TEL AVIV UNIVERSITY
School of Electrical Engineering

Bats Orientation Research Using Deep Neural Network

A project submitted toward the degree of
Master of Science in Electrical and Electronic Engineering

By

Shimon Akrish

This research was carried out in The School of Electrical Engineering
Under the supervision of Prof. Yossi Yovel and Dr. Raja Giryes

August 2019

Contents

1 Table of Figures and Tables

2 Abstract

This project is done by the collaboration between the zoology and electronic faculties.
With the guidance of Prof. Yossi Yovel in zoology I have researched how bats navigate and orient while traveling for long distances. It is known that bats have limited vision capability,
And From that arises the question, how do they manage to navigate form their cave to the same fruit tree across 17 km in a straight line for several nights. To answer this question assuming they navigate based on their vision I have trained a neural network model with footage taken by a drone along their travel route, using this trained network I have simulated their navigation and extracted information from the network that may help understand based on what visual objects those bats rely while they navigate,

3 Introduction

Artificial neural networks are commonly used today in multiple domains such as computer vision, voice recognition, sequence recognition, and have various applications in robotics, self-driving vehicles, face recognition and many more. Some of the neural networks (NN) are built from clustering and classification layers designed to recognize patterns. Supervised learning networks are first trained with labeled data, during the training the values of the weights are determined.

Bats researcher Prof. Yossi Yovel from the zoology faculty suggests a resemblance between bats vision and the trained convolutional neural network (CNN) after its supervised learning.
In this project we look at a different angle on the neural network. By extracting information from the neural network, we try to understand how the bat interoperate visual objects in his long navigation from his home cave 17 km west to his feeding fruit tree in an approximately straight line.

This project was a long research of trial and error until we reached the right methods.
In the first sections of this document I will explain and give details regarding my final methods that I have used, and in the appendix I will give brief overview of the methods that were neglected along the way.

The first challenge was to collect the database that we will be used to train the CNN.
The database for the CNN training was videos we captured with drones across the bats path from the cave to the tree, those videos were taken at night and in the elevation of the bat's flight.

The second challenge was to find an architecture that will be able to correctly classify those images And to choose the right method of labeling the database. The CNN model that was finally chosen is a custom VGG 19 architecture.

After reaching a reasonable accuracy the third challenge was to use the CNN and create a navigation simulations that can successfully navigate based on input images from the bat's path.

The final challenge was extracting information that will help understand better on what visual object the CNN rely while classifying, to do that I have concealed systematically parts of the input images and examined the increase in the classification error.

* Project repository hosted on Github at https://github.com/shimonakrish/bats_nn_research

4 Database

The first step was to collect visual data of the bat's flight path, the home cave of the bat is located in "Beit Guvrin" National Park and the destination fruit tree is located near "Nir Banim" (west point), the distance between those two locations is 17.2 km as shown in figure 2

We tried to create our database as close as possible to what the bat actually sees. By using drones we were able to film the bat's flight path in the same conditions, we filmed only in night and in the elevation of the bat's flight (100m).

We used DJI Mavic Pro and DJI Phantom 4 drones for that task.
Those Drones flight distance is limited by their signal reception distance which is approximately 1 km. We started filming from the cave towards the fruit tree, advancing by car along the path 1 km at a time. Another limitation was the battery which is limited to 20 minutes of flight. Not all parts of the path is accessible by roads, due to that limitation we were able to film semi-continually only the last 11km of the path as you can see in figure 3. The drone camera field of view is 79 degrees, to train the CNN properly the database should include all 360 degrees images from along the whole path. To overcome that limitation every 250 meters we stopped the drone in midair and did a 360 turn to capture the whole surrounding of that location. Due to this nature of filming most of the training database contains images with azimuth directed to the tree.

In order to supervise train the CNN it needs a database of labeled images. For the labels we needed the azimuth of the image. I have extracted the azimuth from the drone log files. The drone samples his sensors including his magnometer 10 times per second. But the frame rate of the video recording was at 30 frames per second (fps), to overcome this difference I used video editing tool and reduced the videos frame rate to 10 fps, after that I have done manual adjustments to fully sync the labels and the video recordings

The drone video recordings was then split into frames, and each frame now have an azimuth extracted from the drone's log. But in order to train the CNN to be able to navigate I couldn't just use the azimuth as a label. Based on the fruit tree destination GPS location, the current GPS location of the image and its azimuth I have calculated the angle of correction from the current image azimuth to the destination in the range of [-180, 180]. That correction in degrees served as the images labels.

Figure 1 demonstrate the correction angel.

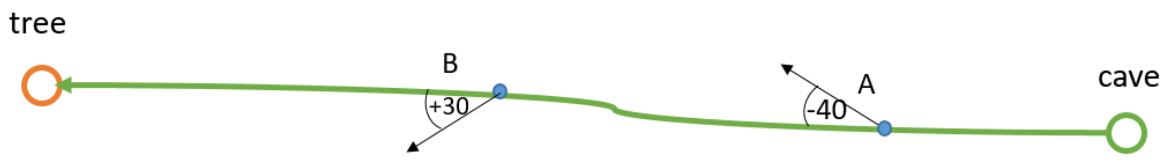

Figure 1 correction angel

The green line is the bat's path, in point A the image azimuth (heading) is 40 degrees above the destination tree so the correction is -40 degrees, in point B the image azimuth is 30 degrees below the destination target thus the correction angle is +30 degrees
The formula of that calculation is displayed in equations 2 and 3 in section 4

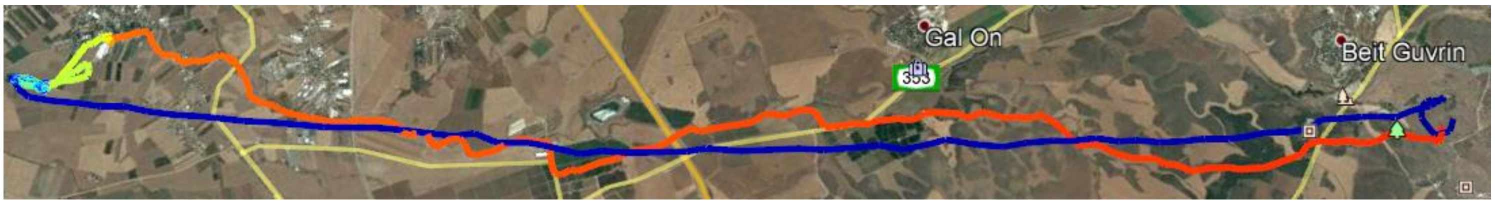

Figure 2 Bat's path from the cave in the right to the fruit tree in the left
this is one day recording of the bat go and back trip

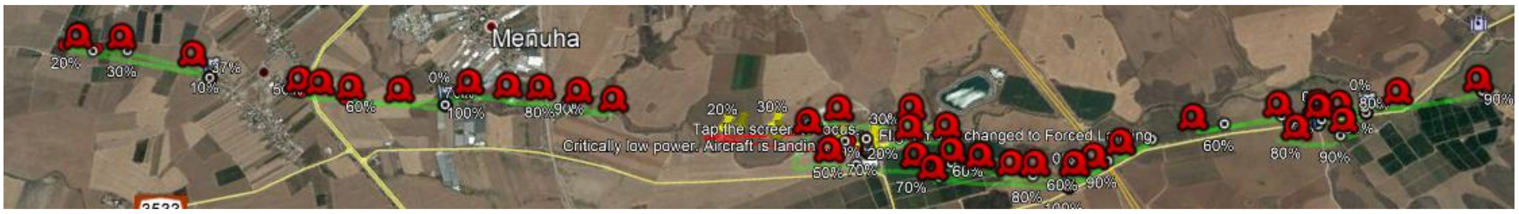
Menuha
Critically low power. Aircraft is landing
changed to Forced Landing
3533

Figure 3 Recorded database
the green lines symbols the train data,
the red objects are the locations with 360° recordings
and the red line is the test data

To test better the trained CNN We took 360 degrees recording from points along the path that have distance of approximately 2 and 5 km from the path, as can be seen in figure 4.

