Individual Project Report

*Name*: Varun Burde (burdeva1@fel.cvut.cz)

*Course*: Individual Project BE3MPROJ8

*Supervisor*: Ing. Michal Reinštein Ph.D. (reinstein.michal@fel.cvut.cz)

*Title*: Deep neural network for classification and segmentation of Google StreetView imagery

February 2019

# **Abstract**

With the advancement computation power and large datasets finally made a huge improvement of Deep neural network leading many widespread applications. One of such application is solving computer vision problems like classification and segmentation. Also Competition like ImageNet Large Scale Visual Recognition Challenge, took the solution to next level , in some cases classification is better than Human .

This report describes the evaluation of pre-trained deep neural network on Google StreetView Images. Pretrained model used are Mask RCNN, Xception , VGG16, VGG19, Inception v3, Inception resnet v2, Resnet50, MobileNet , MobileNet v2, DenseNet, NASNet. Implementation is done in Python using Keras and TesnsorFlow-GPU framework. User interface for the application execution, processing of the input images and visualization of the results is realized using Google Colab with repository in GIT.

A pipeline is created for the task, user provide the parameters like coordinates, heading, field of view for the Google StreetView API, Python script downloads the available images to that location ,Pre Processing of Images to fit into classifier is done, Classification is done on the Images depending on the architecture of the Neural Network.

The results are evaluated based on architecture of different network. Mask RCNN turns out to be best performer among all other Pre-trained model cause of its architecture of performing classification and segmentation side by side. Still there were some misclassification of objects which is visualized in report and the solution which might get the network to get better results.

# **Acknowledgement**

I would like to acknowledge the help of all of those which made this project possible. I would like to express my deep gratitude to my supervisor **Ing. Michal Reinštein, Ph.D** , for his time, patience, guidance and also for allowing the idea to be persuaded originally, and made this project successful. I would also like to thank Jana Zichová who let me to register for this project.

Furthermore, I would like to thank to all those people who works on all open source projects mentioned in reference, and all wonderful people who post the discussion and blogs for all useful learning resources, specially to Stanford open course CS231n: Convolutional Neural Networks for Visual Recognition which. I am also thankful for Google free resources like Google Collab laboratory and Street View API which made the project possible.

**Contents**

**[Abstract](#_Toc862733)** [2](#_Toc862733)

**[Acknowledgement](#_Toc862734)** [3](#_Toc862734)

**[1.](#_Toc862735)****[Introduction](#_Toc862735)** [6](#_Toc862735)

[I. Aims and objective of the Project 6](#_Toc862736)

[II. Overview of the project progress 6](#_Toc862737)

[Pipeline 6](#_Toc862738)

[Problems faced during project 6](#_Toc862739)

[Future Scope 6](#_Toc862740)

[III. Overview of report 6](#_Toc862741)

**[2.](#_Toc862742)****[Related Work](#_Toc862742)** [8](#_Toc862742)

[Keras[1] 11](#_Toc862743)

[Tensorflow[2] 12](#_Toc862744)

[Pre-trained model in Keras 12](#_Toc862745)

[vgg16 and vgg19[11] 12](#_Toc862746)

[resnet50 13](#_Toc862747)

[inception\_v3[13] 13](#_Toc862748)

[Xception[10] 13](#_Toc862749)

[mobilenet\_v2[15] 13](#_Toc862750)

[Densenet[17] 14](#_Toc862751)

[Nasnet[19] 14](#_Toc862752)

[Google Colab [4] 14](#_Toc862753)

[Mask RCNN [5] 15](#_Toc862754)

**[3.](#_Toc862755)****[Implementation](#_Toc862755)** [16](#_Toc862755)

[Google Street view Api 16](#_Toc862756)

[Google Colab 17](#_Toc862757)

**[4.](#_Toc862758)****[Evaluation](#_Toc862758)** [19](#_Toc862758)

**[5.](#_Toc862759)****[Future Scope](#_Toc862759)** [26](#_Toc862759)

**[6.](#_Toc862760)****[Conclusions](#_Toc862760)** [27](#_Toc862760)

**[7.](#_Toc862761)****[References](#_Toc862761)** [28](#_Toc862761)

**[8.](#_Toc862762)****[Appendix](#_Toc862762)** [30](#_Toc862762)

**[List of Figures](#_Toc862763)** [36](#_Toc862763)

# **Introduction**

## Aims and objective of the Project

The aim is to design and implement deep neural network-based solution for online semantic segmentation of Google StreetView images. The proposed software solution should allow the user to request Google StreetView imagery for any given location and output scene description as a list of detected objects (with confidences) and pixel-wise scene segmentation mask. User interface for the application execution, processing of the input images and visualization of the results should be realized using Google Colab and should be easily executable (handling required packages and dependencies on code repositories). Pre-trained existing models should be explored first, thorough experimental evaluation on publicly available datasets should follow. Comparison with related state-of-the-art work is integral part of the project and should be presented in the final report. Recommendation: implementation should be done in Python, using Keras and TensorFlow frameworks.

## Overview of the project progress

Project is done in small steps, there were many unexpected results and problems faced during projects, this section briefly describe about steps of implementation, problem faced and future scope

### Pipeline

1. Use the Google Streetview API to get the Images
2. Preprocess the Images to fit for the model
3. Load the model
4. Get the results
5. Compare the results
6. Finding the misclassification

### Problems faced during project

1. Preprocessing of image
2. Street view API key
3. Fixing the parameter of Google street view API
4. Implementation of script in Google colab

### Future Scope

## Overview of report

This report fully describes the work done in this project along with basic learning of State-of-the-art Neural Networks

1. Introduction: This gives the Introduction of project, its aim and overview of the project work done.
2. Background Research: Analysis of project done in Image segmentation and attempt to define what is state of art. Description of revolution of Computer vision with time.
3. Implementation: The description of how the implementation of the software solution is done in steps
4. Evaluation: The misclassification and other factor which gives the evaluation of the pre-trained model
5. Future Scope: This project will be continued as Master thesis to design and implement the State of art network which will outperform the results of pre-trained model got from the evaluation.
6. Conclusion: Results about how the implementation should take place and what changes should be made to get better results

Appendix – A list of all packages used in the script and the list of all pretrained model used to get the results on the images.

# **Related Work**

The project describes about the two-sub computer vison problem

Image Classification: Classification is process of classifying the object or categorizing objects in some class. In computer vision and machine learning, It’s the task of recognition of object in which pre-determined class it belongs. It generally described by class label along with the confidence level.



Figure 1 Example of Image classification

Semantic Segmentation: In computer vision, Image segmentation is process of partitioning of digital image into multiple segments. Having different segments of image with different boundaries between multiple objects make it easier to analyze and differentiate.



Figure 2Example of Image segmentation

source: https://nicolovaligi.com/deep-learning-models-semantic-segmentation.html

4)Instance segmentation: is the combination of sub problems of Object classification, Object Localization and Semantic segmentation. Combing all together we get bounding box from object localization and a mask pixel wise segmentation [6].

