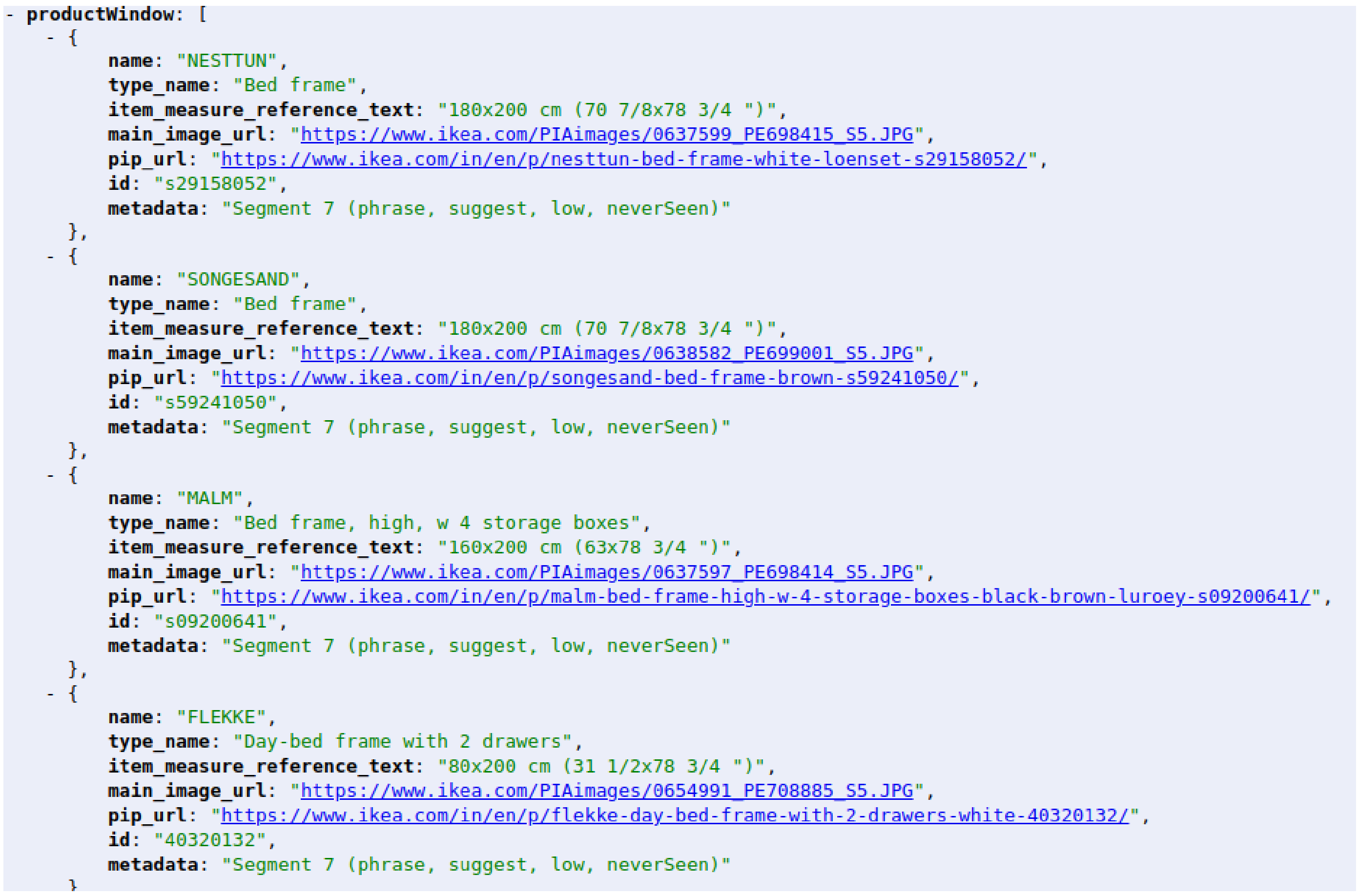
- Python 3.7
- Tensorflow-gpu 1.14.0
- Ubuntu 16.04