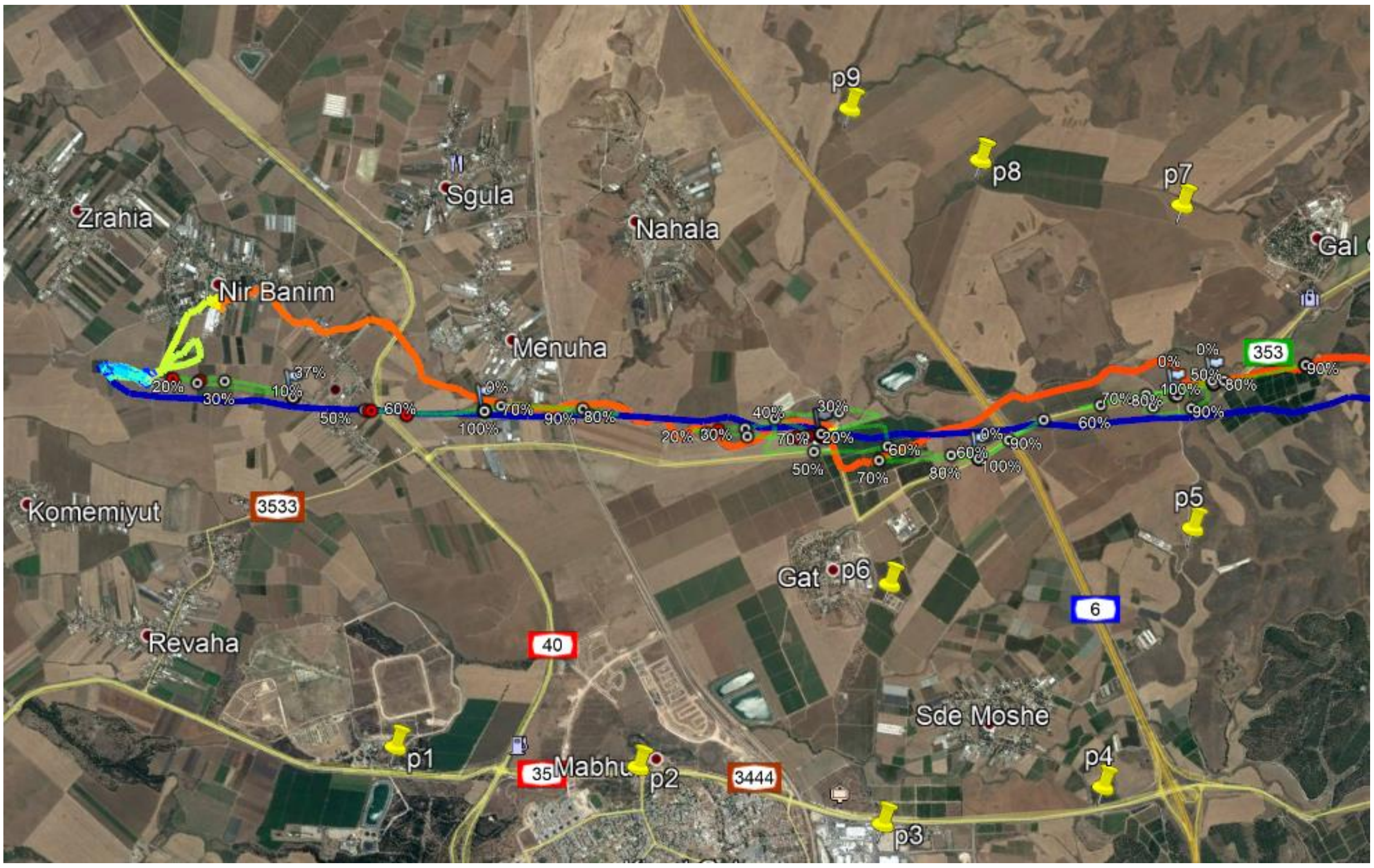

Figure 4 Distanced test data
the nine yellow markers are the locations of the 360° test data recordings

The Bats we study have two light-sensitive proteins at the back of their eyes, enabling them to see 2 colors, in order to simulate their vision I have zeroed the Green color channel from the Database images. Splitting all video recording to Database images resulted with ~33K images

5 The Neural Network

The chosen architecture was a custom VGG 19 [3] in which I replaced the Softmax classifier with a regression classifier, the reason I did this is because the CNN output should be a float number in the range [-180, 180] that represent the correction angle as described in chapter 3.

The programming platform in which I developed this model was TesnsorFlow in Python language.

Due to our small amount of training data (~30K images) I used a technique called "transfer learning" as a starting point for the model training, in this approach the first convolutional layers were pre-trained on a big image database (imageNet), this way I was able to leverage the pre-trained model weighted layers to extract features and by only training the later layers, the model was tuned for the azimuth regression task, Without using this technique the CNN could not converge.

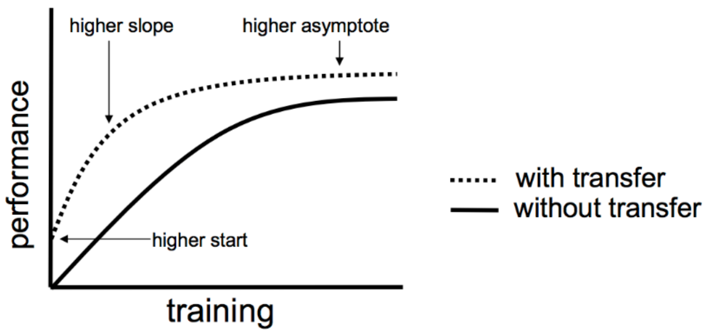

Figure 5 transfer learning [1]

The test set is one of the recordings taken by the drone marked in red in figure 3 the total number of test images is about 3K

I have subsampled the input images resolution to 160x160 using bi-cubic interpolation. This subsamples blurs the images similarly to the bat's blur vision.

The selected batch size was 32, this was the largest batch size I could reach using my Hardware (NVidia GTX970 4GB Ram)

The layers that were not pre trained was initialized with orthogonal initializer [2]. Meaning we initialize each layer (matrix) with eigenvalues of an orthogonal matrix that have absolute value 1. This helps the CNN on preforming repeated matrix multiplication without causing the weights to explode or vanish.

Following VGG paper [3] I the commonly used Relu activations,

One of the problems in training a NN model is that there is more than one solution to the weight optimization problem thus the learning can result with very large weights, to avoid large weights an L2 regularization penalty was added to the loss function. Let $W_{k,l}$ denote the kth weight in layer l weights matrix W_l, the regularization penalty is calculated as elementwise quadratic penalty over all the model weights

$$\boldsymbol{R(W) = \sum_k \sum_l W_{k,l}^2} \qquad \textbf{(1)}$$

Another issue in NN is overfitting VGG19 original model uses Dropout (figure [6]) after every MaxPool
I have tried using a later released method called batch normalization but the result didn’t show improvement so I stayed with the original method which preforms less calculations
I used 0.8 keep ratio during training

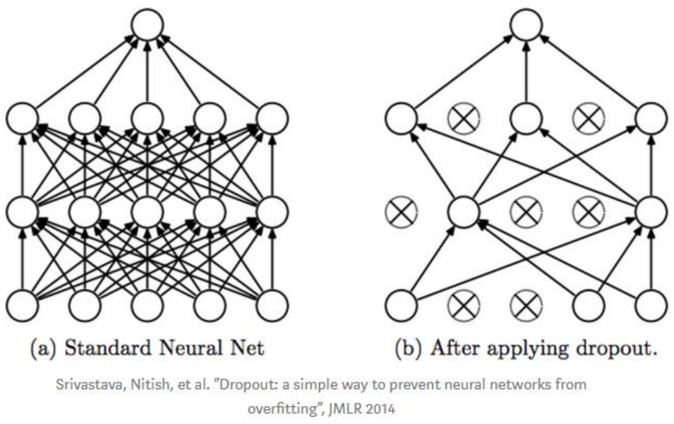
(a) Standard Neural Net
(b) After applying dropout.
Srivastava, Nitish, et al. "Dropout: a simple way to prevent neural networks from overfitting", JMLR 2014

Figure 6 Dropout [4].

For each hidden layer, for each training sample, for each iteration, ignore (zero out) a random fraction, *p*, of nodes (and corresponding activations).

The loss function was modified to reflect the difference between the label and the output of the CNN, the calculation is as follows

$$\boldsymbol{angel1 \triangleq label, \quad angel2 \triangleq CNN\ outpu} \tag{2}$$

$$\boldsymbol{ang\ diff = \min\{|angel1 - angel2|, 360 - |angel1 - angel2|\}} \tag{3}$$

$$\mathbf{loss} = \sum_{\mathbf{image\ 0}}^{\mathbf{image\ 31}} \left\{ \frac{(\mathbf{and\ diff})^{2}}{\mathbf{2*32}} + \mathbf{1e^{-4}} \cdot (\mathbf{regulariztion\ losses}) \right\} \tag{4}$$

eq. (3) calculates the minimum angel difference between the label and the CNN output. The angel difference between two angels can have two results for example:
Angel A = 50°, Angel B = 60°, the clockwise difference from A to B is 10°. But the counter clockwise difference is 350°, because the CNN output could be lower or larger than the label I wish to always get the smallest difference between them.

The CNN's Loss function is a Mean Squared Error (MSE) as described in eq. (4), and because of that the problem is convex which enables us to optimize the model with gradient decent.
The mini-batch size is 32 therefore I sum over 32 images in eq. (4).
The learning rate hyper-parameter controls the size of the steps at which the model learns, setting it too high and the model will not converge, setting it too small and it will converge in a very long time. A simple approach is to start with medium steps and as we come close to the minimum reduce the step size so we don't miss it as can be seen in figure 7.

