Vilnius University

Institute of Data Science and Digital Technologies

Cognitive Computing Group

PhD of Science in Informatics – 09P

Research subject:

Deep Learning for Image Recognition

Research Supervisor:

Prof. Dr. Olga Kurasova

Research consultant:

Dr. Doc. Ernestas Filatovas

Convolutional Neural Networks for object detection in optical satellite imagery

---

Povilas Gudzius

2018 10 22

Table of Contents

Research aim

Design an accurate Convolutional Neural Network (CNN) architecture for object detection in satellite imagery.

Research objectives

The objective of this research is to: explore and develop advanced methodologies of image recognition in satellite imagery such as Convolutional Neural networks to derive a computationally efficient way to process large amounts of pixels-based data collected from high-resolution optical satellites and detect objects with >90% accuracy.

Practical application: Financial services industry. Investment decisions in the financial services industry requires precise, accurate, high quality and timely data. Remote sensing data such as satellite imagery can provide an extremely powerful material information supporting investment decisions. Number of objects (e.g. vehicles) per given Area of Interest (AOI) can become financially material information.

Research tasks

1. Upon the full academic literature review, explore the list of conventional Convolutional Neural Network architectures applied in object recognition of ImageNet;

2. Conduct a comparison between architectures and define the significant features of each architecture with a particular relevance to satellite imagery dataset;

3. Based on this analysis suggest two most appropriate CNN architectures for object detection and segmentation problems in optical satellite imagery;

4. Build the selected CNN architectures in Python to be trained on the Satellite imagery dataset using GPU's on Google Cloud Platform environment;

5. Conduct a practical real-life experiment of the selected CNN networks and conclude with the architecture most suitable for object detection in satellite imagery.

Satellite imagery

Satellite imagery is produced by commercial, public and private satellites operating in 2,000km above the Earth at the Lower Earth Orbit (LEO). Key differences and implications between using satellite imagery compared to the on-the-ground imagery are: wavelength, perspective, resolution and availability. All of these factors represent and important challenges when trying to adapt existing object recognition methodologies used in ImageNet to this particular type of data. We provide further details on those challenges and limitations below**.** We will also consider them when evaluating an optimal CNN architecture of this problem.

Unlike the other computer vision problems where you are given RGB or grayscale images, when processing satellite data most frequently we have to deal with the satellite data that is given both in visual and lower frequency regions. Wavelength frequency (RGB and M Band) and non-traditional data set is yet to be fully utilized via the Machine Learning techniques. Instead of just collecting RGB images we are able to use RGB, M Band and A band range of waves to our training sets as per Figure 1 [13].

On one hand it carries more information, on another it is not really obvious how to use this extra data properly. This unique dataset property might represent one of the research challenges/problems for further investigation.

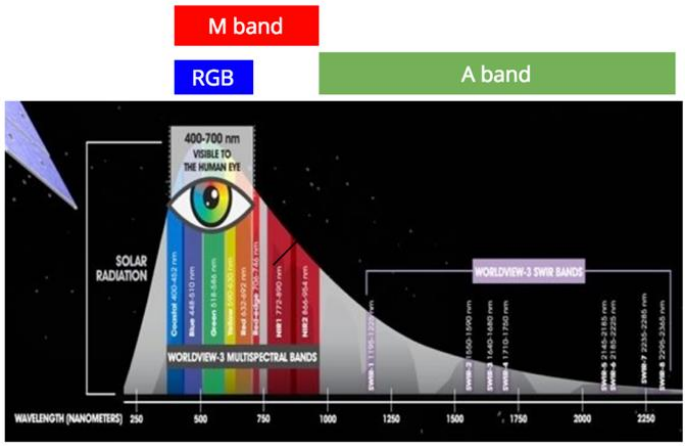


Figure 1 – Rage of multispectral bands that can be acquired by the satellite [13]

Figure below gives four examples of the data range that can be adopted for our data and training set: RGB, P-Band, M-Band and A-band. We are yet to explore the most efficient set of algorithms that can take a full advantage of this unique imagery properties to maximise the accuracy of classification exercise [14]. It's a rich multi-dimensional training set:

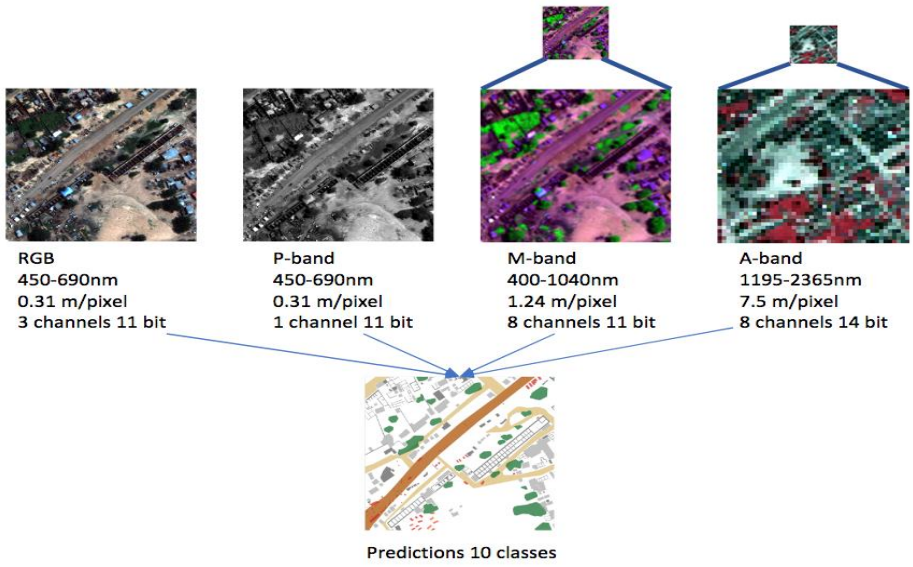


Figure 2 – example of spectral bands used in object recognition

2) Perspective - that means that the perspective images are always from the top-down perspective and object parameters on the ground will always be subject to how they look from above [14]. Figure 3 illustrates the point visually:

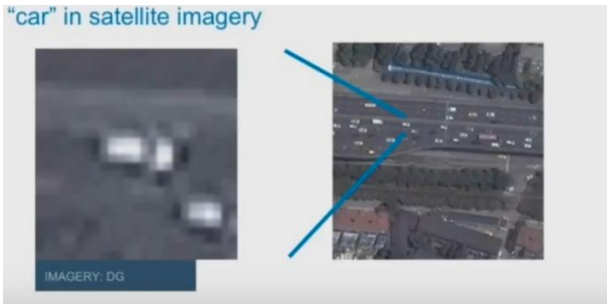


Figure 3 – Object of ImageNet vs Satellite imagery

3) Resolution of satellite imagery varies from 5m per pixel to 30cm per pixel (ultra-high-resolution imagery), compared to ImageNet where you have millions of pixels

within 30cm range, the parameters of deep learning models and training of them are also impacted [B16];

Machine learning methods

As per table below, we can see that CNN methods are providing a promising >90% accuracy in multispectral optical satellite imagery already [32]. Experiment conducted by Langvist et al has investigated 9 different conventional computer vision techniques and applied those models for satellite imagery data sate in particular. As we can see from the example below (Figure 6), only OBIA, Knowledge-based method and Single/Multi CNN's were methods that provided >90% accuracy.