Image Scraping from www.ikea.com (http://www.ikea.com)

Path: `scraping_script/ikea_crawler.py`

Command: `python ikea_crawler.py`

I inspected search page of www.ikea.com (http://www.ikea.com) using `Inspect Element -> Network` option in the browser and fetched the url of the catalogue of products. All product image urls are stored in a JSON. Structure of JSON is given below:



```
- productWindow: [
   - {
         name: "NESTTUN",
         type_name: "Bed frame",
         item_measure_reference_text: "180x200 cm (70 7/8x78 3/4 ")",
         main_image_url: "https://www.ikea.com/PIAimages/0637599_PE698415_S5.JPG",
         pip_url: "https://www.ikea.com/in/en/p/nesttun-bed-frame-white-loenset-s29158052/",
         id: "s29158052",
         metadata: "Segment 7 (phrase, suggest, low, neverSeen)"
     },
   - {
         name: "SONGESAND",
         type_name: "Bed frame",
         item_measure_reference_text: "180x200 cm (70 7/8x78 3/4 ")",
         main_image_url: "https://www.ikea.com/PIAimages/0638582_PE699001_S5.JPG",
         pip_url: "https://www.ikea.com/in/en/p/songesand-bed-frame-brown-s59241050/",
         id: "s59241050",
         metadata: "Segment 7 (phrase, suggest, low, neverSeen)"
     },
   - {
         name: "MALM",
         type_name: "Bed frame, high, w 4 storage boxes",
         item_measure_reference_text: "160x200 cm (63x78 3/4 ")",
         main_image_url: "https://www.ikea.com/PIAimages/0637597_PE698414_S5.JPG",
         pip_url: "https://www.ikea.com/in/en/p/malm-bed-frame-high-w-4-storage-boxes-black-brown-luroey-s09200641/",
         id: "s09200641",
         metadata: "Segment 7 (phrase, suggest, low, neverSeen)"
     },
   - {
         name: "FLEKKE",
         type_name: "Day-bed frame with 2 drawers",
         item_measure_reference_text: "80x200 cm (31 1/2x78 3/4 ")",
         main_image_url: "https://www.ikea.com/PIAimages/0654991_PE708885_S5.JPG",
         pip_url: "https://www.ikea.com/in/en/p/flekke-day-bed-frame-with-2-drawers-white-40320132/",
         id: "40320132",
         metadata: "Segment 7 (phrase, suggest, low, neverSeen)"
```

URL: https://sik.search.blue.cdtapps.com/in/en/search-result-page/more-products?sessionId=8a5f6982-3996-4065-bab7-6b1afc97e304&q= (https://sik.search.blue.cdtapps.com/in/en/search-result-page/more-products?sessionId=8a5f6982-3996-4065-bab7-6b1afc97e304&q=) **item_name&start=0&end=2000**

- **item_name** can be any object of our choice from ikea.com.
- **start** and **end** lets us choose number of images we want to scrape.

- Target was to extract 2000 images per object. After removing duplicate links I got 3560(app.) images urls by iterating over a list of JSONs and extracting *image_urls*.
- The script uses simple request-response method to fetch urls and writes images of 4 objects on disk in their respective directory.
- This task can also be achived with the help of scrapy or beautiful-soup.

Example: https://sik.search.blue.cdtapps.com/in/en/search-result-page/more-products?sessionId=8a5f6982-3996-4065-bab7-6b1afc97e304&q=chair&start=0&end=2000&sort=RELEVANCE (https://sik.search.blue.cdtapps.com/in/en/search-result-page/more-products?sessionId=8a5f6982-3996-4065-bab7-6b1afc97e304&q=chair&start=0&end=2000&sort=RELEVANCE)

CNN Image Classification

Data Preparation

```
import split_folders
split_folders.ratio(path_to_dataset,output='split_sets',seed=1337,ratio=(.8
,.2))
```

- I used above snippet to split the data into training and validation sets. 80% training data and 20% validation data.