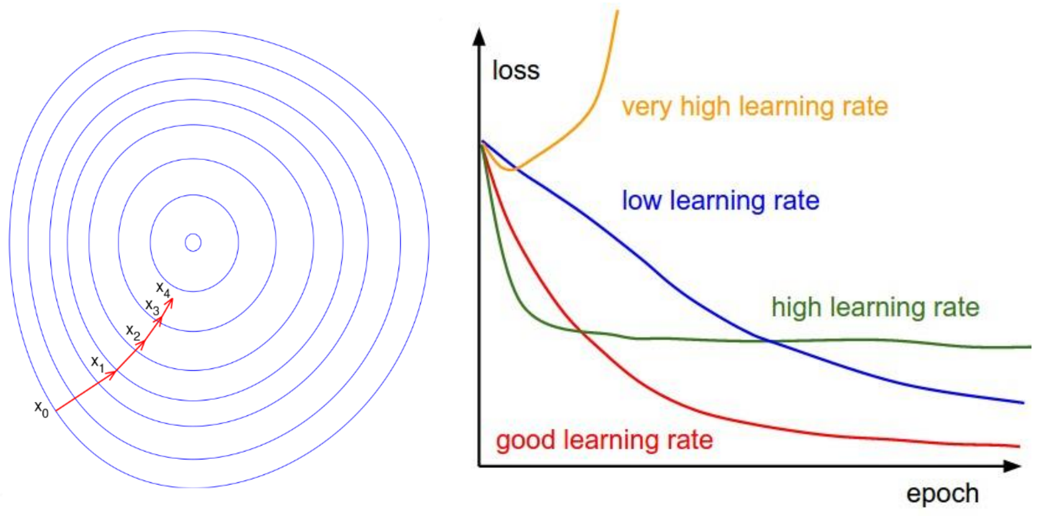

Figure 7 learning rate [5] [6]

Assuming a vector of parameters *x* and the gradient *dx*, the simplest update has the form:

Vanilla update

$$x = x - (learning\ rate) \cdot dx \qquad (5)$$

The previous approach manipulate the learning rate globally, another approach is an adaptive method called Adagrad Optimizer originally proposed by Duchi et al [7]

$$\mathbf{cache} = \mathbf{cache} + (\mathbf{dx})^2 \qquad (6)$$

$$\mathbf{x} = \mathbf{x} - \frac{(\mathbf{learning\ rate})\cdot \mathbf{dx}}{\sqrt{\mathbf{cache}+\boldsymbol{\varepsilon}}} \qquad (7)$$

> This is well explained in [6]:
> "Notice that the variable cache has size equal to the size of the gradient, and keeps track of per-parameter sum of squared gradients. This is then used to normalize the parameter update step, element-wise. Notice that the weights that receive high gradients will have their effective learning rate reduced, while weights that receive small or infrequent updates will have their effective learning rate increased. Amusingly, the square root operation turns out to be very important and without it the algorithm performs much worse. The smoothing term eps (usually set somewhere in range from 1e-4 to 1e-8) avoids division by zero. A downside of Adagrad is that in case of Deep Learning, the monotonic learning rate usually proves too aggressive and stops learning too early"

To overcome the Adagrad downside of monotonic learning rate I have combined the exponential learning rate decay with Adagrad preventing from the model to stop learning too early.
The Initial learning rate is 0.01

Note: Before I chose the VGG19 as the base model I have experimented other architectures

At the beginning I used VGG16 with Softmax classification with 360 classes, the result were not very good, later I moved to a regression model. I cannot compare the accuracy between those two because since I have moved to VGG19 I replaced the Database and modified the whole code. Another attempt was made with Convolutional Auto encoder.

Further explanation regarding my previous attempts will be discussed in the appendix

Table 1 gives a summary of the custom VGG19 model

Layers	**Pre Trained**
Conv 3-64 Conv 3-64	Yes
MaxPool 1 + Dropout	
Conv 3-128 Conv 3-128	Yes
MaxPool 2 + Dropout	
Conv 3-256 Conv 3-256 Conv 3-256 Conv 3-256	Yes
MaxPool 3 + Dropout	
Conv 3-512 Conv 3-512 Conv 3-512 Conv 3-512	Yes
MaxPool 4 + Dropout	
Conv 3-512 Conv 3-512 Conv 3-512 Conv 3-512	Yes
Conv 3-512 Conv 3-512	No
MaxPool 5 + Dropout	
FC-4096	No
Dropout	
FC-1024	No
Dropout	
FC-256	No
Dropout	
FC-1	No

Table 1 Selected CNN Architecture

6 Results

The criteria of success used with the test batch (marked in red in figure 3) is

$$|\boldsymbol{output\ angel} - \boldsymbol{label\ angel}| < \mathbf{25}^{\circ} \tag{8}$$

The meaning of this criteria is that the bat at each point along the path can have a 25 degrees of error either above or under the target heading this gives about 14% maximum error from the true heading

Zooming in on the original recorded bat's flight from Figure 2 it can be seen that the general direction of the flight is kept with heading correction every few meters (Figure 8)

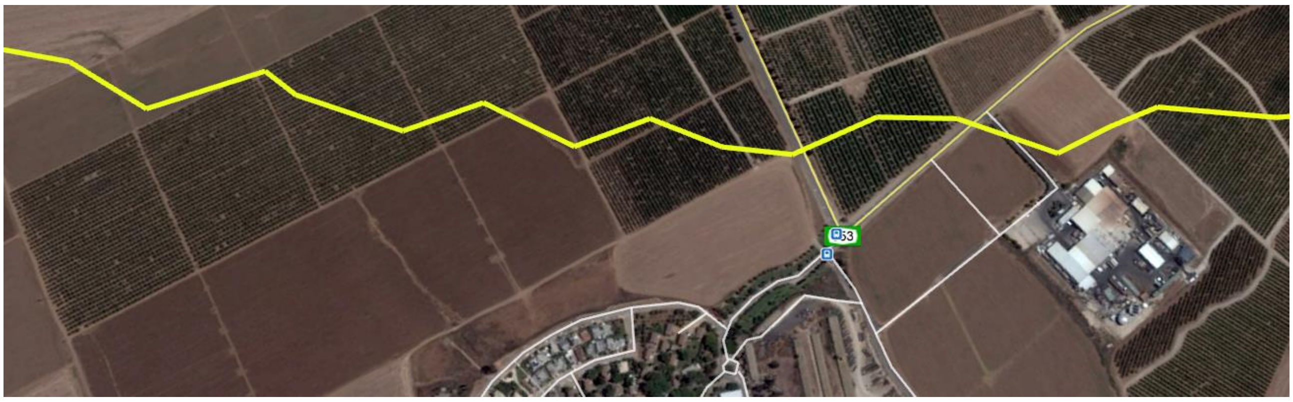

Figure 8 Bat's path Zoom in

6.1 Results on the training test batch

- Training with all three RGB channels the accuracy rate on the test batch is 0.68
  (meaning that in 68% of the test data the model passed the criteria of success that in eq. (8))
- Training only on the R and B channels the average accuracy dropped to 0.622

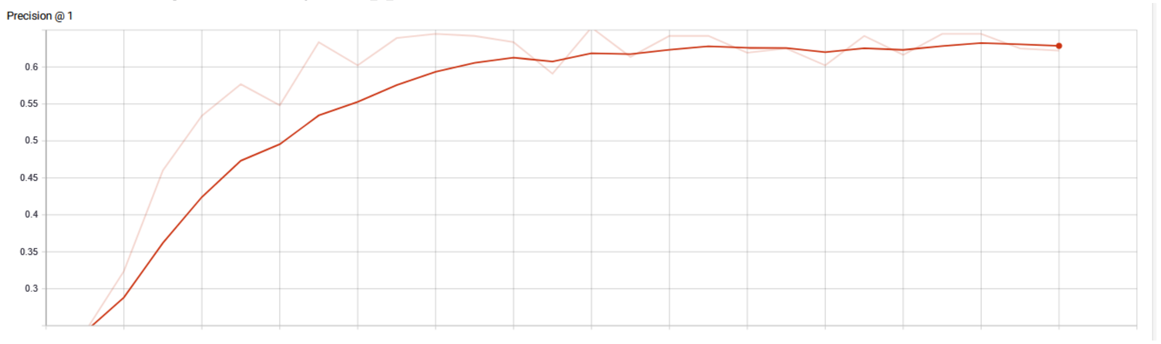

Figure 9 CNN test accuracy

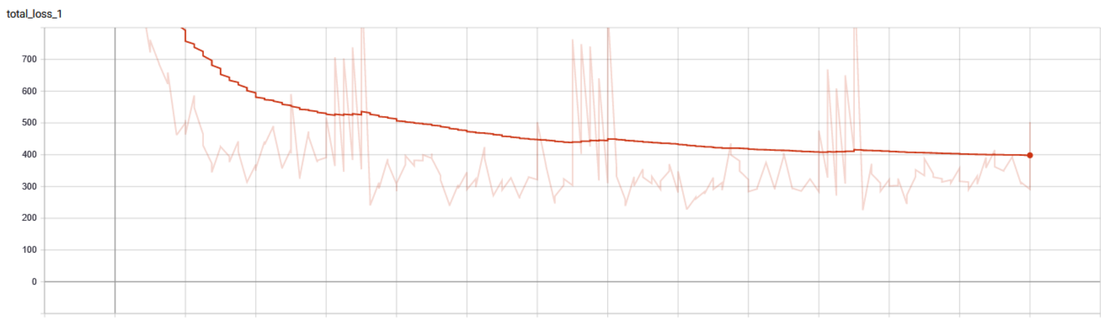

Figure 10 CNN loss

6.2 Test results on the 9 distanced test points

The motivation for this testing is to verify that the CNN can classify untrained images far away from the original path with grater variance than the untrained images along the path. As can be seen in figure 4 the area is populated with bright roads and settlements that causes the input images to be very different from the train data.
I have created a table that specify the angels in which the CNN passed the regression criteria

Point	*Angle range*
1	200 - 220, 231 - 284, 304 - 311, 330 - 340.
2	262 - 300, 306 - 316.
3	no continues correct regression
4	199 - 224.
5	174 - 183, 190 - 257.
6	158 - 261, 273 - 316.
7	182 - 260.
8	179 - 190, 197 - 247.
9	169 - 278.