Method	Overall Accuracy (%)	Data	Categories
Fuzzy C means [73]	68.9%	Aerial image, laser scanning	4 (vegetation, buildings, roads, open areas)
Segmentation and classification tree method [52]	70%	Multispectral aerial imagery	5 (water, pavement, rooftop, bare ground, vegetation)
Classification Trees and TFP [48]	74.3%	Aerial image	4 (building, tree, ground, soil)
Segmentation and classification rules [51]	75.0%	Multispectral aerial imagery	6 (building, hard standing, grass, trees, bare soil, water)
Region-based GeneSIS [66]	89.86%	Hyperspectral image	9 (asphalt, meadows, gravel, trees, metal sheets, bare soil, bitumen, bricks, shadows)
OBIA [31]	93.17%	Aerial orthophotography and DEM	7 (buildings, roads, water, grass, tree, soil, cropland)
Knowledge-based method [50]	93.9%	Multispectral aerial imagery, laser scanning, DSM	4 (buildings, trees, roads, grass)
Single CNN ($L = 1$, $N = 1$, $m = 25$)	$90.02\% \pm 0.14\%$	Multispectral orthophotography imagery, DSM	5 (vegetation, ground, road, building, water)
Multiple CNNs ($L = 1$, $N = 4$, $m = [15, 25, 35, 45]$)	$94.49\% \pm 0.04\%$	Multispectral orthophotography imagery, DSM	5 (vegetation, ground, road, building, water)

Figure 6 - Comparison table of classification accuracy (%) compared to other methods [32]:

This accuracy, however, is presented for the large objects such as buildings, roads, vegetation fields and water reservoirs. Additionally, considering the fact that this experiment conducted by Langvist was done using low resolution satellite imagery (3m per pixel) and our objective of this research will be using much higher-resolution, this provides motivation that CNN's might also be an appropriate deep learning methodology to detect smaller objects is high-resolution satellite imagery (30cm) such as vehicles, ships or planes. We further continue our analysis focusing purely on Convolutional Neural Networks architectures in search for suitable architectures and topologies for object detection in satellite imagery.

Convolutional Neural Networks

Convolutional neural networks were first proposed in 1980 by Fukushima [32] (called NeoCognitron) and then refined by LeCun [32]. Convolutional networks [B3], also known as convolutional neural networks from or CNNs, are a specialized kind of neural network for processing data that has a known grid-like topology. Examples include time-series data, which can be thought of as a 1-D grid taking samples at regular time intervals, and image data, which can be thought of as a 2-D grid of pixels. Convolutional networks have been tremendously successful in practical applications.

The name “convolutional neural network” indicates that the network employs a mathematical linear operation called convolution. Convolution operation is the dot product of two vectors in the given matrix of number, in our case it’s pixel brightness. If vectors a = [a_1, a_2, …, a_n] and b = [b_1, b_2, …, b_n] the convolution can be defined as [B4]:

$$\mathbf{a} \cdot \mathbf{b} = \sum_{i=1}^{n} a_i b_i = a_1 b_1 + a_2 b_2 + \cdots + a_n b_n$$

Convolutional neural networks are simply neural networks that use convolution in place of general matrix multiplication in at least one of their layers (B3). The above computation is applied to the matrix of pixels (convolution filter) and the output is delivered to the “destination pixel” as illustrated in the Figure 8 below:

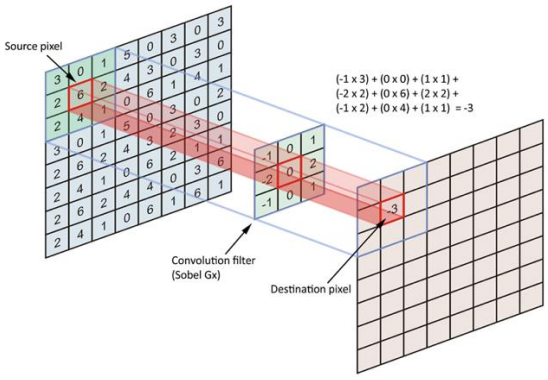
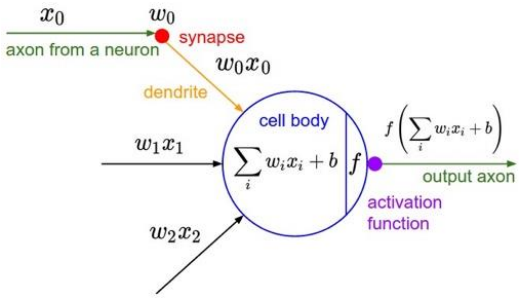


Figure 8 – illustration of convolution operation

Recently, CNNs have been shown to be successful in object recognition [32] object detection, scene parsing [33] and scene classification [33] tasks. For the purpose of this research we will predominantly focus on CNN's application to object recognition problems. We will then further investigate its applicability to a particular type of dataset: optical satellite imagery.

Convolutional Neural Network (CNN) architecture

Neurons within the CNN are comprised of axon from neuron (input), activation function and output axon:



The main difference between Convolutional Neural Networks and basic Neural Network is that neurons between layers are not fully connected. In the architecture of CNN, neurons are limited to their "receptive fields" also called Kernel or Filter. As the name suggests, the C layer computes a spatial convolution of the outputs of the C-1'th layer (in this example it is an input layer):

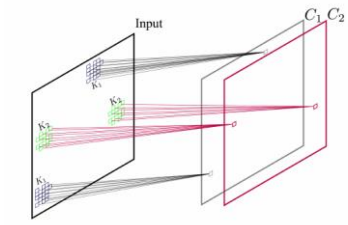


Figure 10 – Kernel K1 and K2 convoluting an output to C1 and C2 respectively [B4]

Kernel K1 and K2 are operating a convolution operation of their respective “receptive fields” and providing an output to the next layer, C1 and C2. Convolution operation in CNN is defined as “dot product” of the matrix of values imported from Kernel’s receptive field of the input layer [B3]. All neurons of the convolutional layer share the identical feature matrix (or weights, or filter, or Kernel) and are of an identical dimensionality.

These constraints reduce sharply the number of parameters to learn and therefore enables CNN’s to extract features in a significantly more computationally efficient way. In addition to that, another key advantage of CNN-based algorithms is that they do not require prior manual feature extraction, thus resulting in higher generalization capabilities.