Link to the dataset: https://drive.google.com/drive/folders/1BUZdmcE7ccaV8lwuqx6LG1-BCBcNsi7L (https://drive.google.com/drive/folders/1BUZdmcE7ccaV8lwuqx6LG1-BCBcNsi7L)

Training

Path to python file: `CNN_Ikea/CNN_Ikea_train.py`

Path to Jupyter Notebook: `CNN_Ikea/CNN_Ikea_train.ipynb`

Last plot and performance evaluation can be seen in the Jupyter file.

CNN configuration:

- Convolution layers: 2
- Kernel size: (3,3)
- Input size: (256,256)
- Pool size: (2,2)
- Dense Layers: 3
- Units per Dense layer: 128
- Optimizer: adam
- Loss Func: categorical_crossentropy
- Batch Size: 32

Model was compiled and fit with above settings.

```
Epoch 1/10
100/100 [==============================] - 218s 2s/step - loss: 1.1545 - acc: 0.5203 - val_loss: 1.1618 - val_acc:
0.5450
Epoch 2/10
100/100 [==============================] - 193s 2s/step - loss: 1.0732 - acc: 0.5685 - val_loss: 1.1289 - val_acc:
0.5254
Epoch 3/10
100/100 [==============================] - 196s 2s/step - loss: 0.9815 - acc: 0.6001 - val_loss: 0.9777 - val_acc:
0.5787
Epoch 4/10
100/100 [==============================] - 196s 2s/step - loss: 1.0161 - acc: 0.5768 - val_loss: 0.9145 - val_acc:
0.6218
Epoch 5/10
100/100 [==============================] - 195s 2s/step - loss: 0.8655 - acc: 0.6504 - val_loss: 0.9115 - val_acc:
0.6193
Epoch 6/10
100/100 [==============================] - 192s 2s/step - loss: 0.8252 - acc: 0.6465 - val_loss: 1.0259 - val_acc:
0.5647
Epoch 7/10
100/100 [==============================] - 192s 2s/step - loss: 0.8202 - acc: 0.6560 - val_loss: 0.9020 - val_acc:
0.6180
Epoch 8/10
100/100 [==============================] - 192s 2s/step - loss: 0.7980 - acc: 0.6754 - val_loss: 0.8745 - val_acc:
0.6510
Epoch 9/10
100/100 [==============================] - 192s 2s/step - loss: 0.7921 - acc: 0.6867 - val_loss: 0.8470 - val_acc:
0.6409
Epoch 10/10
100/100 [==============================] - 192s 2s/step - loss: 0.7881 - acc: 0.6776 - val_loss: 0.8858 - val_acc:
0.6409
```

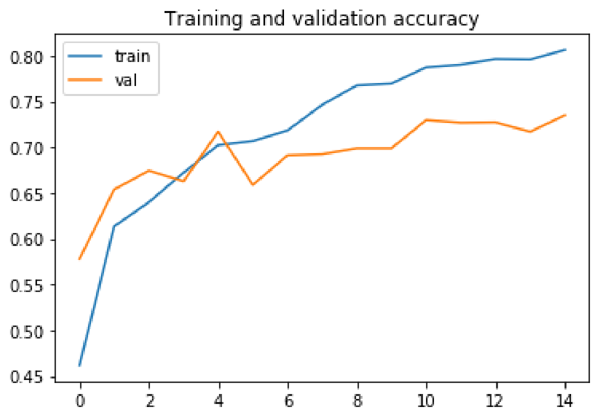
After 10 epochs of training, training Accuracy of 68% and Validation Accuracy of 64% was achieved.