Table 2 Distanced test point results

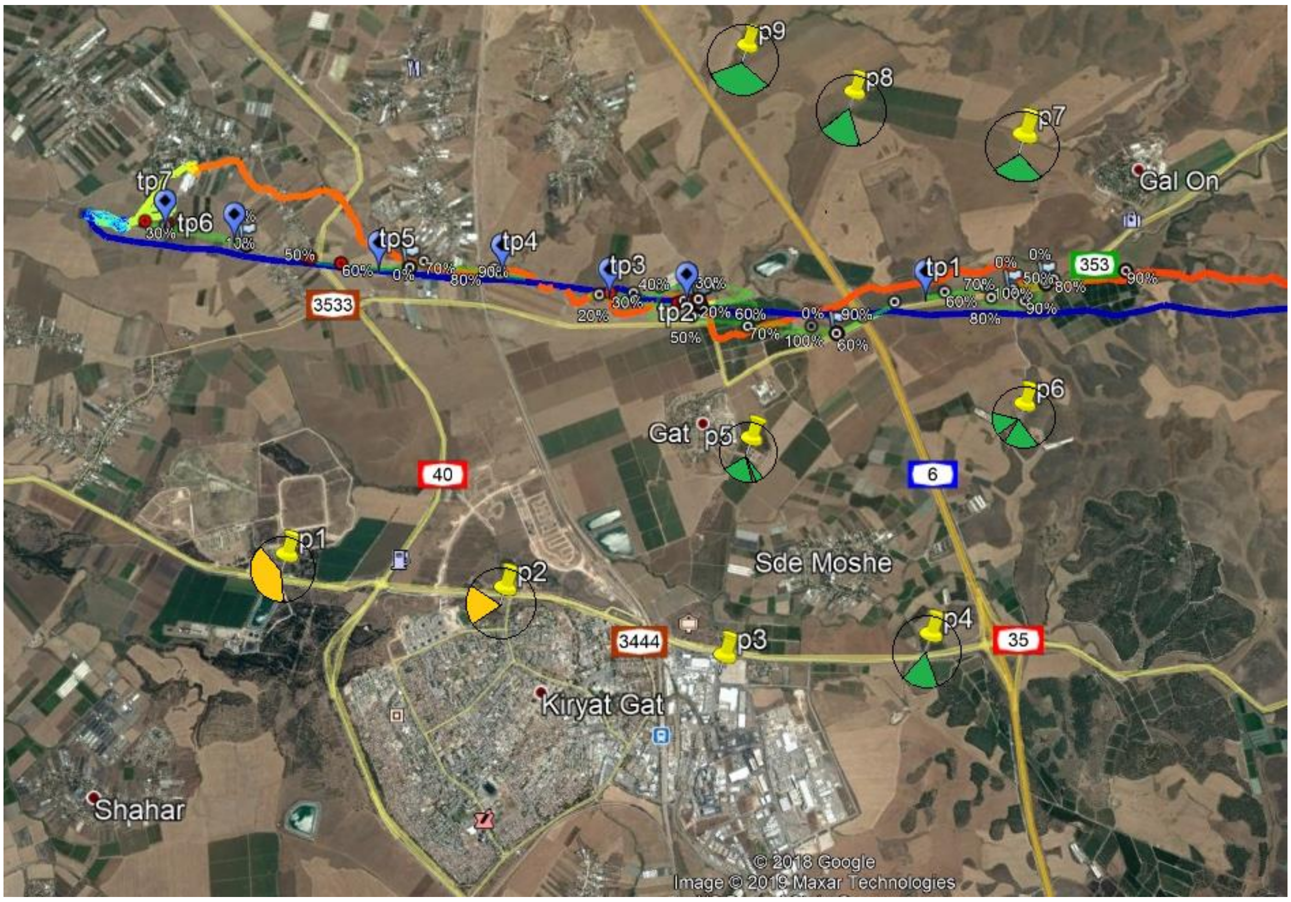

Figure 11 Distanced test points results
The color orange symbols semi continues correct regression range (see table 2),
And green symbols fully continues correct regression range (see table 2).
(The Blue drops are test points that will be explained in section 6.3)

Figure 11 shows the angles from table 2 on the map
It can be seen that points p4 till p9 can classify well the images with azimuth directed to the settlement Kiryat Gat which has bright lightings as can be seen in figure 12
But CNN output azimuth from images too close to kiryat gat like in test points p1, p2 and p3 did not pass well the regression criteria, this is caused because the bright settlement from a close view is very different from the train data displayed in figure 13

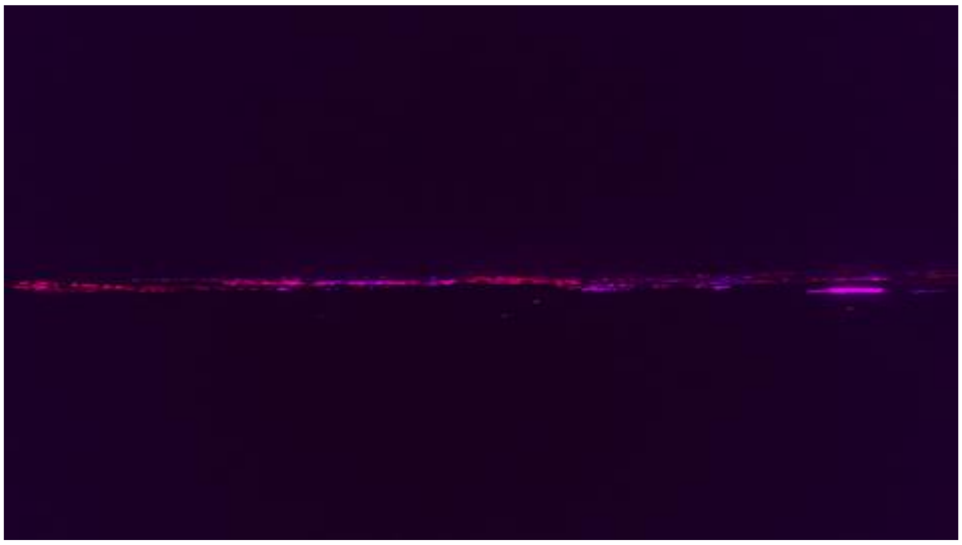
Figure 12 Input image from test point p7, azimuth 256

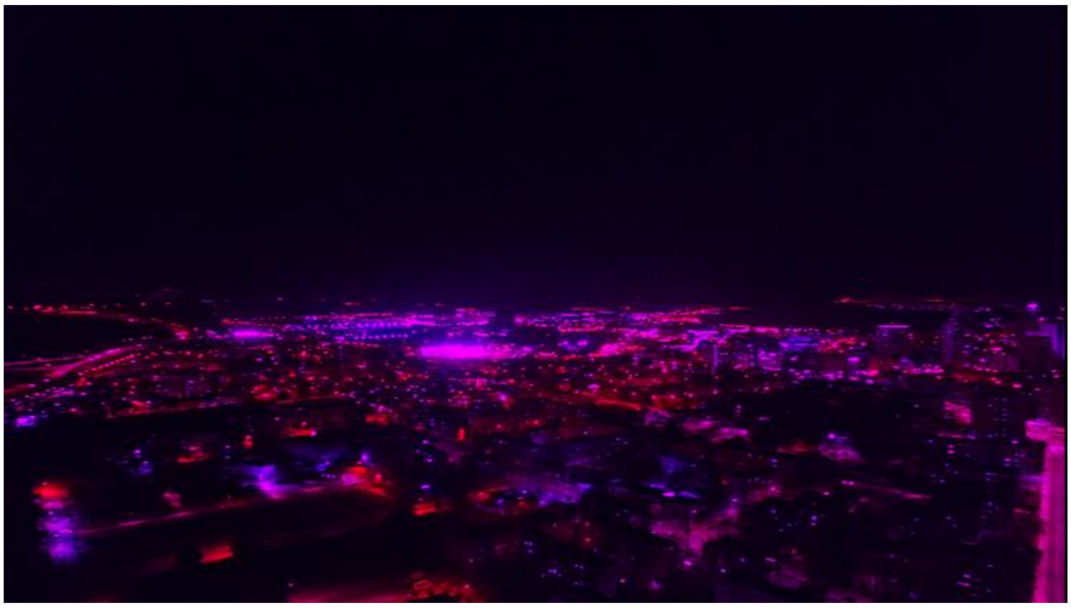
Figure 13 Input image from test point p2

An observation that can be made based on the above results is that the bat uses Kiryat Gat settlement as a reference during his navigation.