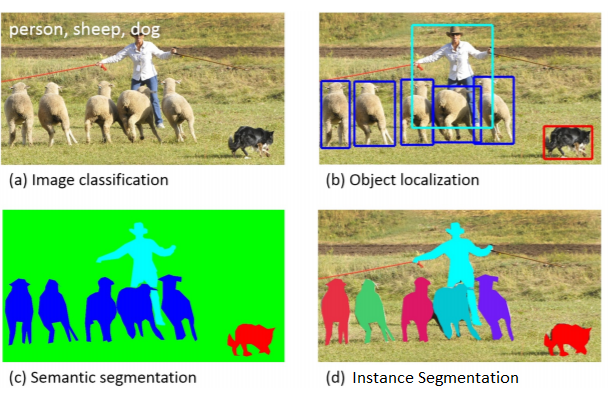


Figure 3 Instance Segmentation

source: http://on-demand.gputechconf.com/gtcdc/2017/presentation/dc7217-abel-brown-deep-learning-object-detection-and-segmentation.pdf

It would be out of scope to how the neural network work for this project, but it would be interesting to know how convolution neural network make this computer vison problem possible. But its Recommended to look at this book of Deep learning [7,8,9].

For a start we can think of every image as a matrix

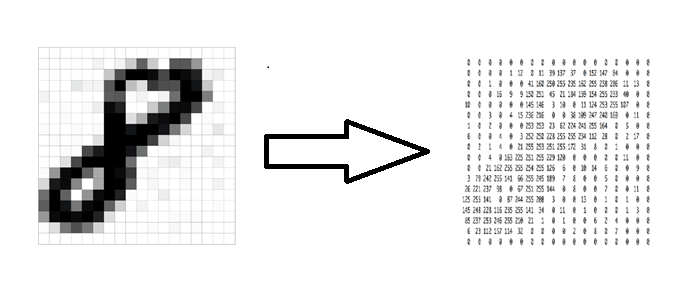


Figure 4 Image as matrix

source: https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/

More typically three channel colored picture can be considered as 3-dimensional array with intensity of color ranging from 0 - 255.

As any other Neural Network Convolutional network have layer inside the deep architecture specifically have convolutional layer, pooling layer

Convolutional layer: what it does is transform input into form of some filter and pass it to other layer

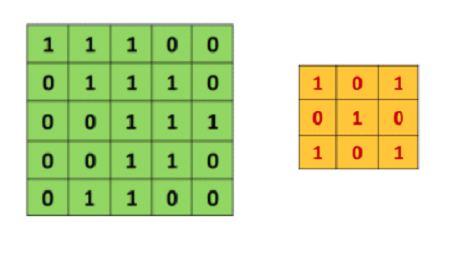


Figure 5 Image and filter matrix

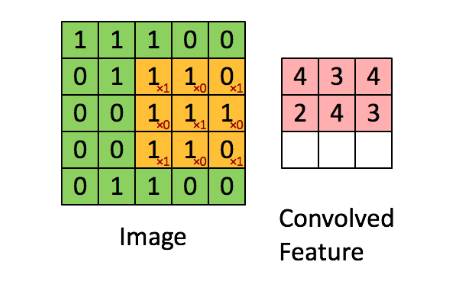


Figure 6 resulted convoluted Feature

source: https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/

This convolved Feature can be imagined as small filter like line, curve, circle etc. detector which become more and more detailed as we go deeper and deeper into networks. These convolved features also called Feature map which is the main detection unit of neural network.

Feature map is controlled by some parameters which can really decide about how the Feature map will be generated and hence the performance of our Neural Network.

Depth: There could be more than one 2d Feature map back to back arranged together. The number of 2d matrices together is the depth.

Stride: Stride is number of pixels we slide our filter matrix over input matrix.

Pooling layer: One of the important concepts of convolution network is pooling, which is form of down sampling. Most common nonlinear function to implement pooling is max pooling in which max value from the frame is taken and move forward.

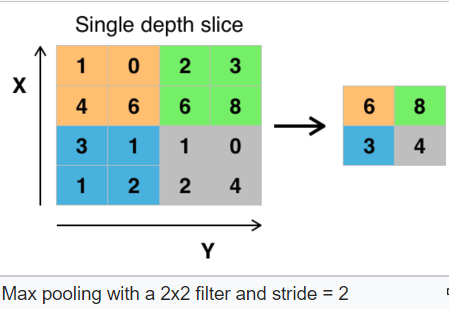


Figure 7 Pooling and Stride

source: https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/

Relu Layer: Relu stands for Rectified Linear unit and it’s a non- linear operation. Its purpose is to replace all negative pixel values from feature map by zero and make the operation less computational expensive. Since most of the real-world data we feed in CNN would be non-linear, so removing the negative pixel value will introduce non-linearity.

Fully connected layer: Its like the same layer from basic perceptron in which every neuron in previous layer is connected to every neuron of next layer. It also consists of SoftMax activation function at the output layer which finally classify the class on their scores.

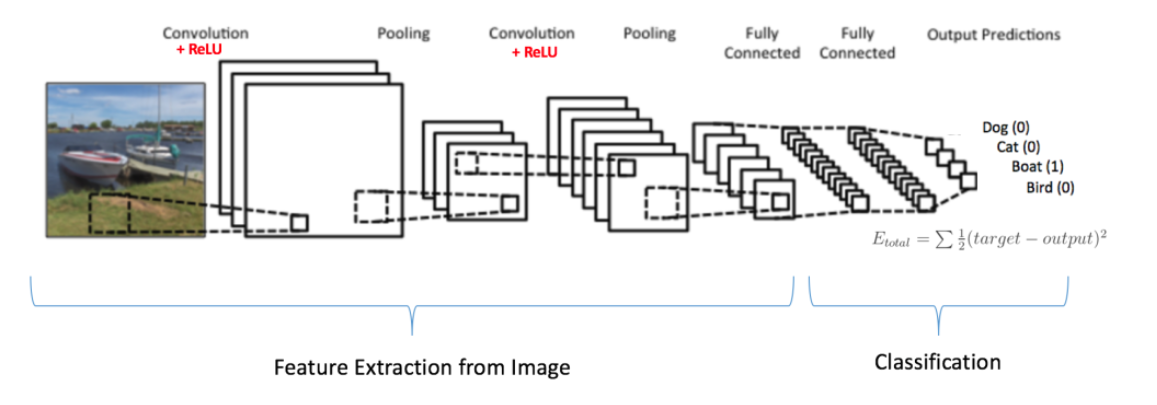


Figure 8 Convolutional network structure

source: https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/

## Keras[1]



Figure 9 Keras Logo

source: https://keras.io/

Keras Is High Level Neural network API, which run on top of TensorFlow, Thaneo or CNTK. It a simple to use open source deep learning library which makes the Implementation of Neural Network much easier [1]. It can be used to create architecture, load/save model, training of neural network in easy way. Also, it has support for CPU and GPU as well TPU in case we use Google Colab. It’s a powerful library with lots of option in tweaking the parameter of layer, loss functions etc.[1]

## Tensorflow[2]



Figure 10 TensorFlow logo

source: https://www.tensorflow.org/

TensorFLow is open-source software library for developed by Google Brain Team. It has well documentation along with example source code to start with [2]. TensorFlow is a symbolic math library for dataflow programming in which data flow is represented by Tensor. The most advantageous feature about TensorFlow is visualization tool TensorBoard which have features like What if and Visualization Graphs which can make a work lot easier in optimizing the learning of Neural network.