To illustrate the parameters of CNN architecture we are using the example diagram in the Figure 1 below. In this example we are processing 227x227x3 dimensional matrix of pixels from RGB image using Convolutional Neural Network (CNN). This network consists of five convolutional layers, each followed by a pooling layer and three fully connected layers:

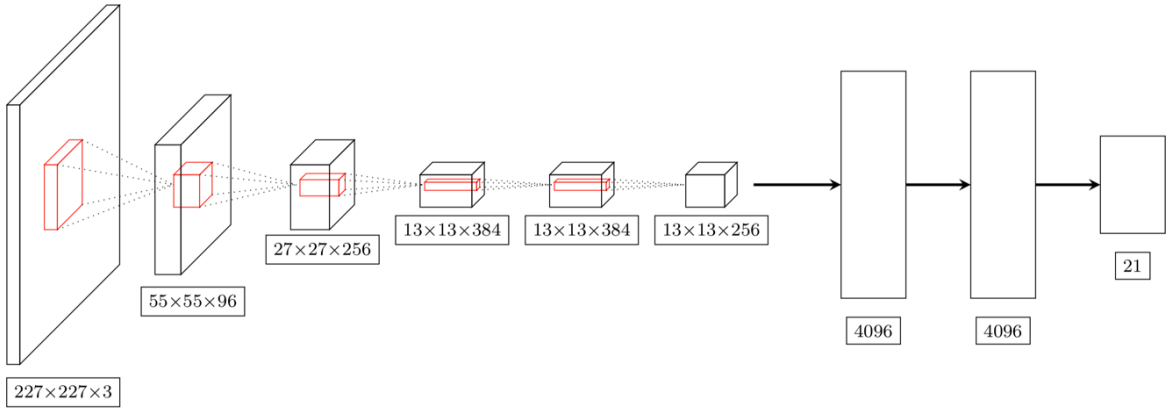


Fig. 1. The CaffeNet architecture used in this work. The boxes show the size of each feature layer and, for fully connected layers, the size of the output. Most receptive fields are 3×3, maxpool layers are not shown. [B24]

Traditional Convolutional Neural Network components: Input layer, Convolutional blocks for feature extraction (Conv, ReLU, Pool), Classification (Flatten, Fully connected, Softmax):

1) Convolutional layers: are the most important ones. They compute the convolution of the input image with the weights of the network. Neurons in the first hidden layer view only a small image window, and learn low-level features. Those in deeper layers view (indirectly) larger portions of the image, and are able to learn more expressive features by combining low-level ones. Each layer is characterized by a few hyper-parameters: the number of filters to learn, their spatial support, the stride between different windows and an optional zero-padding which controls the size of the layer output.

This formula can be directly applied to our kernel operation using the Convolution Theorem ([B4]:

$$\text{feature map} = \text{input} \otimes \text{kernel} = \sum_{y=0}^{\text{columns}} \left(\sum_{x=0}^{\text{rows}} \text{input}(x-a, y-b)\, \text{kernel}(x, y) \right) = \mathcal{F}^{-1}\left(\sqrt{2\pi}\mathcal{F}[\text{input}]\mathcal{F}[\text{kernel}] \right)$$

Figure 14 – Convolution theorem formula [B4]

This equation is the 2D discrete convolution theorem for discrete image data.
Here x denotes a convolution operation, F denotes the Fourier transform, F^(-1) the inverse Fourier transform, and 2p^(-2) is a normalization constant.

2) Pooling layers: reduce the size of the input layer through some local non-linear operations, for example maxpooling max(), so as to reduce the number of parameters to learn and provide some translation invariance. The most relevant hyper-parameters are the support of the pooling window and the stride between different windows.

Max pooling will take the most significant parts of our receptive field matrix which will narrow down the matrix and reduce computational complexity of our model.

3) Normalization (or activation) layers: inspired by inhibition schemes present in the real neurons of the brain, aim at improving generalization. Examples of activation functions are: Sigmoid, tanh, ReLu, Leaky ReLU, Maxout, ELU. Activation functions allow our model to learn non-linear functions

4) Fully-connected layers: are typically used as the last few layers of the network. By removing constraints, they can better summarize the information conveyed by lower-level layers in view of the final decision. Despite full connectivity, their complexity is still affordable thanks to the previous size- reducing layers.

Hyperparameters of CNN to tune your network: (*good Andrew Karpathy – working content on CNN hyperparameters)*

- Activation functions (Sigmoid, tanh, ReLu, Leaky ReLU, Maxout, ELU)
- Number of features
- Number of Neurons
- Window Stride pace (1-pixel, 2-pixel stride)
- Filter configuration (size, dimensions, number of filters)
- Zero padding (how big on corners of the image, or mirror padding, other padding)
- Pooling (alternative to Max pooling, other pooling)
- Lully connected layer setup (Softmax/Sigmoid/derivative)
- Regularization Strength, Learning rate
- Weight initialization
- Optimisation: (SGD, SGD+momentum, Adagrad, RMSprop, Adam – all have learning rate as a hyperparameter, step-up-decay

U-net architecture experiment:

"Semantic Segmentation in Satellite imagery of light vehicles using U-net"

Following from the review of the most commonly used CNN architectures, we have derived that Semantic Segmentation is the appropriate methodology to apply to our research problem. Given the limitations and specific parameters of satellite imagery [B4], segmentation task will allow us to detect vehicles in 30cm resolution by deploying polygon mask layer on top of the object within the given coordinates.

Objectives of the experiment:

1) Replicate the theoretical design of the structure of the U-net network as described in the "U-net" section under "CNN architectures";

2) Build a U-net Convolutional Neural Neutwork in Python using machine learning libraries such as Keras, Pytorch, Tensorflow, Pandas, Scipy and Numpy;

3) Setup the Google Cloud Platform "Virtual Machine" GPU's to be able to train, optimize and validate our models;

4) Acquire a publicly available high-resolution satellite imagery and using QGIS geospatial data management software manually label at least 20,000 polygons of objects (light vehicles/cars) for training and validation;

5) Train the U-net on the RGB band training set and test various hyper-parameters to discover the most accurate configuration. Conduct experiments with hyperparameters such as: epochs, training size, batch size and activation function;

6) Use Jaccard metric to measure the precise pixel accuracy of real-life object polygons vs the predicted ones. Additionally, use the manual object count metric to assess the object count accuracy;

7) Analyze, compare and conclude the findings of the experiments.

Step 1 - *Replicate the theoretical design of the structure of the U-net network as described in the "U-net" section under "CNN architectures"*

Our fully convolutional model is based on the U-Net architectures [B14], where low-level feature maps are combined with higher-level ones, which enables precise localization. This type of network architecture was especially designed to effectively solve image segmentation problems. Due to this reason this architecture has been chosen as a default architecture for our research problem. Visual representation of the structure is depicted in the Figure below:

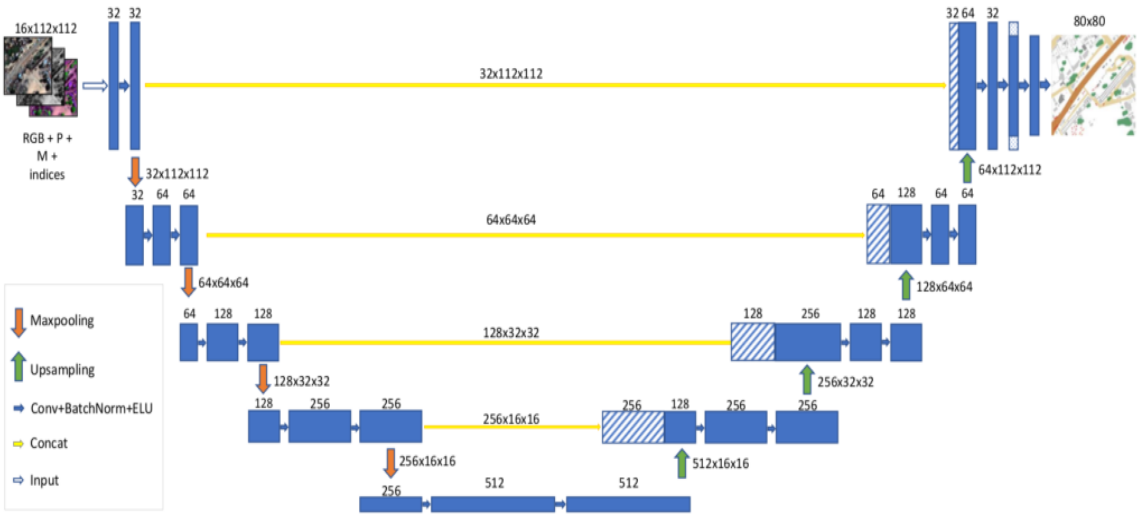


Figure 17 – Visual representation of a standard U-net architecture [B12]