Disclaimer that should be taken in account is that because the 9 test point were recorded far away from the train data, many bright roads and settlements that that comes between the test point and the fruit tree made those test points images to be quite different from train data, but in Kiryat Gat azimuth there is less distractions and they can be neglected compared to the brightness and size of Kiryat Gat therefore we had better results in that azimuth.

Note that the successful navigation simulation in section 7 don't rely on Kiryat Gat when using as input the images from the original bats path.

6.3 Error vs Azimuth

7 test points along the path were selected displayed in figure 14. In each test point images were selected for every Azimuth with one degree resolution meaning 360 images for every test point.
The selected images are from the vicinity of the test point because the video recording can't grantee to have for every point in the path 360 degrees images, the selected images for each test point are the closest images to the test point that have the selected azimuth for test.
The images may be selected either from the test data or train data as well

Two tests were made, the first test uses the CNN that was trained on all the training data available
The training data is represented as the green lines in figure 14
The second test uses CNN that was trained only on part of the train data and tested only on points tp4 and tp5 as can be seen as green lines in figure 15.

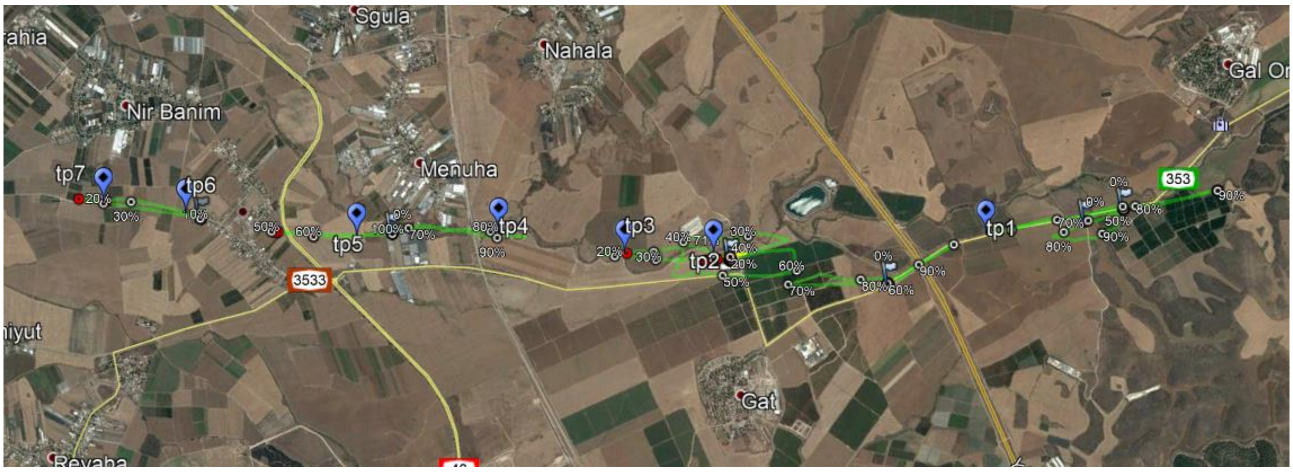

Figure 14 Full training database

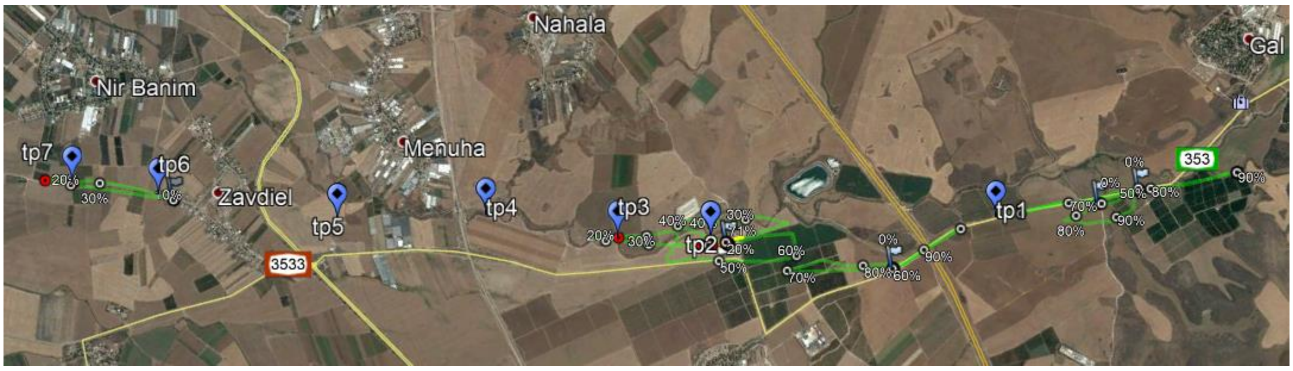

Figure 15 Partial training database

Figures 16 and 17 displays the moving average on the CNN output error in degrees vs the azimuth on the blue test points.

The blue graph represents the CNN trained with all training data except the test batch
And the graph is an average of the CNN output from all blue test points (tp1 -> tp7)

The red graph represents the CNN trained with part of training data
And the graph is an average of the CNN output from test points tp4 and tp5 only

Both graphs are aligned so that azimuth 0 is in the direction of the fruit tree.

As expected the blue graph that represent the CNN that was trained on all the training data has lower error than the Red graph in most degrees

Because that the training data was recorded with drone flying from the bat's cave towards the fruit tree most of the train images are with azimuth headed to the fruit tree and much less images that facing the opposite direction, as expected the error has a peak in the opposite direction.
The yellow line in Figure 19 shows the azimuth with peak error

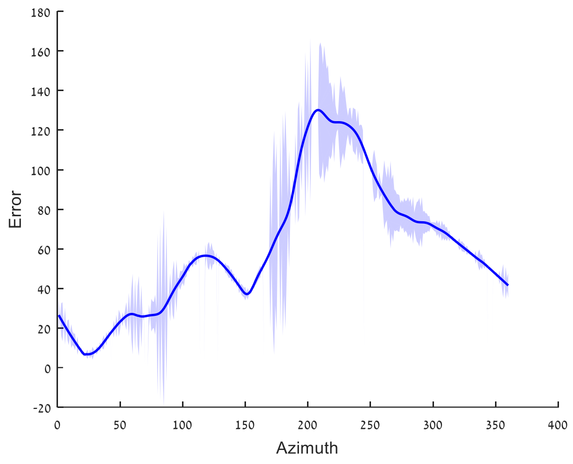

Figure 17 Error Vs input image azimuth (CNN trained on all data)

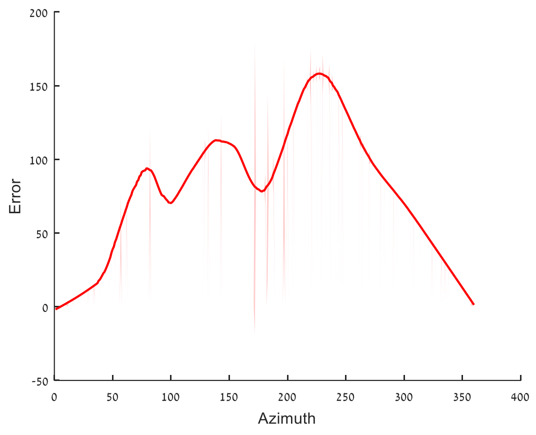

Figure 16 Error Vs input image azimuth (CNN trained on part of the data as can be seen in figure 15)