## Pre-trained model in Keras

Keras Applications are deep learning models that are made available alongside pre-trained weights. These models can be used for prediction, feature extraction, and fine-tuning. Weights in Network are pre-trained on ImageNet [3].

### vgg16 and vgg19[11]

The 16 and 19 stands for number of weight layer in the network. Due to its depth and fully connected it was really hard to train the network. In order to make training easier it should be trained less weight layer first and then after smaller converged network can be used as Initializer for the larger deeper network and process was called Pre-training.[11]

### resnet50

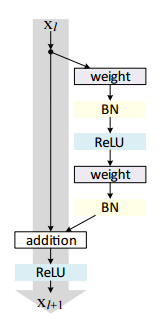


Figure 11Residual model in Resnet source [11]

source https://www.pyimagesearch.com/2017/03/20/imagenet-vggnet-resnet-inception-xception-keras/

In general convolution network several layers are stacked and are trained to for feature filter layer by layer. In residual learning, network will try to learn the residual. Residual is the subtraction of feature learned from the input of the layer. Its architecture is based on micro-architecture which have small building blocks which can be used to construct the network. The collection of micro-architecture building block leads to macro architecture.

### inception\_v3[13]

The Inception model make multi-level feature extractor by computing 1x1, 3x3 and 5x5 convolution within the same module of the network and out put od these filters are stacked along the channel dimension before being fed into the next layer in next layer. [13,11]

### Xception[10]

Xception was proposed by Feancois Chollet the creator of Keras Library. It is an extension of Inception architecture which replaces the standard Inception modules with depth wise separable convolution. [10,11]

### mobilenet\_v2[15]

The purpose of mobile net was to have general purpose computer vison neural network for mobile devices in mind to support classification, detection and more. Mobile net v2 introduces two new features to architecture 1) linear bottleneck between layers and 2)shortcut connection between the bottleneck. See[15] for details.[16]

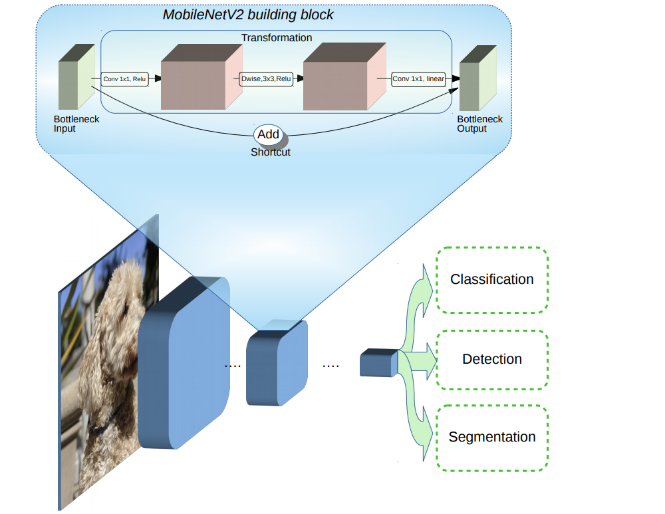


Figure 12 mobile net architecture

source https://ai.googleblog.com/2018/04/mobilenetv2-next-generation-of-on.html

### Densenet[17]

Densely convolution network which connect each layer to every other layer in feed-forward fashion. [17] tells convolution network can be substantially deeper ,more accurate and efficient to train if they contain shorter connections between layers close to the input and those close to the output.[18]. Please refer [17] for further reading.

### Nasnet[19]

Google introduced "AutoML" that automates the design of machine learning model, a controller neural net can propose a “child” model architecture, which can then be trained and evaluated for quality on a task. That feedback is then used to inform the controller how to improve its proposals for the next round. We repeat this process thousands of times — generating new architectures, testing them, and giving that feedback to the controller to learn from. Please refer [19] for detailed explanation. With this method, AutoML was able to find the best layers that work well on CIFAR-10 but work well on ImageNet classification and COCO object detection. These two layers are combined to form a novel architecture, which called “NASNet”[20]]

## Google Colab [4]

One of the important part of task was to realize the developed pipeline in Google Collab. It’s a free research tool for machine learning education and research tool which uses Jupyter notebook Environment which can be rum maximum of 12 hours in single go, which makes it not proper tool for training of Neural network, though all evaluation and testing are done in the Google Colab.

Good thing about Google colab is it have support for deep learing application like Keras, TensorFLow ,Pytorch and Opencv .This make task easier and other dependencies could be easily installed using pip installer .

## Mask RCNN [5]

Mask RCNN is state of the art convolution Neural network which can do Instance segmentation. The working principle of Mask R-CNN was to stitch 2 previously existing state of the art models together and tweak with the linear algebra. The model can be roughly divided into 2 parts — a region proposal network (RPN) and binary mask classifier. [21]

Mask R-CNN (regional convolutional neural network) is a two-stage framework: the first stage scans the image and generates proposals (areas likely to contain an object). And the second stage classifies the proposals and generates bounding boxes and masks. [23]

Backbone: Its standard convolution network that serve as feature extractor, behave pretty much like the other convolution network like Resnet50 and others).

Feature pyramid Network: The Feature Pyramid Network (FPN) was introduced in Mask R-CNN as an extension that can better represent objects at multiple scales. FPN improves the standard feature extraction pyramid by adding a second pyramid that takes the high-level features from the first pyramid and passes them down to lower layers. By doing so, it allows features at every level to have access to both, lower and higher-level features.

Roi Align: In each ROI bin, the value of the four regularly sampled locations are computed directly through bilinear interpolation. Thus, avoid the misaligned problem. [5]

# **Implementation**

## Google Street view Api

The First part of the implementation of this project was to use Google Street view Api to download Images for any given location and store it for further detection and Image processing.

Google street view Api gives user many parameters to work with, example of parameters is given below,

apiargs = {  
 **'location'**:**'50.100471, 14.392636'**,  
 **'size'**: **'640x640'**,  
 **'heading'**: **'0;45;90;135;180;225;270'**,  
 **'fov'**: **'90'**,  
 **'key'**: **XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX'**,  
 **'pitch'**: **'0'**}

Parameters were chosen on basis will give us the best images with orientation, less noise and perfect fit for our model to test. For ex – having pitch to -90 or 90 gives the images which make no sense.



Figure 13Figure 12pitch with -90



Figure 14 pitch with 90

Google street view Api requires Developer key to access the resources by Google, higher resolution images can be downloaded using premium plan.