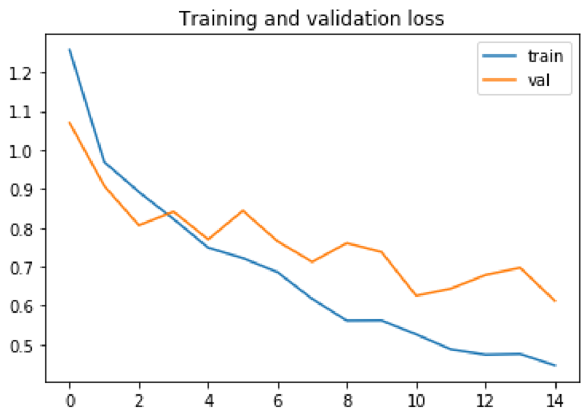
Improving the performance

- To improve accuracy of the model I **added a third convolution layer** followed by another max pooling layer.
- This was done to extract more features from the image.
- I tried to tune hyperparameters by manipulating batch size and number of hidden layers. Adding a third convolution layer with 32 filters gave better results.

```
Epoch 1/15
100/100 [==============================] - 211s 2s/step - loss: 1.2576 - acc: 0.4616 - val_loss: 1.0699 - val_acc:
0.5779
Epoch 2/15
100/100 [==============================] - 200s 2s/step - loss: 0.9693 - acc: 0.6135 - val_loss: 0.9079 - val_acc:
0.6538
Epoch 3/15
100/100 [==============================] - 194s 2s/step - loss: 0.8921 - acc: 0.6399 - val_loss: 0.8061 - val_acc:
0.6744
Epoch 4/15
100/100 [==============================] - 196s 2s/step - loss: 0.8224 - acc: 0.6721 - val_loss: 0.8415 - val_acc:
0.6628
Epoch 5/15
100/100 [==============================] - 195s 2s/step - loss: 0.7487 - acc: 0.7023 - val_loss: 0.7700 - val_acc:
0.7169
Epoch 6/15
100/100 [==============================] - 215s 2s/step - loss: 0.7018 - acc: 0.7067 - val_loss: 0.8438 - val_acc:
0.6589
Epoch 7/15
100/100 [==============================] - 201s 2s/step - loss: 0.6857 - acc: 0.7183 - val_loss: 0.7653 - val_acc:
0.6911
Epoch 8/15
100/100 [==============================] - 202s 2s/step - loss: 0.6169 - acc: 0.7463 - val_loss: 0.7122 - val_acc:
0.6924
Epoch 9/15
100/100 [==============================] - 207s 2s/step - loss: 0.5611 - acc: 0.7676 - val_loss: 0.7606 - val_acc:
0.6988
Epoch 10/15
100/100 [==============================] - 196s 2s/step - loss: 0.5616 - acc: 0.7696 - val_loss: 0.7376 - val_acc:
0.6988
Epoch 11/15
100/100 [==============================] - 197s 2s/step - loss: 0.5259 - acc: 0.7873 - val_loss: 0.6252 - val_acc:
0.7297
Epoch 12/15
100/100 [==============================] - 197s 2s/step - loss: 0.4871 - acc: 0.7902 - val_loss: 0.6434 - val_acc:
0.7268
Epoch 13/15
100/100 [==============================] - 194s 2s/step - loss: 0.4736 - acc: 0.7963 - val_loss: 0.6786 - val_acc:
0.7272
Epoch 14/15
100/100 [==============================] - 200s 2s/step - loss: 0.4753 - acc: 0.7958 - val_loss: 0.6972 - val_acc:
0.7169
Epoch 15/15
100/100 [==============================] - 194s 2s/step - loss: 0.4461 - acc: 0.8064 - val_loss: 0.6119 - val_acc:
0.7349
```

```
In [11]: plot_training()
```





After 15 epochs of training, training accuracy of 80.64% and validation accuracy of 73.49% was achived.