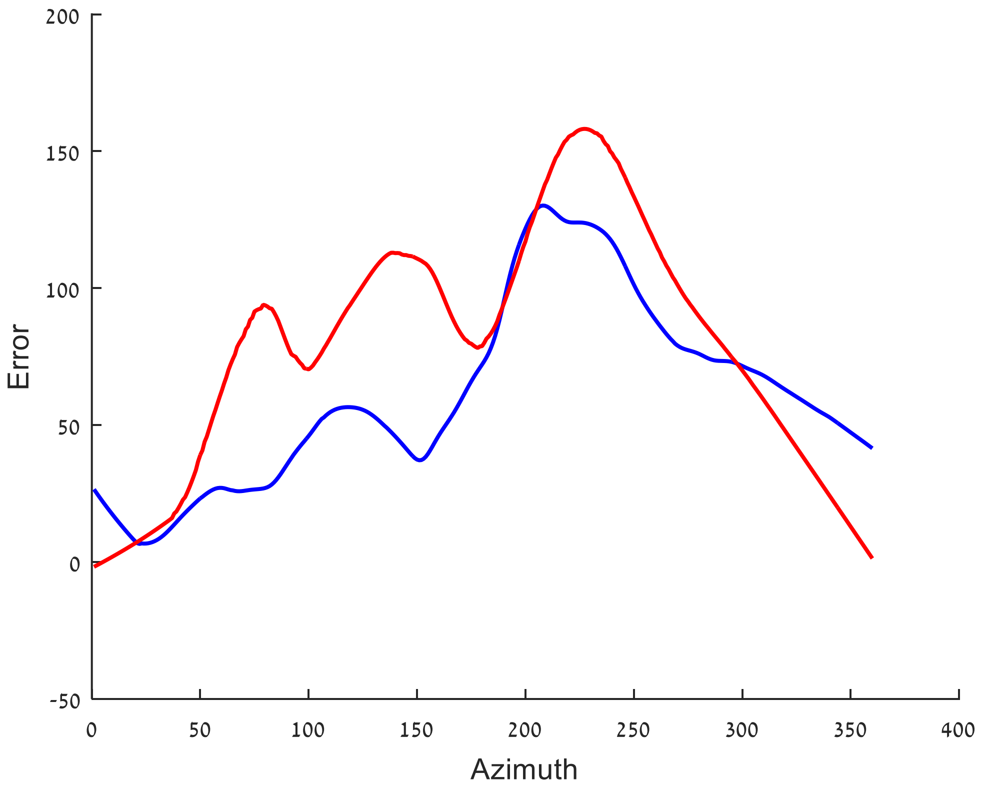

Figure 18 Regression Error Vs input image azimuth combined graph

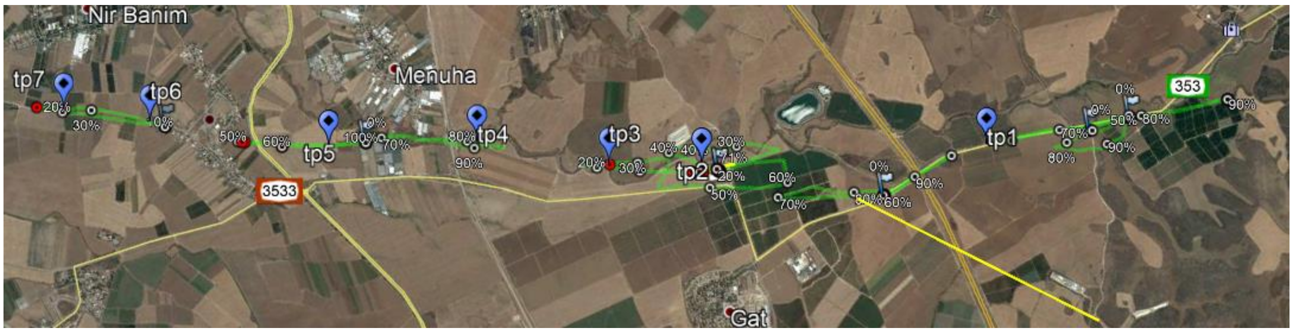

Figure 19 Azimuth with peak error

6.4 Images concealments

Another method we did trying to understand on what the CNN relay to classify,
Is by concealing parts of the input images that goes to the CNN and measuring the increase of the error in the regression

The concealed test images were taken from the 9 distanced test points that can be seen in figure 4. The chosen images form the 9 test point are only those images that the CNN could classify correctly under the regression criteria in eq. (7)

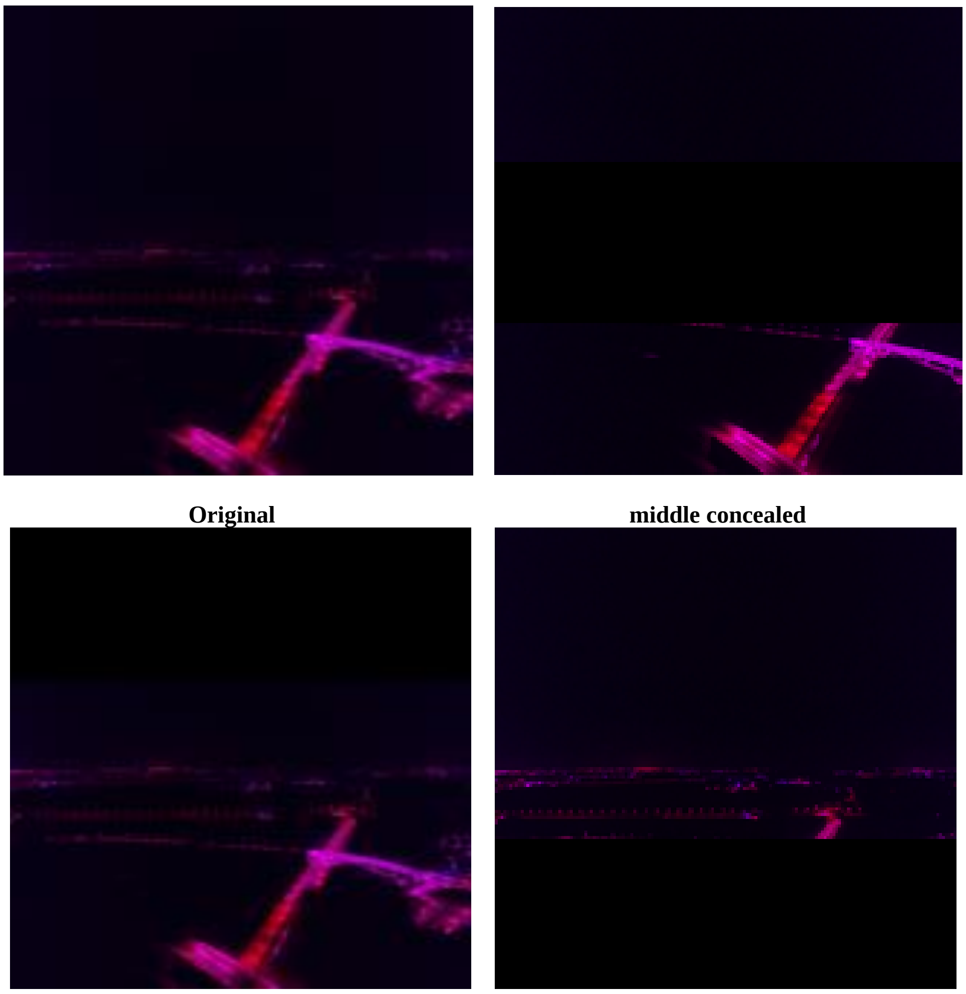

Figure 20 Input image concealment samples

Result of the average added error per concealment section can be seen in the following table

	bottom	middle	top
Mean added error	16.762	101.4749	26.47846
Standard Deviation	65.209	43.027	56.215

Table 3 Concealment test results

The conclusion from that experiment is that the horizon is the most important part in the image regression. Because most of the images have information concentrated mostly in the middle section as can be seen in figure 21
Another conclusion is that the sky information is more helpful than the bottom more rapidly changing section.

Figure 21 Typical input image

6.5 Neurons test

This test was created for further investigating if there is neurons that has high activations in a specific azimuth no matter where we test along the path.

In the bottom of the CNN we have three fully connected layers that are connected in series with output length of 4096, 1024 and 256 this layer were trained with the training database.
Each fully connected output goes through a Relu activation layer (the CNN architecture can be seen in table 1) the output of the Relu activation is the neurons that we investigate.

In the test I feed the CNN with images in each and every one of the 360 azimuth from the 7 test points (Figure 14). For each azimuth I look on the activations of the fully connected layers and averaged them upon the 7 test points,

For example the first fully connected layer has output of 4096 neurons, thus the first matrix dimensions are 360x4096 while each line is one of the 360 direction and each value in that line represents the neuron's activations average from the 7 test point's.

This matrixes will be processed and studied by Prof. Yossi Yovel.

7 Navigation simulation

To see if the CNN can actually navigate from the cave to the fruit tree based on its training data I have created a stepwise simulation that will try to orient itself along the bat's path.

The navigation done with fixed size steps (350 meters), in each step the CNN needs to determine the azimuth of the next step based on input images from the current location. Figure 22 illustrates this stepwise navigation.