The task was to find the images which not only consist of normal streets consist of vehicle, traffic lights on which Models were trained but also park, public and general places which can get us the proper evaluation of Detection. To start working with API, it needs to install and import in the script.

## Google Colab

Although the project was developed in local machine, one of the important part of the project was to implement it in Google Colab. Google Colab provide the interactive notebook in ipynb file. It is easy to edit and make changes. Code and text can be added in small cells.

To set up Google Colab ,1st need to login to Google account then choose the python notebook you want to work on. It has support for python 2 and 3. This project is done in python 3. After choosing your preference it take you to interactive notebook to start development. It also allows 3 different kind of learning like GPU , CPU and TPU which can accelerate the learning.

The basic Idea to implement the task was to make the pipeline. First download all the necessary libraries, use the Google street view API to download the images, then load the pre-trained model into the memory. Second to Preprocess the Images to fit in the classifier, Classify the Image and save it .

The Implementation in Google Colab is quite like what we do in out local development in IDE with some small tweaks.

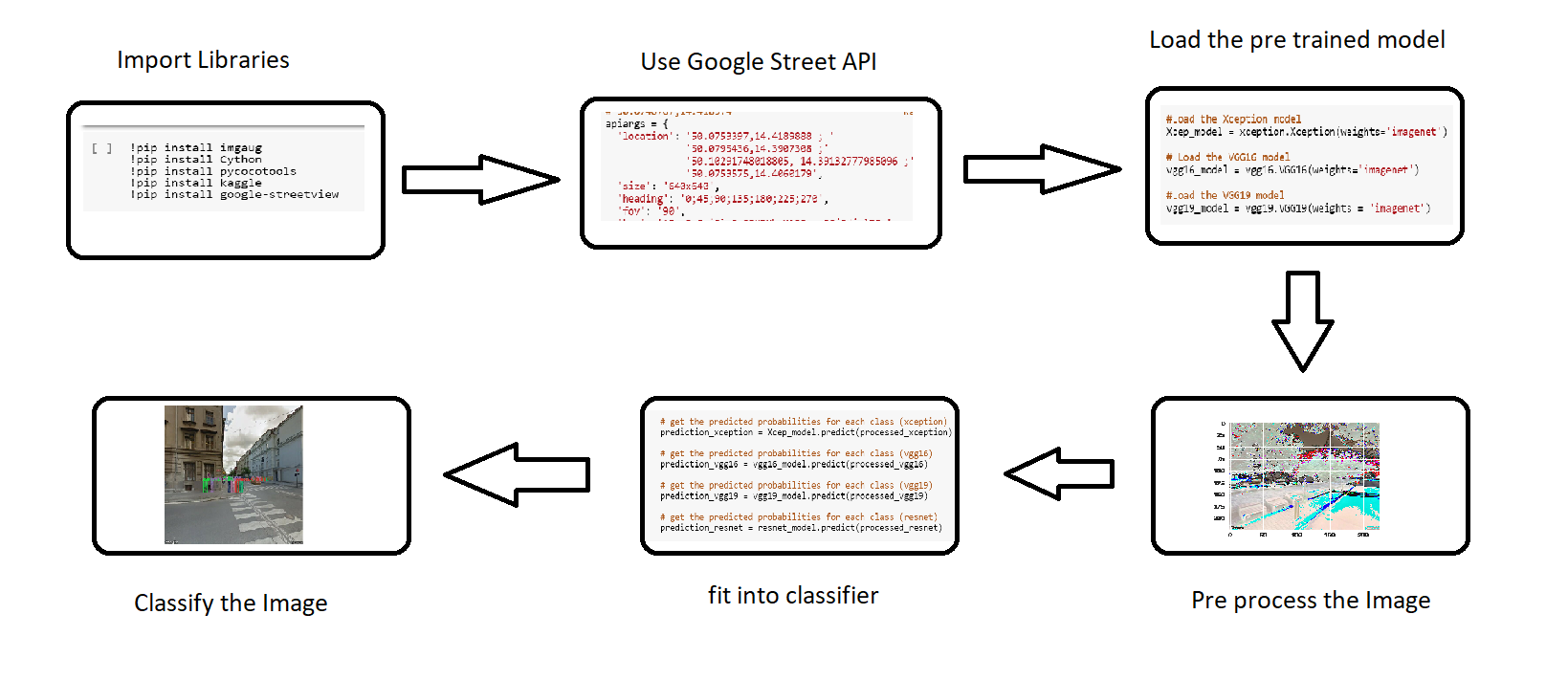


Figure 15 Pipeline

# **Evaluation**

During Random Experimentation out of 27 images in 4 images some objects are found out to be misclassified.

Here are some perfectly classified Images.

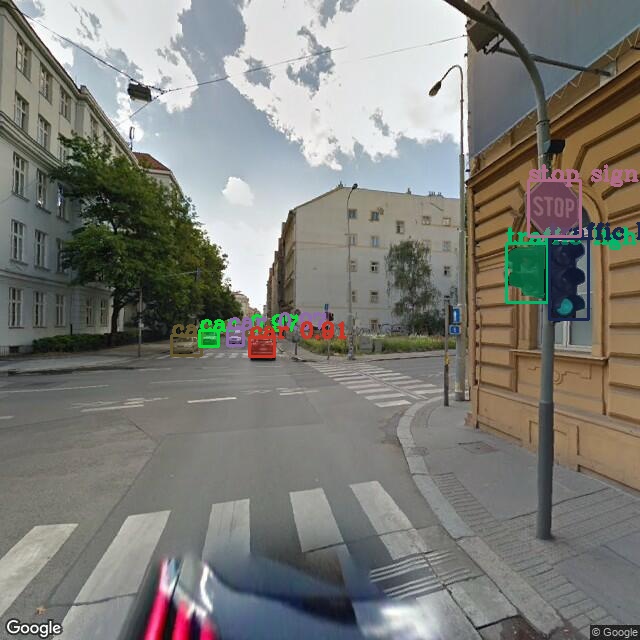
 



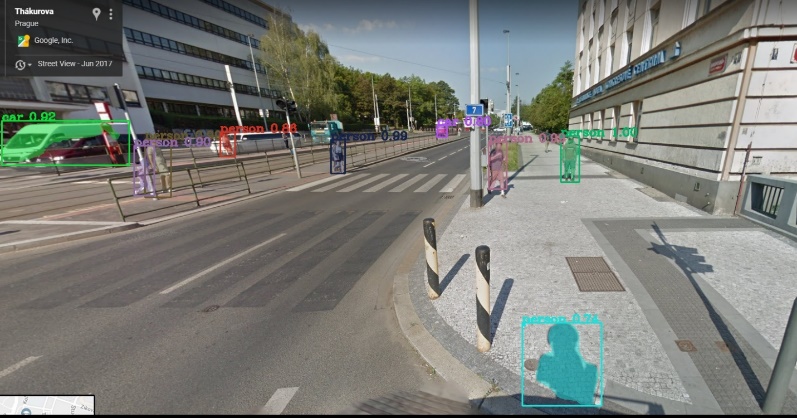
Figure 16 Perfectly classified Images

Here are the Results of some misclassified Images with in Google Street Downloaded Images