Typical convolutional neural network architecture involves increasing the number of feature maps (channels) with each max pooling operation. In our network we decided to keep a constant number of 64 feature maps throughout the network. This choice was motivated by two observations [B14]:

1) Firstly, we can allow the network to lose some information after the down-sampling layer because the model has access to low level features in the up-sampling path;

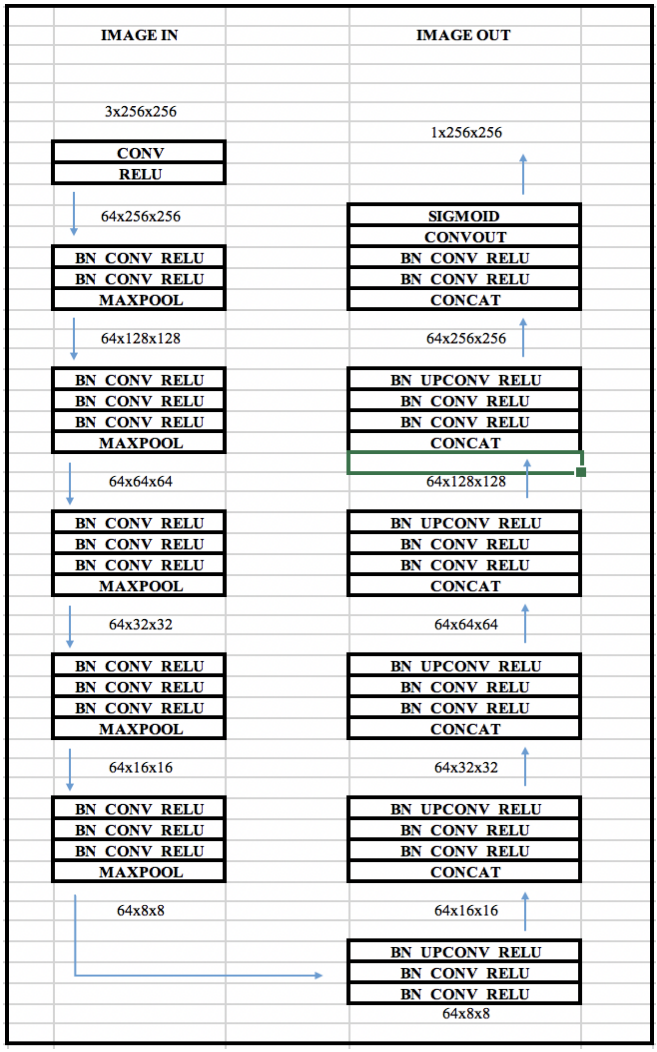
2) Secondly, in satellite images there is no concept of depth or high-level 3D objects to understand, so a large number of feature maps in higher layers may not be critical for good performance.

In the tables below, we provide the breakdown of created network architecture and breakdown per layer:

Figure 8: Layers description

CONV	**Conv2d**
	in_channels
	out_channels
	kernel_size=(3,3)
	stride=(1,1)
	padding=(1,1)
UPCONV	**ConvTranspose2d**
	in_channels
	out_channels
	kernel_size=(3,3)
	stride=(1,1)
	padding=(1,1)
	output_padding=(1,1)
BN	**BatchNorm2d**
	in_channels
	momentum=0.01
	affine=false
MAXPOOL	**MaxPool2d**
	kernel_size=(2,2)
	stride=(2,2)
	padding=(0,0)
BN_CONV_RELU	**BN**
	CONV
	RELU
BN_UPCONV_RELU	**BN**
	UPCONV
	RELU
CONVOUT	**Conv2d**
	in_channels
	out_channels=1
	kernel_size=(1,1)
	stride=(1,1)
	padding=(0,0)

Figure 9: U-net architecture



As per tables above, system architecture is pretty straightforward with a standard components such as CONV, RELU, MaxPool, CONCAT and SIGMOID activation.

STEP 2 - *Build a U-net Convolutional Neural Neutwork in Python using machine learning libraries such as Keras, Pytorch, Tensorflow, Pandas, Scipy and Numpy*

In order to be able to experiment with various CNN architectures and hyperparameters we had to build our own U-net in practice. As a starting point we have used the U-net framework from Kaggle competition “DSTL feature detection” [B15] and then customized the U-net to our specific requirements. Libraries on Python with TensorFlow backend deployed: Keras, Pylab, Numpy, Pandas, Sklearn and Scipy:

```
import matplotlib.pyplot as plt
from keras.models import model_from_json
from pylab import plot, show, subplot, specgram, imshow, savefig
import numpy as np
import cv2
import pandas as pd
from shapely.wkt import loads as wkt_loads
import tifffile as tiff
import os, sys
import random
from keras.models import Model
from keras.layers import Input, merge, Conv2D, MaxPooling2D, UpSampling2D, Reshape, core, Dropou
from keras.layers import concatenate
from keras.optimizers import Adam
from keras.callbacks import ModelCheckpoint, LearningRateScheduler, TensorBoard
from keras import backend as K
from sklearn.metrics import jaccard_similarity_score
import skimage.transform
import skimage.morphology
from shapely.geometry import MultiPolygon, Polygon
import shapely.wkt
import shapely.affinity
from collections import defaultdict
import scipy
from scipy import ndimage
```