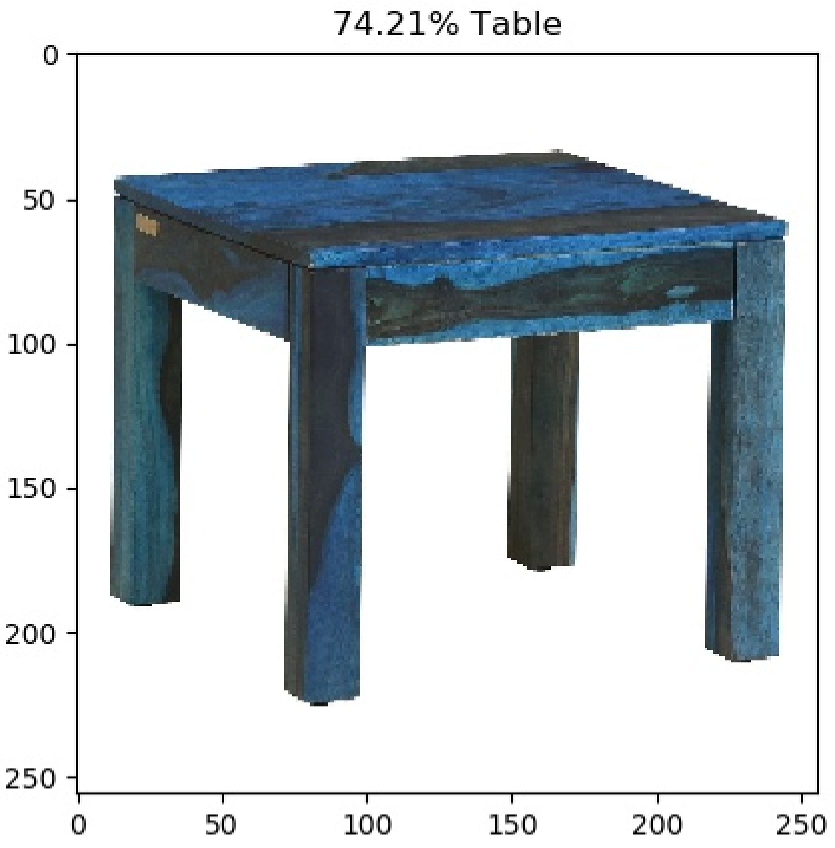
Only the best model among all is saved i.e maximum val accuracy and minimum val loss. `ModelCheckpoint()` is used to do the job.

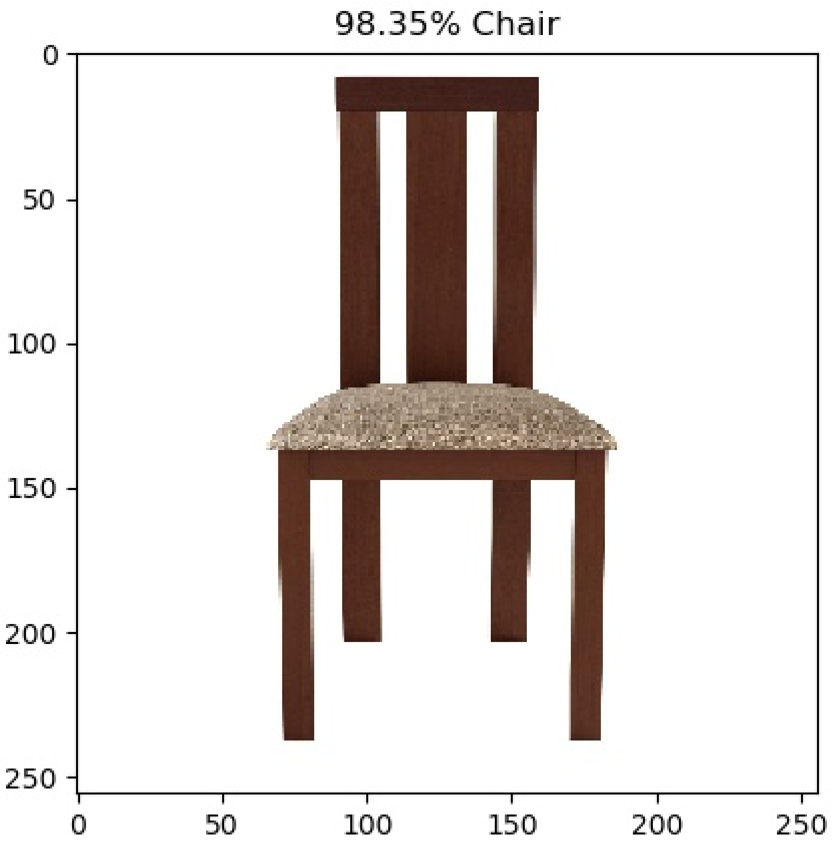
Testing on random images taken from [www.pepperfry.com (http://www.pepperfry.com)](http://www.pepperfry.com)