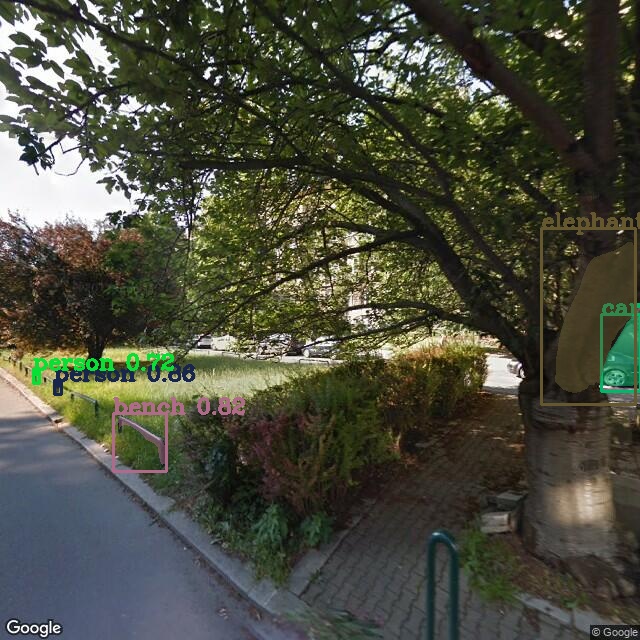
 



Figure 17 Misclassified Objects

Some objects which may be important but not classified



Figure 18 Image with no detection of traffic light which was present in training Data



Figure 19 No classification of Car at Instance

Some Classification from Pretrained model in Keras

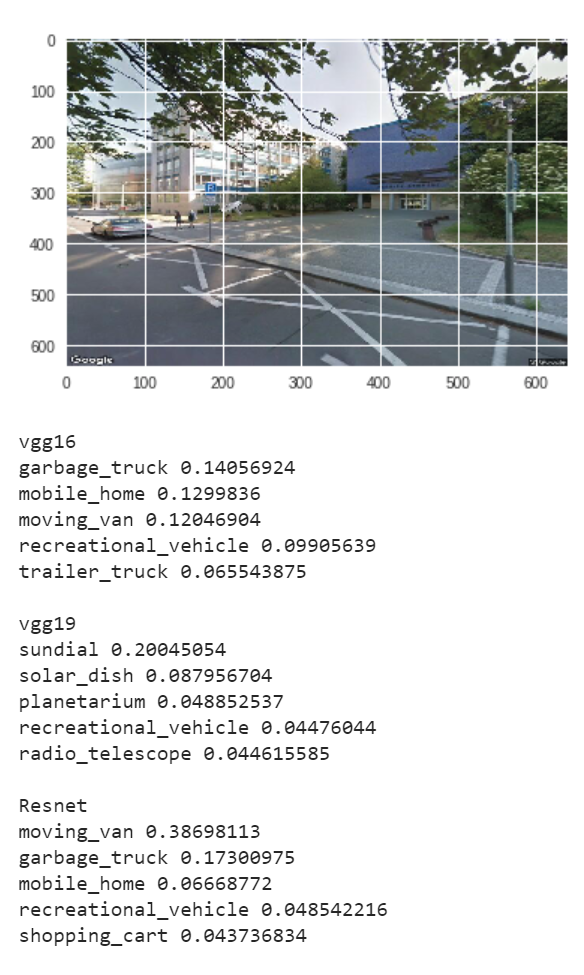
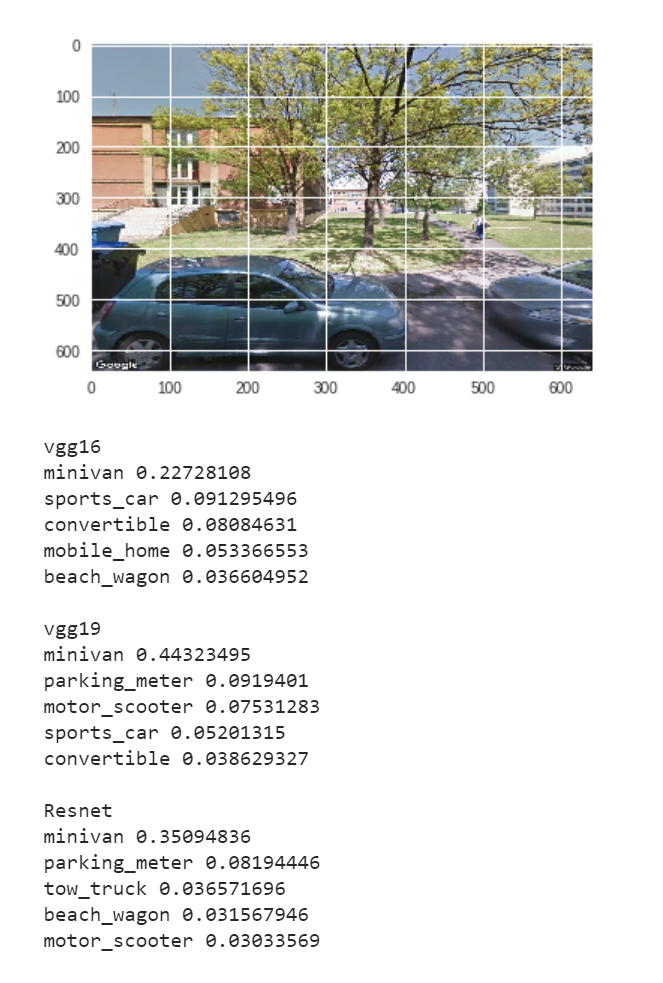


Figure 20 classification by pretrained model

# **Future Scope**

This project will be continued for the master thesis with the aim is to design, implement and experimentally evaluate a deep neural network-based solution for city mapping using Google Street View images. The proposed software solution should allow the user to request Google Street View imagery for any given location specified as geo json, perform analysis and feature extraction using deep neural network(s) and output vectorized description projected and visualized over an underlying map. User interface for the application execution, processing of the input images and visualization of the results should be realized using Google Colab to utilize Google TPUs. Existing pre-trained models should be explored first, thorough experimental evaluation on publicly available datasets should follow. Comparison with related state-of-the-art work is integral part of the work and should be presented in the final thesis. Recommendation: implementation should be done in Python, using Keras and TensorFlow frameworks.

# **Conclusions**

With Aim of project to get software solution which gives the scene description as a list of detected objects and pixel wise scene segmentation in Google street view Imaginary was implemented using Mask RCNN. Due to incapability of not getting the classification of multiple objects in single picture and instance segmentation, the other mentioned neural network model wasn’t able to perform well, though they are still state of the art. The results of classification and segmentation of Google street view was more than expected, Mask RCNN was able to detect roughly 99% of objects and classify them correctly. Though there were few misclassifications and left out objects, but the results were quite satisfying for start. The trees and other fine textured objects were able to hallucinate the network sometimes, but results can be still usable.