Figure 18: Machine learning libraries, Jupyter Notebook screenshot

```
def get_unet_2():
    inputs = Input((N_bands, ISZ, ISZ))
    conv1 = Conv2D(32, (3, 3), activation='relu', padding='same', data_format="channels_first")(
    conv1 = Conv2D(32, (3, 3), activation='relu', padding='same', data_format="channels_first")(
    pool1 = MaxPooling2D(pool_size=(2, 2), data_format="channels_first")(conv1)

    conv2 = Conv2D(64, (3, 3), activation='relu', padding='same', data_format="channels_first")(
    conv2 = Conv2D(64, (3, 3), activation='relu', padding='same', data_format="channels_first")(
    pool2 = MaxPooling2D(pool_size=(2, 2), data_format="channels_first")(conv2)

    conv3 = Conv2D(128, (3, 3), activation='relu', padding='same', data_format="channels_first")
    conv3 = Conv2D(128, (3, 3), activation='relu', padding='same', data_format="channels_first")
    pool3 = MaxPooling2D(pool_size=(2, 2), data_format="channels_first")(conv3)

    conv4 = Conv2D(256, (3, 3), activation='relu', padding='same', data_format="channels_first")
    conv4 = Conv2D(256, (3, 3), activation='relu', padding='same', data_format="channels_first")
    pool4 = MaxPooling2D(pool_size=(2, 2), data_format="channels_first")(conv4)

    conv5 = Conv2D(512, (3, 3), activation='relu', padding='same', data_format="channels_first")
    conv5 = Conv2D(512, (3, 3), activation='relu', padding='same', data_format="channels_first")

    up6 = concatenate([UpSampling2D(size=(2, 2), data_format="channels_first")(conv5), conv4], a
    conv6 = Conv2D(256, (3, 3), activation='relu', padding='same',  data_format="channels_first"
    conv6 = Conv2D(256, (3, 3), activation='relu', padding='same',  data_format="channels_first"

    up7 = concatenate([UpSampling2D(size=(2, 2), data_format="channels_first")(conv6), conv3], a
    conv7 = Conv2D(128, (3, 3), activation='relu', padding='same',  data_format="channels_first"
    conv7 = Conv2D(128, (3, 3), activation='relu', padding='same',  data_format="channels_first"

    up8 = concatenate([UpSampling2D(size=(2, 2), data_format="channels_first")(conv7), conv2], a
    conv8 = Conv2D(64, (3, 3), activation='relu', padding='same',  data_format="channels_first")
    conv8 = Conv2D(64, (3, 3), activation='relu', padding='same',  data_format="channels_first")
```

Figure 18: Code of the U-net layers, Jupyter Notebook screenshot

STEP 3 - *Setup the Google Cloud Platform GPU's on the Virtual Machine to be able to train, optimize and validate our models*

STEP 4 - *Acquire a publicly available high-resolution satellite imagery and using QGIS geospatial data management software manually label at least 20,000 polygons of objects (light vehicles/cars) for training and validation*

STEP 5 - *Train the U-net on QGIS training set and test various hyper-parameters*

In order to find the right architecture and configuration of the U-net at total number of 17 experiments were conducted applying various combinations of hyperparameters. Further details of the experiments' findings are provided in the "Step 7" section on this paper. Purpose of the experiments were to test the accuracy of the prediction by re-adjusting the configuration of hyperparameters below:

- Batch size
- Pixel window mosaic "ISZ"
- Number of Epochs
- Training and Validation size
- Activation functions: ReLu vs LeakyRelu
- Gradient methods (e.g. Adam)

```
img_count = 9   # po kiek pav imsim mokymui 3 - reiskia 3x3; 5 - 5x5
img_size = 1300  # koks apmokymui skirtu pav. dydis pikseliais
N_Cls = 1 #10
N_bands = 3
N_ToPredict = 200
epochs = 20
train_size=20000
val_size=0.25 #procentide dalis valicacijai
ba_size = 128  # was 64
ISZ = 160 # mazo paveiksliuko dydis (prazioje buvo 160)
Part_Proc= 0.02 #proporcine dalis kiek visame paveikliuke pikleliu priklauso reikiamiems objekta
pixel_threshold = 100 # minimla number of pixels on the prediceted mask for the car
Data_Folder = 'AOI_train_81';
weights_spacenet = Data_Folder + '/trained_models/unet_1_jk_' + str(epochs) + 'ep_' + str(train_
model_file = Data_Folder + '/trained_models/model_spacenet_'+ str(epochs) + 'ep_' + str(train_si
temp_weights = Data_Folder +'/trained_models/unet_tmp_' +str(epochs) + 'ep_' + str(train_size) +
#inDir = 'C:\\Users\\ernes\\source\\repos\\Spacenet_8bands\\Spacenet_8bands'
#GS_s = pd.read_csv('AOI_train_81/input_data/grid_sizes_81_new.csv', names=['ImageId','Xmin', 'X
#DF_s = pd.read_csv('AOI_train_81/input_data/combined_polygons_81.csv')
Log_Dir = Data_Folder +'/log_data/'
GS_s = pd.read_csv(Data_Folder +'/input_data/grid_sizes_81_new.csv', names=['ImageId','Xmin', 'X
DF_s = pd.read_csv(Data_Folder +'/input_data/combined_polygons_81.csv')
Big_x = Data_Folder + '/composed_data/x_trn_3_%d'
Big_y = Data_Folder + '/composed_data/y_trn_3_%d'
#inDir = '../input'
#DF = pd.read_csv(inDir + '/train_wkt_v4.csv')
#GS = pd.read_csv(inDir + '/grid_sizes.csv', names=['ImageId', 'Xmax', 'Ymin'], skiprows=1)
#SB = pd.read_csv(os.path.join(inDir, 'sample_submission.csv'))

smooth = 1e-12
print("done")
```

Figure 28: Screenshot of the python code from our test Jupyter Notebook framework

STEP 6 - *Use Jaccard metric to measure the precise pixel accuracy of real-life object polygons vs the predicted ones. Additionally, use the manual object count metric to assess the object count accuracy.*

The prediction for vehicle recognition was evaluated independently using Average Jaccard Index (also known in literature as Intersection-over-Union). The Jaccard index [B17] is a measure of overlap or similarity between two contour areas, and is defined as:

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}$$

Where *J* is the Jaccard index, *A* and *B* are two overlapping contour areas [B16]

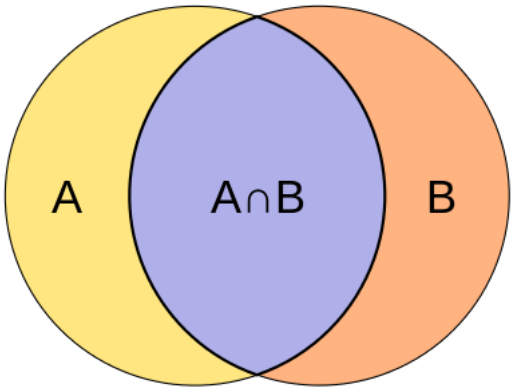


Figure 29 - visualization of Jaccard metric [B16].

Jaccard metric is a great measure to access the precise accuracy of the semantic segmentation-based methods for object detection. It is, however, predominantly utilized in the medical applications [B12] where for example cell types are to be determined in the imagery generated by a microscope [B12].

Limitation 1: Jaccard metric allows to examine the prices accuracy of prediction and we use it for the purpose of fine-tuning our U-net. However, the end objective is not to match pixels perfectly, but is simply to detect an object. Jaccard metric might be too sensitive in our case. We simply need to intersect more than 51% of pixels of a single polygon to detect an object.

Limitation 2: polygons in the training set have been measured manually and might not perfectly fit the contours of an object. We estimate at least 20% of pixels to be inaccurately marked in the training as a human-error and therefore even predicted accurately might provide lower accuracy rate of the network.