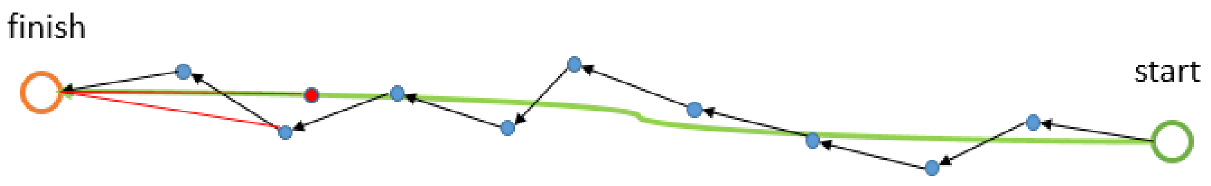

Figure 22 Navigation simulation ilustration

The database doesn't contain data from all possible headings in all the locations along the path, But because the images from two adjacent locations are similar, I search for the closest image in the step’s azimuth and use it instead. This approximation can be done only if the simulation's navigation don't deviant too far from the database path. The simulation input images are taken from test and train data together

During the bat’s flight he can turn his head around and orient itself based on his visible surrounding. The field of view (FOV) of the drone's camera is limited to 79°, to imitate the bats capabilities I averaged the results from 3 input images with the azimuth of the step direction. The 2 other input images are +/-2 degrees from the step azimuth. This averaging helps the CNN overcome the regression error.

In the fruit tree azimuth, the regression error is low as can be seen in figure 18 once the CNN manage to orient itself to the fruit tree azimuth it will continue stepping in the correct azimuth because of the relative low regression error in that azimuth

Two compensations are needed because of the above approximation method of taking adjacent images to the current location

In each step the calculated correction is added to the current step azimuth resulting with the next step azimuth. But if the best input image I could find is not exactly in the current step azimuth The CNN's regression result will also have a small angel difference.
Thus a compensation to that correction needed to be made by subtracting the CNN result with the angle difference between the input image and the current step azimuth.

The second compensation is needed if the current location is more to the north or to the south from the input test image as can be seen figure 22 near the finish point in red, in the figure you can see the current location is under the finish point while the input images comes from above it on the green path, feeding this to the CNN will result with an offset of that angel difference. This correction is more crucial as the simulation comes closer to the finish point. To compensate I add the difference in degrees between the current simulation location heading to the target and the result heading to the target.

I have created a test bench in which I ran the simulation many times with different starting conditions. Each test can start in different point along the path in the range of 1km from the beginning of the path and in [-30°, 30°] deviation from the fruit tree azimuth

I have made simulations on the two differently trained CNNs, the first CNN that was trained on all training data (as can be seen in figure 14) and the second CNN that was trained on part of the training data (as can be seen in figure 15)

Here is a sample of three simulations that was made based on all training data CNN

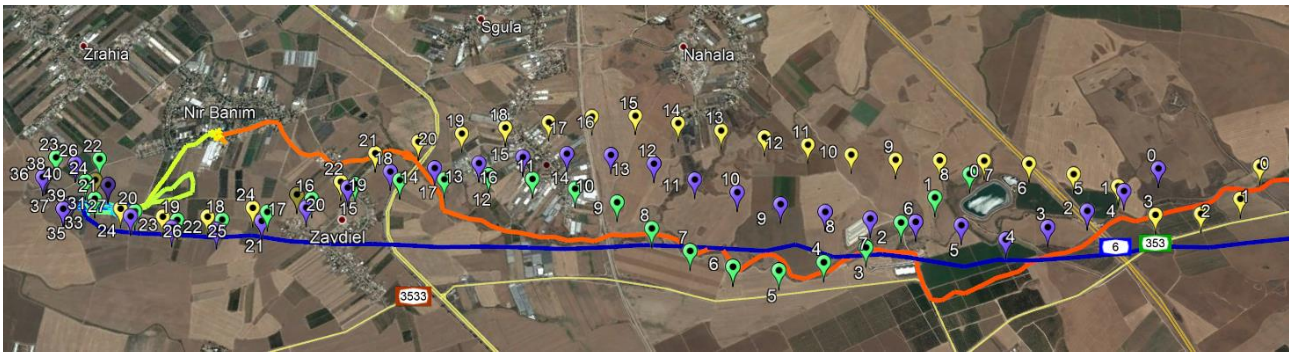

Figure 23 Navigation Simulation based on all training data

And here is a sample of three simulations that was made based on part of the training data CNN

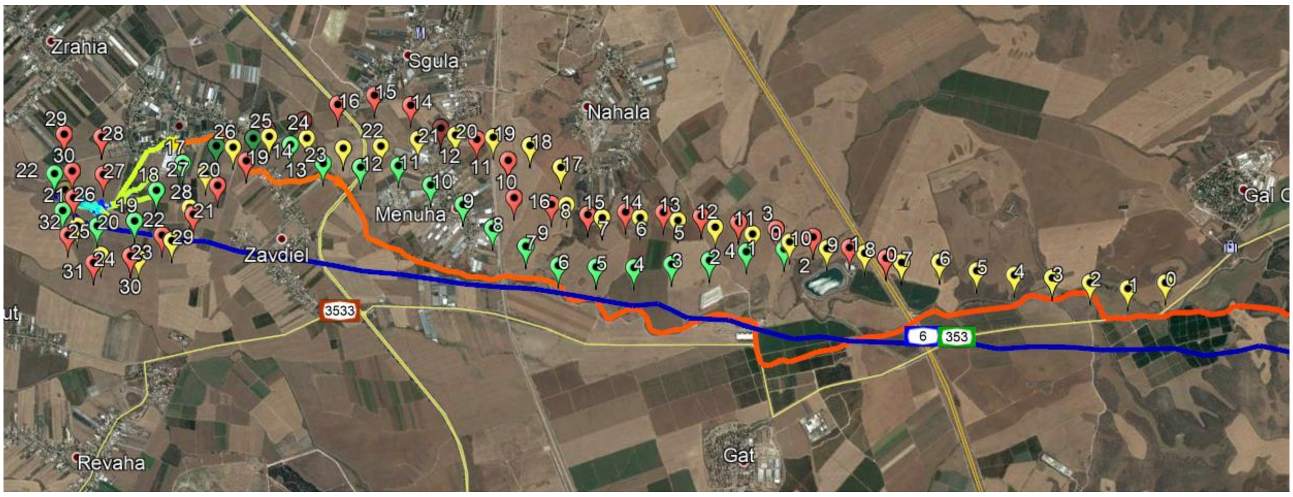

Figure 24 Navigation Simulation based on part of the training data

As expected the first CNN that was trained on all the training data preforms better around the missing training data

8 Discussion and Conclusions

In this project I tried to investigate the resemblance between bats vision and the trained convolutional neural network

From the test results we saw that the CNN error has a minimum in the direction of the fruit tree as can be expected as most of the training data is directed in that azimuth.

The concealment test leaned us that the most important part for orientation is in the middle of the input image, maybe because most of the horizon lights located there and in dark areas in the path most of the information is located in that section.
Another observation is that the upper section is more important than the bottom one because it is more stable and don't variant a lot.

We can see that although the accuracy rate of the CNN is not very high it can still be used to navigate from the cave to the fruit tree along the bats path.
This may suggest that the bat doesn't necessarily need to be 100% oriented in his flight to the fruit tree, As he is still able to orient itself based on several identifiable objects along his way like Kiryat Gat settlement.

In conclusion I think we can learn from this project that although a CNN is far from a brain of an animal, But by training it with close as possible database to what a bat see we can still use it to figure out elements that are important in orientation and navigation based on visual sensing.

9 Future Work

There is some additional work that can be done further on:

One is continue to investigate the neuron test results and try to get more information out of it.

This project suffered from lack of training data thus the CNN accuracy was not high and I had to use transfer learning in order to be able to train a basic model like VGG19, I think that to be able to properly train a CNN we need much more training data.

But even if I had more training data I was limited by my home computer hardware so I could not train and test in a reasonable time a deeper and more complex architectures that could have a better accuracy results. With a proper hardware a more complex architectures can be tested.

Another enhancement is by collecting and training the network with data in the opposite direction and mange to create a simulation that can navigate in both directions like a bat can, thus making the CNN navigation test closer an actual bat capability.=

10 Bibliography

[1] Machine learning mastery. "A Gentle Introduction to Transfer Learning for Deep Learning" In: (2017). URL: https://machinelearningmastery.com/transfer-learning-for-deep-learning/

[2] Saxe et al. "Exact solutions to the nonlinear dynamics of learning in deep linear neural networks", 2014

[3] Karen et al. "Very Deep Convolutional Networks For Large-Scale Image Recognition", 2015

[4] Medium. "Dropout in (Deep) Machine learning" In: (2016). URL: https://medium.com/@amarbudhiraja/https-medium-com-amarbudhiraja-learning-less-to-learn-better-dropout-in-deep-machine-learning-74334da4bfc5

[5] Wikipedia. "Gradient descent". URL: https://en.wikipedia.org/wiki/Gradient_descent

[6] CS231n. "convolutional Neural Networks for Visual Recognition". URL: http://cs231n.github.io/neural-networks-3/#ada

[7] Duchi et al. "Adaptive Subgradient Methods forOnline Learning and Stochastic Optimization", 2011

[8] Movable Type Scripts. "Destination point given distance and bearing from start point ". URL: http://www.movable-type.co.uk/scripts/latlong.html

[9] Wikipedia. "Radon Transform". URL: https://en.wikipedia.org/wiki/Radon_transform

[10] Wikipedia. "Autoencoder". URL: https://en.wikipedia.org/wiki/Autoencoder

[11] Matthew el al. "Visualizing and Understanding Convolutional Networks", 2013

[12] GitHub. "CNN visualization tool in TensorFlow". URL: https://github.com/InFoCusp/tf_cnnvis

11 Appendix A: Unused Work

11.1 Unused database

Before I found out how to extract the drone's internal log and synchronize it to the video recording I used more complicated method to label the database.