The implementation of the project with the Google colab was done properly with small tweak. There were many times when Google colab crashes and restart itself which can be quite irritation but on the other hand it provides high end resources like GPU and TPU for free which is quite good for Machine Learning Enthusiast.

This project will be extended as the mater thesis with more scope as described in the future scope section with the complete software solution with combination of data science and user Interface.

# **References**

*[1] Keras Documentation:* [*https://keras.io/*](https://keras.io/)

*[2] TensoFlow Documentation and Tutorials:* [*https://www.tensorflow.org/tutorials*](https://www.tensorflow.org/tutorials)

*[3] ImageNet: http://www.image-net.org/*

*[4] Google Colab FAQ:* [*https://research.google.com/colaboratory/faq.html*](https://research.google.com/colaboratory/faq.html)

*[5] He, K., Gkioxari, G., Dollar, P., & Girshick, R. (2017). Mask R-CNN. Proceedings of the IEEE International Conference on Computer Vision, 2017–Octob, 2980–2988. https://doi.org/10.1109/ICCV.2017.322*

*[6]* *Brown, A. (2017). Introduction to Object Detection & Image Segmentation.*

*[7]* *[13] GOODFELLOW, Ian; BENGIO, Yoshua and COURVILLE, Aaron. Deep Learning.*

*MIT Press, 2016.* [*http://www.deeplearningbook.org*](http://www.deeplearningbook.org)*.*

*[8]* [*https://towardsdatascience.com/how-do-artificial-neural-networks-learn-773e46399fc7*](https://towardsdatascience.com/how-do-artificial-neural-networks-learn-773e46399fc7)

*[9]* [*https://medium.com/@RaghavPrabhu/understanding-of-convolutional-neural-network-cnn-deep-learning-99760835f148*](https://medium.com/@RaghavPrabhu/understanding-of-convolutional-neural-network-cnn-deep-learning-99760835f148)

*[10] Chollet, F. (2017). Xception: Deep learning with depthwise separable convolutions. Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, 2017–January, 1800–1807. https://doi.org/10.1109/CVPR.2017.195*

*[11]* *https://www.pyimagesearch.com/2017/03/20/imagenet-vggnet-resnet-inception-xception-keras/*

*[12]* *https://www.quora.com/What-is-the-deep-neural-network-known-as-%E2%80%9CResNet-50%E2%80%9D*

*[13]* *Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., & Wojna, Z. (2015). Rethinking the Inception Architecture for Computer Vision.* [*https://doi.org/10.1109/CVPR.2016.308*](https://doi.org/10.1109/CVPR.2016.308)

*[14]* Chollet, F. (2017). Xception: Deep learning with depthwise separable convolutions. *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, *2017*–*Janua*, 1800–1807. https://doi.org/10.1109/CVPR.2017.195

*[15]* *Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., & Chen, L.-C. (2018). MobileNetV2: Inverted Residuals and Linear Bottlenecks. https://doi.org/10.1134/S0001434607010294*

*[16]* [*https://ai.googleblog.com/2018/04/mobilenetv2-next-generation-of-on.html*](https://ai.googleblog.com/2018/04/mobilenetv2-next-generation-of-on.html)

*[17]* Huang, G., Liu, Z., Van Der Maaten, L., & Weinberger, K. Q. (2017). Densely connected convolutional networks. *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, *2017*–*January*, 2261–2269. https://doi.org/10.1109/CVPR.2017.243

*[18]* *https://towardsdatascience.com/densenet-2810936aeebb*

*[19]* Zoph, B., & Shlens, J. (n.d.). Learning Transferable Architectures for Scalable Image Recognition.

*[20]* *https://ai.googleblog.com/2017/11/automl-for-large-scale-image.html*

*[21]* *https://medium.com/@ajayuppili/mask-r-cnn-explained-7f82bec890e3*

*[22]* *Ren, S., He, K., Girshick, R., & Sun, J. (n.d.). Faster R-CNN : Towards Real-Time Object Detection with Region Proposal Networks, 1–14.*

*[23]* *https://engineering.matterport.com/splash-of-color-instance-segmentation-with-mask-r-cnn-and-tensorflow-7c761e238b46*

*[24]* *http://cs231n.stanford.edu/*

# **Appendix**

Code for pretrained model Masked RCNN

*Automatically generated by Colaboratory.  
  
Original file is located at  
 https://colab.research.google.com/drive/16AIGaLGc\_1lHwez0NJ8WF0AnX0VoiDbg  
  
# Install libraries  
"""*!pip install imgaug  
!pip install Cython  
!pip install pycocotools  
!pip install kaggle  
!pip install google-streetview  
  
**"""# \*\* Google Street api Setup\*\*"""  
  
import** google\_streetview.api  
**import** google\_streetview.helpers  
  
*# 50.10291748018805, 14.39132777985096 dejvice   
# 50.0795436,14.3907308 Strahov  
# 50.0746767,14.418974 Karlovo namesti*apiargs = {  
 **'location'**: **'50.0753397,14.4189888 ; '  
 '50.0795436,14.3907308 ;'  
 '50.10291748018805, 14.39132777985096 ;'  
 '50.0753575,14.4060179'**,  
 **'size'**: **'640x640'**,  
 **'heading'**: **'0;45;90;135;180;225;270'**,  
 **'fov'**: **'90'**,  
 **'key'**: **'XXXXXXXXXXXXXXXXXXXXXXXXXXX'**,  
 **'pitch'**: **'0'**}  
*# Get a list of all possible queries from multiple parameters*api\_list = google\_streetview.helpers.api\_list(apiargs)  
  
*# Create a results object for all possible queries*resultsg = google\_streetview.api.results(api\_list)  
  
*# Preview results  
#resultsg.preview()  
  
# Download images to directory 'downloads'*resultsg.download\_links(**'StreetImages'**)  
  
*# Save metadata*resultsg.save\_metadata(**'metadata.json'**)  
  
**"""# Clone Repo"""**!git clone https://github.com/matterport/Mask\_RCNN  
  
**import** os   
os.chdir(**'Mask\_RCNN/samples'**)  
  
**"""# Prepare Module"""  
  
import** sys  
**import** random  
**import** math  
**import** numpy **as** np  
**import** skimage.io  
**import** matplotlib  
**import** matplotlib.pyplot **as** plt  
**from** os **import** listdir  
**from** os.path **import** isfile, join  
  
*# Root directory of the project*ROOT\_DIR = os.path.abspath(**"../"**)  
  
*# Import Mask RCNN*sys.path.append(ROOT\_DIR) *# To find local version of the library***from** mrcnn **import** utils  
**import** mrcnn.model **as** modellib  
**from** mrcnn **import** visualize  
*# Import COCO config*sys.path.append(os.path.join(ROOT\_DIR, **"samples/coco/"**)) *# To find local version***import** coco  
  