In order to address these two limitations of Jaccard metric we will generate a manual "Number of cars per given Area of Interest (AOI)" index as best real-life approximation of our U-net model accuracy. We've selected a random sample of satellite imagery with 288 objects and conducted a manual car count as per below:

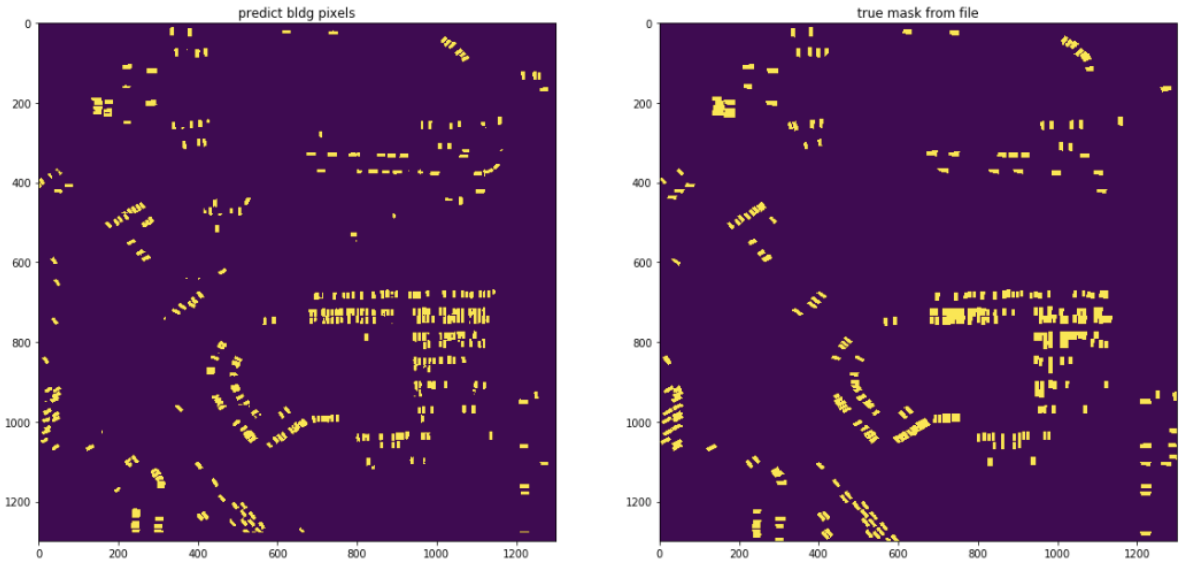


Figure 29: Predicted image (left), training set (right), screenshot from our Jypyter Notebook random imagery output after the U-net with ~64% accuracy was trained

STEP 7 - *Analyze, compare and conclude the findings of the experiment*

Total number of 17 experiments where conducted. Each experiment consisted of training the neaural network with exactly the same training set data and fine-tuning its configuration, and hyperparameters. Most significant findings are analyzed below:

	Epochs	Training	Validation	Batch Size	"ISZ" mozaic size	Part_proc (proc c	Activation function	Pixel threshold	Jaccard Accuracy
U-net Experiment 8	20	25,000 objects	0.25	256	160x160 pixel	0.02	ReLU	100	ResourceExhaustedError
U-net Experiment 9	160	25,000 objects	0.25	64	160x160 pixel	0.02	ReLU	100	62.18%
U-net Experiment 10	20	25,000 objects	0.25	64	80x80 pixel	0.02	ReLU	100	50.02% (4 times faster)
U-net Experiment 11	20	25,000 objects	0.25	64	80x80 pixel	0.02	Leaky_ReLU	100	Leaky_config
U-net Experiment 12	20	25,000 objects	0.25	256	80x80 pixel	0.02	ReLU	100	47.99%
U-net Experiment 13	50	25,000 objects	0.25	512	80x80 pixel	0.02	ReLU	100	51.49%
U-net Experiment 14	50	25,000 objects	0.25	32	200x200 pixel	0.02	ReLU	100	MEMORY ERROR
U-net Experiment 15	50	25,000 objects	0.25	512	160x160 pixel	0.02	ReLU	100	ResourceExhaustedError
U-net Experiment 16	50	25,000 objects	0.25	128	160x160 pixel	0.01	ReLU	100	MEMORY ERROR

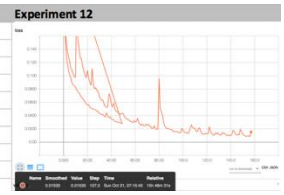


Figure (table) 27: example matrix of U-nets experiments and findings

Finding number 1: Winning U-net. the highest Jaccard metric validation accuracy reached was 63.90% using 50 number of epochs for the backpropagation and the training set. Winning U-net parameters: Batch_size: 128, pixel frame: 160x160, 2% polygon qualification to the training set, ReLU as an activation function at each of the Conv layer, 50 epochs of training:

	Epochs	Training	Validation	Batch Size	"ISZ" mozaic size	Part_proc (proc c	Activation function	Pixel threshold	Jaccard Accuracy
U-net Experiment 17	50	25,000 objects	0.2(5)	128	160x160 pixel	0.02	ReLU	100	63.90%

Figure 28: Winning U-net configuration

Finding number 2: Real-life accuracy. As discussed in “Step 6” of the experiment, Jaccard metric is not the most representative metric give our research problem. After manual object count examination, we have established that 63.90% Jaccard metric accuracy is equivalent to an impressive 94.09% of object count measure:

SpaceNet 30cm high-res optical satellite imagery (RGB)	*1.5km2*
Number of manually market car polygons AOI	*288*
Number of objects that U-net model predicted accurately	*271*
Number of objects were detected	*17*
Number of objects detected yet nor marked in the training set	*28*
Model accuracy	***94.09%***
Jaccard Index of best performed U-net	*63.90%*

Figure (table): results of “Manual Object Count Examination”

There is no study that examines the human error in satellite imagery object classification/detection task, however for the indicative purposes we can compare with the study conducted by Russakovsky in 2014 [B9] which concludes that human-level accuracy/estimated human classification error is 5.1% by a trained human annotator in ImageNet classification. That results in 94.9% accuracy. Even though ImageNet

dataset classification task, by nature is more complex, we could state that our designed U-net CNN architecture provides a close-to-human accuracy for object count problems.

Finding number 3: Above human performance. Interestingly enough, in addition to finding number 2, we can argue that our U-net architecture is some cases provided "better-than human accuracy" since it has detected 28 objects within the given manual test sample that were not discovered by human annotator. Meaning that vehicles that were partially covered by a tree, or a shadow or otherwise hard to detect in satellite imagery by human eye – were in fact detected by our algorithm. Illustrated example:

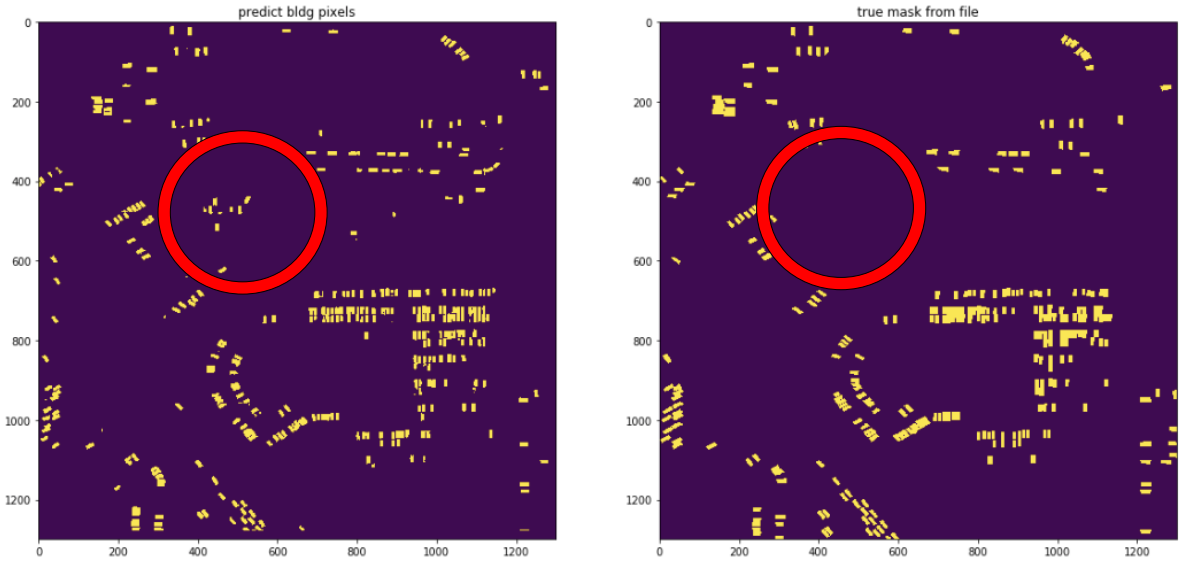


Figure 29: Predicted image (left), training set (right), screenshot from our Jypyter Notebook

Finding number 4: Epochs. We have trained multiple U-nets with the similar configuration using different number of epochs to establish what is the optimal amount between the required number of epochs and accuracy/loss function:

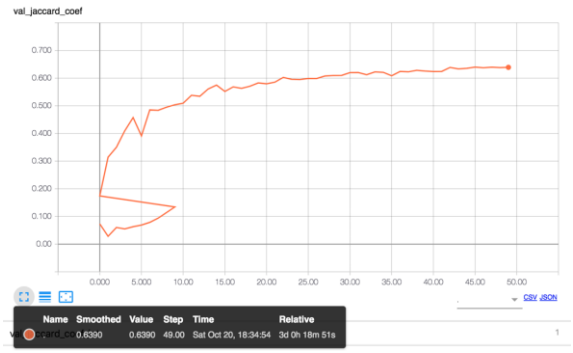


Figure 29: Epochs vs accuracy curve of a winning U-net architecture training process (TensorBoard)

We can see that beyond 25 epochs for this network, the accuracy rate curve starts plateau and further training has only incremental improvement to accuracy rate.

Also, less efficient network was trained for 150 epochs. As we can see from the figure below it has never reached the accuracy of our winning network configuration, however, it keeps on upgrading accuracy rate up until 60th epoch, beyond that point model starts to overfit. To conclude, 1) efficient networks are able to train faster and 2) an optimal number of two-way backpropagations is to be conducted between 25-50 times for network to be trained if considering computational restrains.

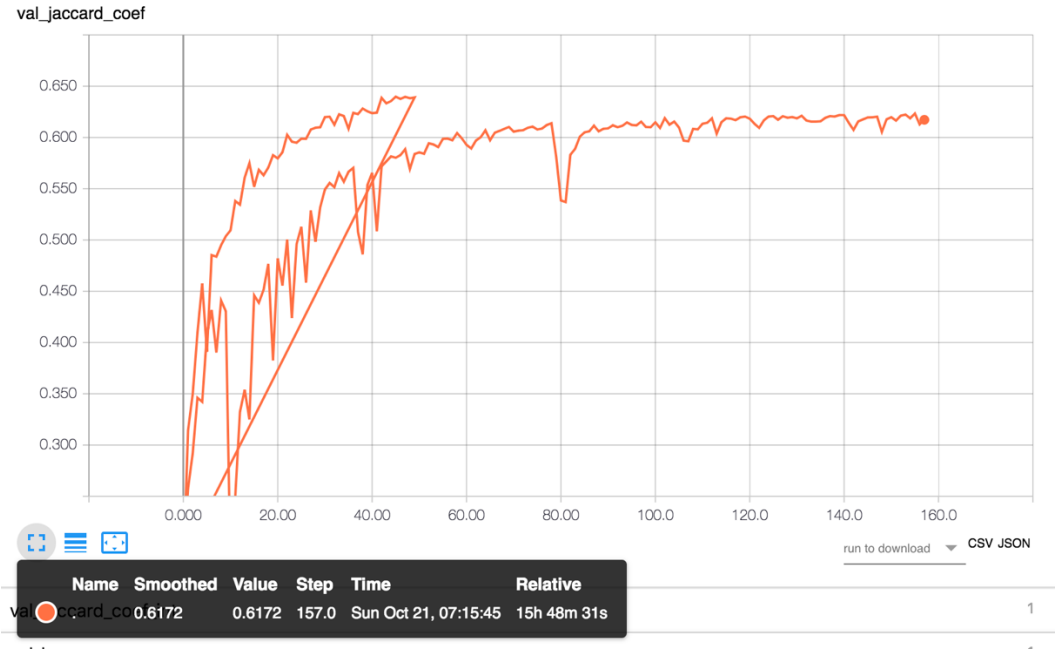


Figure: Experiment results visualization via TensorBoard, (top curve – our winning U-net architecture, bottom curve an alternative less effective architecture (reaching a max of 61.72% Jaccard coefficient after 150 epochs).

Finding number 5: Batch_size. Significant increase in the overload of the batch size (size of training batch) gives only an incremental improvement in the accuracy rate. As per our experiments 12 and 13 illustrated below figure it accrued 47.99% for 256 images per training batch vs 51.49% for 512 per training batch:

U-net Experiment 11	20	25,000 objects	0.25	64	80x80 pixel	0.02	Leaky_ReLU	100	Leaky_config
U-net Experiment 12	20	25,000 objects	0.25	256	80x80 pixel	0.02	ReLU	100	47.99%
U-net Experiment 13	50	25,000 objects	0.25	512	80x80 pixel	0.02	ReLU	100	51.49%

Figure: Table of training batch sizes experiments

Finding number 7: Pixel_frame. The size of the initial cropped mosaic of the satellite imagery matters. Conceptually speaking the larger the frame (in pixels) the more contextual information the feature map can absorb for a single object (or a part of it). It comes at a heavy computational cost and the maximum pixel matrix we were able to run on our GPU with 30GB of RAM was 160x160:

	Epochs	Training	Validation	Batch Size	"ISZ" mozaic size	Part_proc (proc c	Activation function	Pixel threshold	Jaccard Accuracy
U-net Experiment 17	50	25,000 objects	0.2(5)	128	160x160 pixel	0.02	ReLU	100	63.90%
U-net Experiment 2	20	25,000 objects	0.25	64	320x320 pixel	0.015	ReLU	100	MEMORY ERROR
U-net Experiment 3	20	25,000 objects	0.25	32	320x320 pixel	0.015	ReLU	200	MEMORY ERROR
U-net Experiment 4	20	25,000 objects	0.25	32	270x270 pixel	0.015	ReLU	200	MEMORY ERROR
U-net Experiment 5	20	25,000 objects	0.25	32	220x220 pixel	0.015	ReLU	200	MEMORY ERROR
U-net Experiment 6	20	25,000 objects	0.25	32	190x190 pixel	0.015	ReLU	200	MEMORY ERROR

Figure: Table of training pixel_frame experiments

160x160 configuration was also a part of the winning U-net architecture. This also suggests that for further experimentation, there might be a potential to further improve the accuracy of the model by expanding the pixel_frame ("ISZ"mosaic size) since it provides the freature maps with more of the contextual information that is valuable especially at the first derivative features. For this we would have to upgrade to a more powerful GCP computational infrastructure.