In the first recordings we used DJI phantom 3 drone with a remote control that connects to an android phone. The drone recorded the video to its internal SD card and broadcasted it over Wi-Fi to the controller android phone as can be seen in Figure 26.

I used screen recording app to record the screen of the controller phone. In the bottom left corner phone's screen is a cursor that represent the drone's current azimuth.

First thing I did was to reduce the frame rate of the phone using video editing application so it will be equal to the drone's camera frame rate.
Next I used computer vision program I wrote in Matlab that did the following:

1. Crop the red cursor (Figure 27)
2. Filter it to enhance the darker line along its axis (Figure 25)
3. Use Radon transform [9] to extract it's angel
4. Smooth the extract azimuth graph

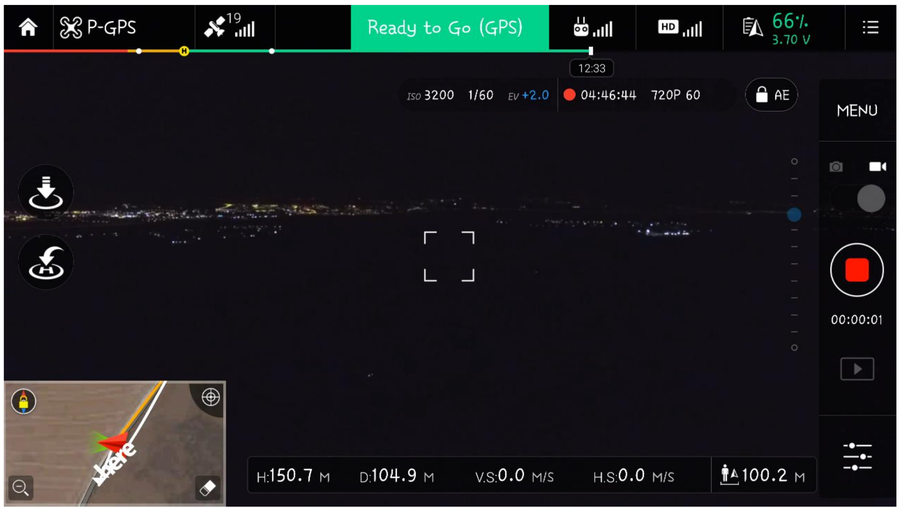

Figure 25 Phone recording

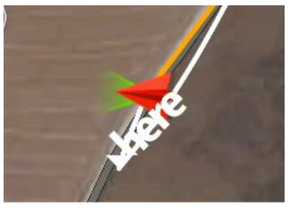

Figure 26 cropped arrow

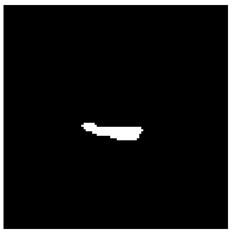

Figure 27 filtered arrow

Syncing manually both camera video and extracted azimuth was a difficult and not very accurate method of labeling.

There were several major problem with this method. The first issue is the phone screen recorded video had a lot of jitters and freezes which made the synchronization task very difficult,
The second problem is that the Matlab program worked well on most of the frames but some of them had label covering the cursor axis line and it was not so visible in all frames.

The recording starting point in every recording must be fixed and precise.

All of the above entered errors to the labeling process which effected the learning of the CNN. Because of this I needed to discard four nights of recorded videos and Do it again.

11.2 Unused Architectures

Autoencoder [10]

One of the first ideas was to train unsupervised autoencoder with the recorded database, Afterwards use only the trained encoder part of the model and connect to it another classifier model with fully connected layers and train only the new layers with labeled images

This method was created because of the small database and because autoencoders are less deep than other architectures which makes the training process easier.

After training it on the whole path database the idea was to encode some input images, The encoded image was thought to be a condensed vector that stores all the important information of the image. By manipulating the encoded image in different ways and decode it to get some different meaningful outputs that I can research. For example how different types of transformations effect the output image, classifying the output image and compare its classification error with the original input image classification error etc.

But unfortunately after many attempts to make the training converge I was unable to achieve good accuracy results with code layer that is smaller than the original input image.

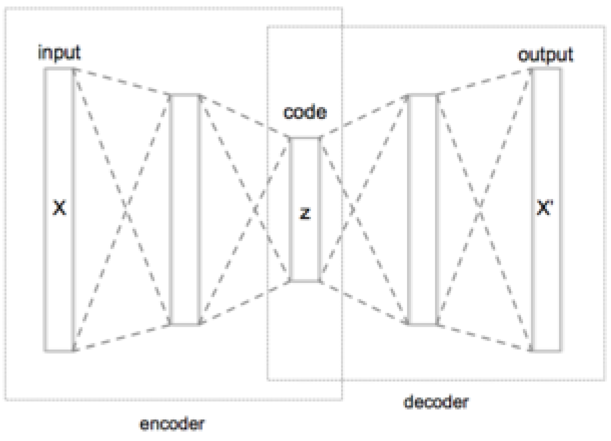

Figure 28 Autoencoder [10]

VGG16

Before I ended with the custom VGG19 model I tried first the original VGG16 model with 360 classes while each class represent an Azimuth out of the 360. But the test result of it was inferior to the VGG19 regression model so I quickly abandoned this model.

11.3 CNN visualization

One of the ways we tried to figure what the CNN has learnt and from that to get some conclusions regarding the bats was CNN visualization.

This method based on the paper of by Matthew D. Zeiler and Rob Fergus [11] reconstructs the input image from the information contained in any given layers of the convolutional neural network.

Here is for example input image that was reconstructed

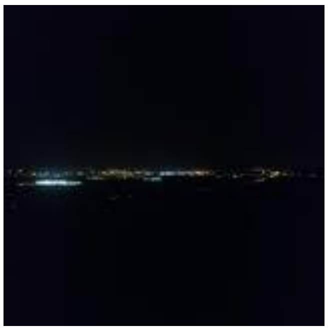

Figure 29 CNN visualiztion input image

And here is the reconstruction of the first 64 channels layer in the model

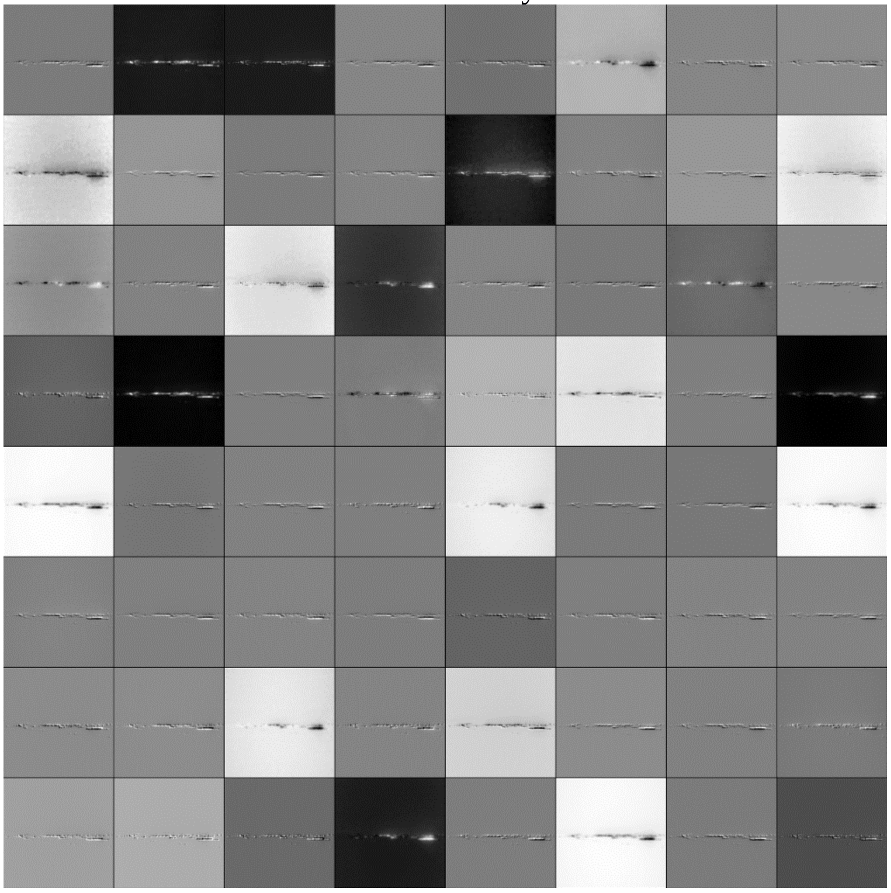

Figure 30 CNN visualiztion reconstraction

To do it I used Tensorflow library from [12]
Unfortunately this method did not add much information to our understanding.