LUMEN Data science

2022

Project Documentation

1 Contents

2 Business Understanding

2.1 Business objective

The business objective of this project is to design a program that can use an input of one or multiple images to predict geographical coordinates of the location where the pictures were taken. At minimum, the customer expects a REST API that can be used to retrieve the coordinates but a web application that can visualize the results would be appreciated.

2.2 Situation Assessment

2.2.1 Inventory of resources

2.2.2.1 Personnel

The team consists of multiple members. All of the team members have experience in programming in multiple languages including Python. Some members have a history of solving computer vision tasks which will be crucial for solving this task. Also, some members have experience in creating web applications and REST APIs which will be used in this project.

2.2.2.2 Data

Data is provided by competition organizers. It consists of 16000 annotated images. More data can be acquired online if necessary. Data will be more thoroughly discussed in the next chapter.

2.2.2.3 Computing resources

We have two GPUs at our disposal. One is a Nvidia K80 provided by Google and can be accessed in Google Colab. The other is a NVIDIA GeForce RTX 2070 provided by one of the team members.

2.2.2 Requirements, assumptions and constraints

2.2.2.1 Requirements

There are two main requirements for the project. First is to provide customers with a REST API that can be used to retrieve the predicted results. As mentioned earlier, a web application would be beneficial. Second requirement is the deadline which is set for May 6, which was later extended to May 8.

2.2.2.2 Assumptions

Only assumptions of the project are that the data provided is clean and that the location of the images truly corresponds to their locations in real life. Assumption about data being

clean will be verified and talked about in the next chapter. The assumption about location of the images remains unverified since it would be very hard to verify this assumption for 64000 images.

2.2.2.3 Constraints

The constraints of the project are mostly related to availability of GPUs. GPU provided by Google Colab has a time limit assigned both for one session and overall usage. Exceeding overall usage limit can lead to a lengthy ban from using GPUs in Colab which is what ultimately happened to us.

2.2.3 Risks

The biggest risk was the lack of training equipment. Google Colaboratory was our primary way of training the model and receiving a ban could slow down the project. During development we managed to access one of our teammates GPUs reducing the impact of this problem on development. Another significant risk was that team members wouldn’t have enough time to focus on the project since all of us are university students and some are employed.

2.3 Success criteria

To define the success criteria we decided to compare the performance of our team members with the model's performance. We think that a useful model is a model that can outperform humans. Our mean error was 105 km.

3 Data Understanding

3.1 Data Description

Data is given in two parts; one being a .csv file containing annotations and the second being a folder with images. The .csv file consists of three columns: uuid – a randomly generated name of a folder where the images for the location are located, latitude, and longitude of the location both expressed in degrees. The dataset consists of 16000 folders each containing four images. The four images represent the same location but with a point of view shifted by 90° starting from 0°. The dataset is supposed to cover only Croatia, although Croatian islands should be excluded.

3.2 Data Understanding

3.2.1 Data visualization

To visualize the data we plotted the coordinates on top of a map of Croatia, as can be seen in *Image 1*. We can already see that the dataset truly does not include islands, however some locations are outside Croatia. We will further explore outliers in following chapters. The data seems to be evenly distributed across all of Croatia which came as a surprise to us as we expected it to be grouped around some bigger cities.

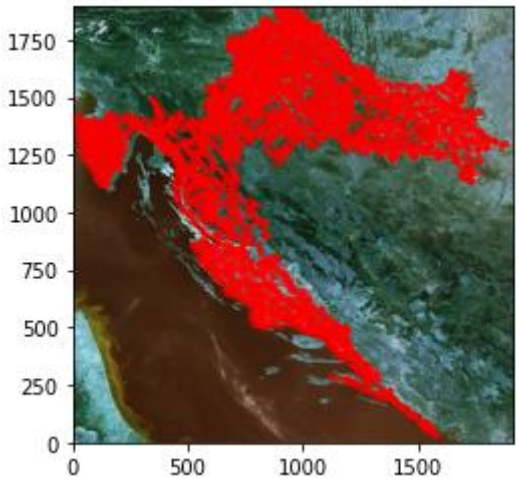

Image 1 - Data visualization

Next, we visualized a few images to see what parts of the picture we would take into account if we had to guess the location. Our guess to what could be important in these images is the architecture of the buildings if there are any and most importantly the vegetation. In most cases both attributes are located on the edges of the image, some examples of this can be seen in *Image 2*. This significantly impacted our choice of data augmentation which will be discussed later.

Since the data is discrete, it is important to visualize its distribution. To achieve this, we plotted a histogram of latitude and longitude which can be seen in *Images 3* and *4*. The data is extremely unevenly distributed, especially for latitude. This was to be expected as the shape of Croatia is quite unusual.