Finding number 6: Speed. The positive aspect of relatively small pixel_frame however, is speed. As per experiment 12 illustrated below (batch size: 256, pixel frame: 80) it took only 18 epochs to reach 49.63% Jaccard coefficient (as well as took at least 4 times less time (apprx. 35s) to backpropagate for each epoch compared to a model with 160x160 pixel frame:

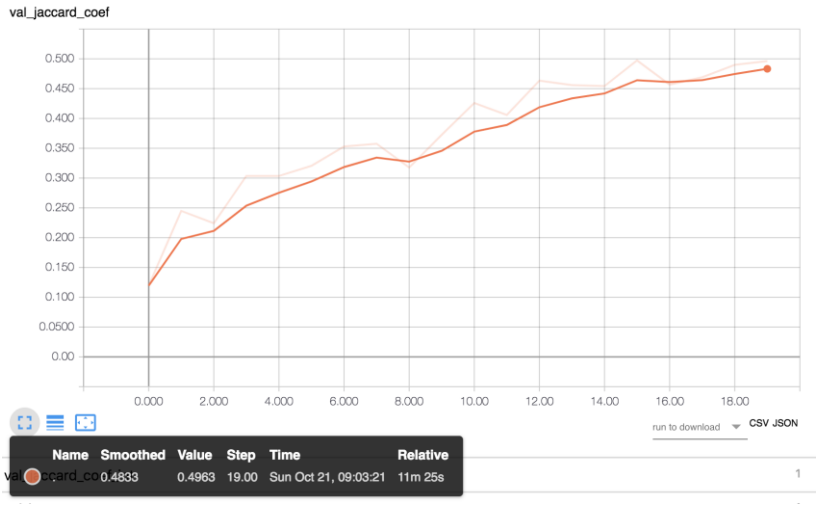


Figure: Experiment 12, accuracy improvement during training

This might be a useful piece of finding when examining the optimal U-net architecture between computational cost, speed and accuracy. Particularly relevant metrics when looking at its applications to the financial markets.

Final conclusion:

The most efficient network for our research problem is the NET. It provides the accuracy and effectively reaches close to human 94% accuracy and sometimes exceeds it.

Bibliography

[B1] LeCun, Y. (1989). Generalization and network design strategies. Technical Report, CRG-TR-89-4, University of Toronto. 326, 345

[B2] LeCun, Y., Jackel, L. D., Boser, B., Denker, J. S., Graf, H. P., Guyon, I., Henderson, D., Howard, R. E., and Hubbard, W. (1989). Handwritten digit recognition: Applications of neural network chips and automatic learning. IEEE Communications Magazine, 27(11), 41–46. 362

[B3] Ian Goodfellow, Yoshua Bengio, and Aaron Courville: Deep learning: The MIT Press, 2016, 800 pp, ISBN: 0262035618, Convolutional Networks, pp.330

[B4] - S. Lipschutz; M. Lipson (2009). *Linear Algebra (Schaum's Outlines)* (4th ed.). McGraw Hill. ISBN 978-0-07-154352-1.

[B5] - A Krizhevsky, I Sutskever, GE Hinton (2012), "Imagenet classification with deep convolutional neural networks". Advances in neural information processing systems, 1097-1105

[B6] - K Simonyan, A Zisserman (2014), "Very Deep Convolutional Networks for Large-Scale Image Recognition," arXiv preprint arXiv: 1409.1556

[B7] - C Szegedy, W Liu, Y Jia, P Sermanet, S Reed (2014), "Going deeper with convolutions"

[B8] - Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun (2015), "Deep Residual Learning for Image Recognition"

[B9] – O Russakovsky, D Hao S Jonathan K/ Sanjee, S. Sean, M. Zhiheng, H. Andrej, A. Khosla, M.Bernstein, C. BergLi Fei-Fei, "ImageNet Large scale visual recognition challenge"

[B10] - Alfredo Canziani, Adam Paszke, Eugenio Culurciello (2017) An Analysis of Deep Neural Network Models for Practical Application,

[B11] - Long, Jonathan, Evan Shelhamer, and Trevor Darrell. (2015) ”Fully convolutional networks for semantic segmen- tation.” Proceedings of the IEEE Conference on Com- puter Vision and Pattern Recognition

[B12] - Ronneberger, Olaf, Philipp Fischer and Thomas Brox (2015) ”U-net: Convolutional networks for biomedical image segmentation.” International Conference on Medical Image Computing and Computer-Assisted Intervention. Springer International Publishing

[B13] - Sabour, Sara, Frosst, Nicholas, and Hinton, Geoffrey E. Dynamic routing between capsules. In Advances in Neural Information Processing Systems, pp. 3857–3867, 2017.

[B14] - Dilin Wang, Qiang Liu (2017), “AN OPTIMIZATION VIEW ON DYNAMIC ROUTING BE- TWEEN CAPSULES” Department of Computer Science, University of Texas at Austin

[B15] https://www.kaggle.com/c/dstl-satellite-imagery-feature-detection

[B16] https://registry.opendata.aws/spacenet/

[B17] - Jaccard, P. 1912. The distribution of the flora in the alpine zone. New Phytologist 11 37–50.

[B18] Längkvist, Martin, et al. "Classification and segmentation of satellite orthoimagery using convolutional neural networks." Remote Sensing 8.4 (2016): 329

[B19] Castelluccio, M.; Poggi, G.; Sansone, C.; Verdoliva, L. Land Use Classification in Remote Sensing Images by Convolutional Neural Networks; (2016)

[B20] Castelluccio, Marco, et al. "Land use classification in remote sensing images by convolutional neural networks." (2015).

[B21] Chatfield, K. Simonyan, A. Vedaldi, and A. Zisserman, “Return of the devil in the details: Delving deep into convolutional nets,” in British Machine Vision Conference, 2014

[B22] Wang, Haipeng, et al. "Application of deep-learning algorithms to MSTAR data." Geoscience and Remote Sensing Symposium (IGARSS), 2015 IEEE International. IEEE, 2015.

[B23] Abdullah, Qassim, et al. "New Standard for New Era: Overview of the 2015 ASPRS Positional Accuracy Standards for Digital Geospatial Data." *Photogrammetric Engineering & Remote Sensing* 81.3 (2015): 173-176.

[B24] Längkvist, Martin, et al. "Classification and segmentation of satellite orthoimagery using convolutional neural networks." Remote Sensing 8.4 (2016): 329.