*# %matplotlib inline   
  
# Directory to save logs and trained model*MODEL\_DIR = os.path.join(ROOT\_DIR, **"logs"**)  
  
*# Local path to trained weights file*COCO\_MODEL\_PATH = os.path.join(ROOT\_DIR, **"mask\_rcnn\_coco.h5"**)  
*# Download COCO trained weights from Releases if needed***if not** os.path.exists(COCO\_MODEL\_PATH):  
 utils.download\_trained\_weights(COCO\_MODEL\_PATH)  
  
*# Directory of images to run detection on  
#IMAGE\_DIR = 'downlaods'*IMAGE\_DIR = **'/content/StreetImages'**print(IMAGE\_DIR)  
print(ROOT\_DIR)  
  
**"""# Some helper fuctions"""  
  
def** random\_colors(N):  
 np.random.seed(1)  
 colors = [tuple(255 \* np.random.rand(3)) **for** \_ **in** range(N)]  
 **return** colors  
   
**def** display\_instances(image, boxes, masks, ids, names, scores):  
 *"""  
 take the image and results and apply the mask, box, and Label  
 """* n\_instances = boxes.shape[0]  
 colors = random\_colors(n\_instances)  
  
**"""# Create Inference Object"""  
  
class** InferenceConfig(coco.CocoConfig):  
 *# Set batch size to 1 since we'll be running inference on  
 # one image at a time. Batch size = GPU\_COUNT \* IMAGES\_PER\_GPU* GPU\_COUNT = 1  
 IMAGES\_PER\_GPU = 1  
  
config = InferenceConfig()  
config.display()  
  
*# Create model object in inference mode.*model = modellib.MaskRCNN(mode=**"inference"**, model\_dir=MODEL\_DIR, config=config)  
  
*# Load weights trained on MS-COCO*model.load\_weights(COCO\_MODEL\_PATH, by\_name=**True**)  
  
*# COCO Class names  
# Index of the class in the list is its ID. For example, to get ID of  
# the teddy bear class, use: class\_names.index('teddy bear')*class\_names = [**'BG'**, **'person'**, **'bicycle'**, **'car'**, **'motorcycle'**, **'airplane'**,  
 **'bus'**, **'train'**, **'truck'**, **'boat'**, **'traffic light'**,  
 **'fire hydrant'**, **'stop sign'**, **'parking meter'**, **'bench'**, **'bird'**,  
 **'cat'**, **'dog'**, **'horse'**, **'sheep'**, **'cow'**, **'elephant'**, **'bear'**,  
 **'zebra'**, **'giraffe'**, **'backpack'**, **'umbrella'**, **'handbag'**, **'tie'**,  
 **'suitcase'**, **'frisbee'**, **'skis'**, **'snowboard'**, **'sports ball'**,  
 **'kite'**, **'baseball bat'**, **'baseball glove'**, **'skateboard'**,  
 **'surfboard'**, **'tennis racket'**, **'bottle'**, **'wine glass'**, **'cup'**,  
 **'fork'**, **'knife'**, **'spoon'**, **'bowl'**, **'banana'**, **'apple'**,  
 **'sandwich'**, **'orange'**, **'broccoli'**, **'carrot'**, **'hot dog'**, **'pizza'**,  
 **'donut'**, **'cake'**, **'chair'**, **'couch'**, **'potted plant'**, **'bed'**,  
 **'dining table'**, **'toilet'**, **'tv'**, **'laptop'**, **'mouse'**, **'remote'**,  
 **'keyboard'**, **'cell phone'**, **'microwave'**, **'oven'**, **'toaster'**,  
 **'sink'**, **'refrigerator'**, **'book'**, **'clock'**, **'vase'**, **'scissors'**,  
 **'teddy bear'**, **'hair drier'**, **'toothbrush'**]  
  
**"""# Prediction and Visualization"""***# Load image from the images folder*file\_names = next(os.walk(IMAGE\_DIR))[2]  
procimg = np.empty(len(file\_names), dtype=object)  
*#print(file\_names)***for** n **in** range(0, len(file\_names)):  
 **if** file\_names[n] == **'metadata.json'**:  
 **break**;  
 image = skimage.io.imread(os.path.join(IMAGE\_DIR, file\_names[n]))  
  
 *# Run detection* results = model.detect([image], verbose=1)  
  
 *# Visualize results* r = results[0]  
 procimg = display\_instances(image, r[**'rois'**], r[**'masks'**], r[**'class\_ids'**],   
 class\_names, r[**'scores'**])  
 visualize.display\_instances(image, r[**'rois'**], r[**'masks'**], r[**'class\_ids'**],   
 class\_names, r[**'scores'**])

Code for Pretrained model available in KERAS

*Automatically generated by Colaboratory.  
  
Original file is located at  
 https://colab.research.google.com/drive/1cT0avReSwNfDYV0EfAgND81s\_-JjaXWc  
  
# Loading all model available in Keras  
"""***from** keras.applications **import** xception,vgg16,vgg19,resnet50,inception\_v3,inception\_resnet\_v2,mobilenet,mobilenet\_v2,densenet,nasnet  
**import** numpy **as** np  
  
*#Load the Xception model*Xcep\_model = xception.Xception(weights=**'imagenet'**)  
  
*# Load the VGG16 model*vgg16\_model = vgg16.VGG16(weights=**'imagenet'**)  
  
*#Load the VGG19 model*vgg19\_model = vgg19.VGG19(weights = **'imagenet'**)  
  
*# Load the Inception\_V3 model*inception\_model = inception\_v3.InceptionV3(weights=**'imagenet'**)  
  
*# Load the Inception\_resner\_v2 model*inception\_res = inception\_resnet\_v2.InceptionResNetV2(weights=**'imagenet'**)  
  
*# Load the ResNet50 model*resnet\_model = resnet50.ResNet50(weights=**'imagenet'**)  
  
*# Load the MobileNet model*mobilenet\_model = mobilenet.MobileNet(weights=**'imagenet'**)  
  
*# Load the MobileNet\_v2 model*mobilenet\_v2 = mobilenet\_v2.MobileNetV2(weights=**'imagenet'**)  
  
*# Load the DenseNet model*densenet\_model = densenet.DenseNet201(weights=**'imagenet'**)  
  
*# Load the NASNet model*nasnet\_model = nasnet.NASNetLarge(weights=**'imagenet'**)  
  
**"""# \*\* Google Street api Setup\*\*"""**!pip install google-streetview  
**import** google\_streetview.api  
**import** google\_streetview.helpers  
  
*# 50.10291748018805, 14.39132777985096 dejvice   
# 50.0795436,14.3907308 Strahov  
# 50.0746767,14.418974 Karlovo namesti*apiargs = {  
 **'location'**: **'50.0753397,14.4189888 ; 50.0795436,14.3907308 ;50.10291748018805, 14.39132777985096'**,  
 **'size'**: **'640x640'**,  
 **'heading'**: **'0;45;90;135;180;225;270'**,  
 **'fov'**: **'90'**,  
 **'key'**: **'XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX'**,  
 **'pitch'**: **'0'**}  
  