Image 2 - Data examples

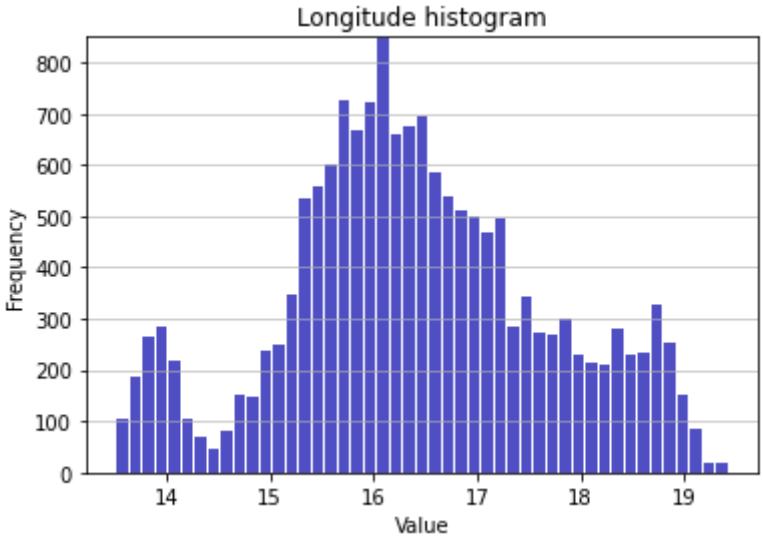

Image 3 - Longitude histogram

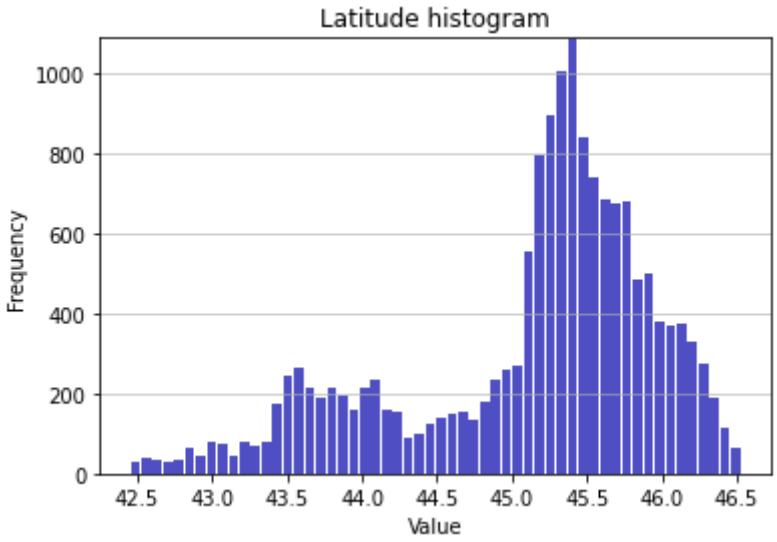

Image 4 - Latitude histogram

3.2.2 Data Quality

2.2.2.1 Missing data

The .csv file truly contains 16000 rows as expected. No rows or columns are missing and there are no empty cells, None values or altering data types. When trying to open images we found that two folders from the csv file were missing. This is not something that was concerning to us since two folders are negligible in a big dataset like ours.

2.2.2.2 Outliers and dirty data

We already mentioned that some locations are outside Croatian borders. There are a few outliers near the border with Serbia and a few near the border with Bosnia and Herzegovina. We visualized the outliers with blue circles in *Image 5*.

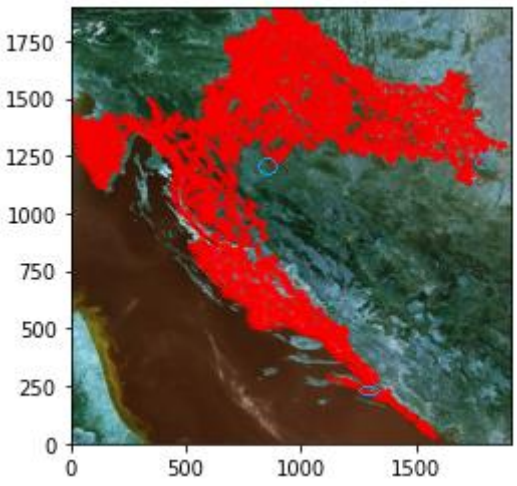

Image 5 - Outliers visualized on a map

4 Data Preparation

4.1 Data cleaning

In chapter 2.3.2.2 we identified some outliers. In the end we decided not to remove the outliers from the data. There were two reasons for this decision. First being that these outliers are very close to the Croatian border. Since we feel that it is impossible to make a model that is precise enough for such small errors to significantly impact the final result, we decided not to remove them. Second reason was that even if we did try to remove them our assessment was that we would lose more by removing clean data then we would gain by removing outliers.

4.2 Data Labeling

For our classification model to work we needed to create classes. We have decided to divide Croatia into six regions based on geographical features that we feel are specific to each region. The region boundaries are given in *Table 1* and are visualized in *Image 7*.