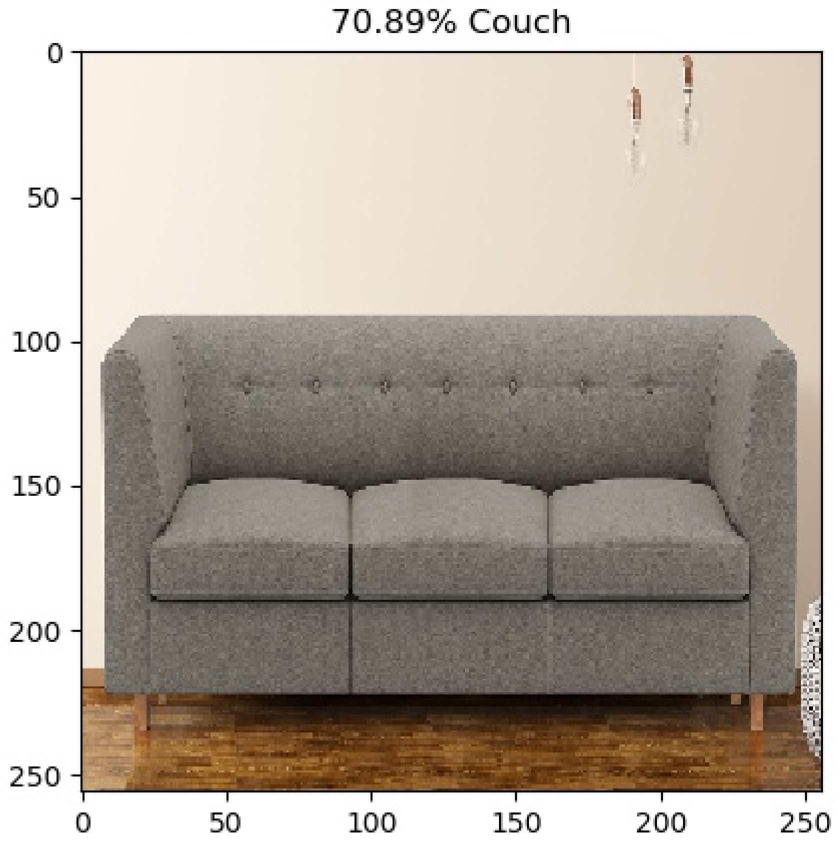
I downloaded random images of 4 objects because they are completely unknown to the CNN.

Path: `CNN_ikea/ikea_cnn_classify_pepperfry.py`

Command: `python ikea_cnn_classify_pepperfry.py`


74.21% Table


98.35% Chair



70.89% Couch
0
50
100
150
200
250
0
50
100
150
200
250

99.84% Wardrobe
0
50
100
150
200
250
0
50
100
150
200
250

To improver the performance of the image classification model we can use Transfer Learning

We can take a CNN that has previously been trained on a very large dataset like ImageNet(1.2M train data) and use it to train smaller datasets. The base model(Inceptionv3, ResNet, etc) is frozen and the fully connected layer is replaced with a custom fully connected layer. We train only the last layer with very few trainable parameters. Therefore, instead of training the network from scratch we can transfer knowledge of another network that has already seen many similar objects that we want to classify.