*# Get a list of all possible queries from multiple parameters*api\_list = google\_streetview.helpers.api\_list(apiargs)  
  
*# Create a results object for all possible queries*resultsg = google\_streetview.api.results(api\_list)  
  
*# Preview results  
#resultsg.preview()  
  
# Download images to directory 'downloads'*resultsg.download\_links(**'StreetImages'**)  
  
*# Save metadata*resultsg.save\_metadata(**'metadata.json'**)  
  
**"""# importing the packages"""  
  
from** keras.preprocessing.image **import** load\_img  
**from** keras.preprocessing.image **import** img\_to\_array  
**from** keras.applications.imagenet\_utils **import** decode\_predictions  
**import** matplotlib.pyplot **as** plt  
**import** os  
**import** skimage.io  
**from** skimage.transform **import** resize  
**from** PIL **import** Image  
**from** skimage.viewer **import** ImageViewer  
*# %matplotlib inline*IMAGE\_DIR = **'/content/StreetImages'  
  
"""# Prediction and visulaizeing the results"""***# Load image from the images folder*file\_names = next(os.walk(IMAGE\_DIR))[2]  
procimg = np.empty(len(file\_names), dtype=object)  
  
*#print(file\_names)***for** n **in** range(0, len(file\_names)):  
 print(**' '**)  
   
 **if** file\_names[n] == **'metadata.json'**:  
 **break**;  
   
 *#Load the image* image1 = skimage.io.imread(os.path.join(IMAGE\_DIR, file\_names[n]))  
 image = Image.open(os.path.join(IMAGE\_DIR, file\_names[n]))  
   
 *#processing the Image* image\_resized = image.resize((224,224), Image.ANTIALIAS)  
 numpy\_image = img\_to\_array(image\_resized)  
 image\_batch = np.expand\_dims(numpy\_image, axis=0)  
   
 *# prepare the image for the VGG model* processed\_xception = xception.preprocess\_input(image\_batch.copy())  
 processed\_vgg16 = vgg16.preprocess\_input(image\_batch.copy())  
 processed\_vgg19 = vgg19.preprocess\_input(image\_batch.copy())  
 processed\_resnet = resnet50.preprocess\_input(image\_batch.copy())   
 processed\_inception = inception\_v3.preprocess\_input(image\_batch.copy())  
 processed\_inception\_resnet = inception\_resnet\_v2.preprocess\_input(image\_batch.copy())  
 processed\_mobilenet = mobilenet.preprocess\_input(image\_batch.copy())  
 processed\_densenet = densenet.preprocess\_input(image\_batch.copy())  
 processed\_nasnet = nasnet.preprocess\_input(image\_batch.copy())  
   
*# # get the predicted probabilities for each class (xception)  
# prediction\_xception = Xcep\_model.predict(processed\_xception)  
   
 # get the predicted probabilities for each class (vgg16)* prediction\_vgg16 = vgg16\_model.predict(processed\_vgg16)  
   
 *# get the predicted probabilities for each class (vgg19)* prediction\_vgg19 = vgg19\_model.predict(processed\_vgg19)  
   
 *# get the predicted probabilities for each class (resnet)* prediction\_resnet = resnet\_model.predict(processed\_resnet)  
  
 *#Display Image* plt.imshow(image,interpolation=**'nearest'**, aspect=**'auto'**)  
 plt.show()  
 skimage.io.imshow(image1)  
 plt.show()  
 plt.imshow(image\_resized)  
 plt.show()  
 plt.imshow(np.uint8(numpy\_image))  
 plt.show()  
 plt.imshow(np.uint8(image\_batch[0]))  
 plt.show()  
 plt.imshow(np.uint8(processed\_xception[0]))  
 plt.show()   
 plt.imshow(np.uint8(processed\_vgg16[0]))  
 plt.show()   
 plt.imshow(np.uint8(processed\_vgg19[0]))  
 plt.show()   
 plt.imshow(np.uint8(processed\_resnet[0]))  
 plt.show()   
 plt.imshow(np.uint8(processed\_inception[0]))  
 plt.show()   
 plt.imshow(np.uint8(processed\_inception\_resnet[0]))  
 plt.show()   
 plt.imshow(np.uint8(processed\_mobilenet[0]))  
 plt.show()   
 plt.imshow(np.uint8(processed\_densenet[0]))  
 plt.show()   
 plt.imshow(np.uint8(processed\_nasnet[0]))  
 plt.show()   
 *# convert the probabilities to class labels  
 # We will get top 5 predictions which is the default* print(**'Xception'**)  
 **for** prediction **in** decode\_predictions(prediction\_xception)[0]:  
 print(prediction[1], prediction[2])  
   
 print(**' '**)  
 print(**'vgg16'**)  
   
 **for** prediction **in** decode\_predictions(prediction\_vgg16)[0]:  
 print(prediction[1], prediction[2])  
   
 print(**' '**)  
 print(**'vgg19'**)  
   
 **for** prediction **in** decode\_predictions(prediction\_vgg19)[0]:  
 print(prediction[1], prediction[2])  
 print(**' '**)  
   
 print(**'Resnet'**)  
 **for** prediction **in** decode\_predictions(prediction\_resnet)[0]:  
 print(prediction[1], prediction[2])  
   
 print(**' '**)  
  
 print(**'Xception'**)  
 **for** prediction **in** decode\_predictions(prediction\_xception)[0]:  
 print(prediction[1], prediction[2])  
   
 print(**' '**)

# **List of Figures**

[Figure 1 Example of Image classification 8](#_Toc852388)

[Figure 2Example of Image segmentation 8](#_Toc852389)

[Figure 3 Instance Segmentation 9](#_Toc852390)

[Figure 4 Image as matrix 9](#_Toc852391)

[Figure 5 Image and filter matrix 10](#_Toc852392)

[Figure 6 resulted convoluted Feature 10](#_Toc852393)

[Figure 7 Pooling and Stride 11](#_Toc852394)

[Figure 8 Convolutional network structure 11](#_Toc852395)

[Figure 9 Keras Logo 11](#_Toc852396)

[Figure 10 TensorFlow logo 12](#_Toc852397)

[Figure 11Residual model in Resnet source [11] 13](#_Toc852398)

[Figure 12 mobile net architecture 14](#_Toc852399)

[Figure 13Figure 12pitch with -90 16](#_Toc852400)

[Figure 14 pitch with 90 17](#_Toc852401)

[Figure 15 Pipeline 18](#_Toc852402)

[Figure 16 Perfectly classified Images 20](#_Toc852403)

[Figure 17 Misclassified Objects 24](#_Toc852404)

[Figure 18 Image with no detection of traffic light which was present in training Data 24](#_Toc852405)

[Figure 19 No classification of Car at Instance 25](#_Toc852406)

[Figure 20 classification by pretrained model 25](#_Toc852407)