Region	North boundary	East boundary	South boundary	West boundary
Istria	45.14° N	15.06° E	44.27° N	13.50° E
Lika	45.99° N	16.34° E	44.27° N	15.06° E
Dalmatia	44.27° N	18.53° E	42.40° N	13.50° E
North Croatia	46.55° N	17.64° E	45.99° N	15.06° E
Slavonia	45.99° N	19.45 ° E	17.64° N	17.64° E
Central Croatia	45.99° N	17.64° E	44.27° N	15.06° E

Table 1 - Region boundaries

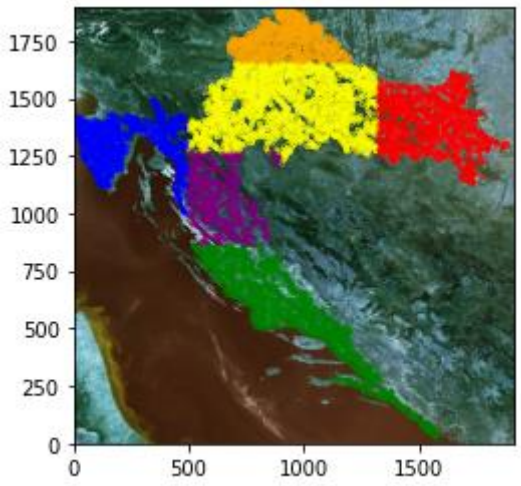

Image 6 - Visualized region boundaries

4.3 Data Balancing

It is important for data to be properly balanced before feeding it to the model. After running a quick summary of the data classes, we gained the following results:

There are 2626 points in Slavonia
There are 1953 points in Istria
There are 1869 points in North
There are 3109 points in Dalmatia
There are 5066 points in Central
There are 1377 points in Lika

We can see that Central Croatia has the most locations by far and that Lika has very few. If we were to feed the model unbalanced data like this there is a high chance it would learn to only predict Central Croatia and Dalmatia and wouldn't actually learn the features of the data.

The two commonly used methods for balancing data are upsampling and downsampling. Upsampling is a method where we add data to a class either by finding new images or by repeating already existing images multiple times until all classes have roughly the same number of images. Downsampling is a method where we remove data from the dataset until all classes have roughly the same amount of data. Since Google Maps data is copyrighted, we couldn't get new data. Repeating the data could lead to overfitting – model learning to classify training images correctly but failing to classify images it has not seen before. Downsampling can also lead to overfitting if we remove too much data. In the end we decided to downsample the dataset to 1700 images per class. Lika is still a bit underrepresented however we decided that downsampling to 1400 images would lose too much data.

4.4 Histogram Equalization

In chapter 3.2.1 we visualized distributions for latitude and longitude by plotting a histogram. It is clear from the histograms that their distributions are skewed. Similarly to the situation where we had unbalanced classes, if we were to feed the model this data it would most likely only learn to predict longitudes and latitudes that appear more often.

One solution for this problem is a method called histogram equalization. First, we rescale the data to a range between 0 and 1. Next, we calculate what percentage of samples are in each bin. A bin is visualized as a stripe on a histogram and it represents how many values are in a certain range. We can then use the calculated percentage to determine a new range to which the data is to be scaled. For example, if the original bin contained one tenth of data, but had been assigned one fifth of the total range, data assigned to that bin will be rescaled so that it covers a tenth of overall data. The results are visualized in *Images 8* and *9*. As we can see the data is now equally distributed.

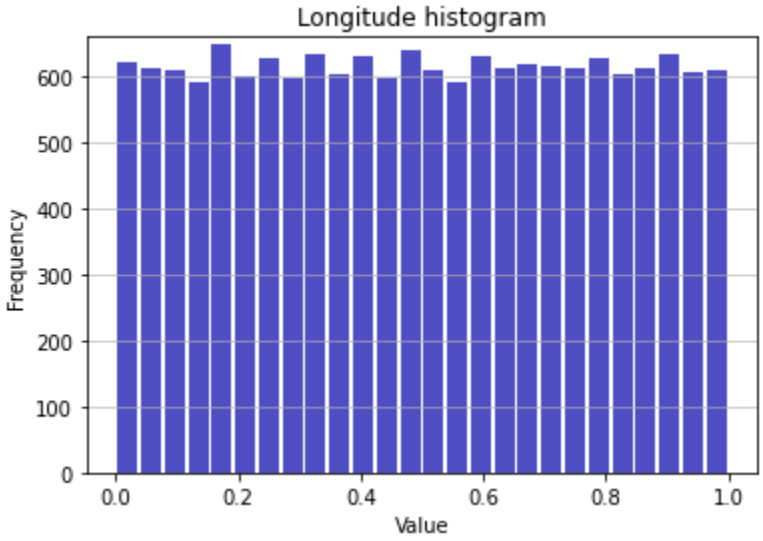

Image 7 - New longitude histogram

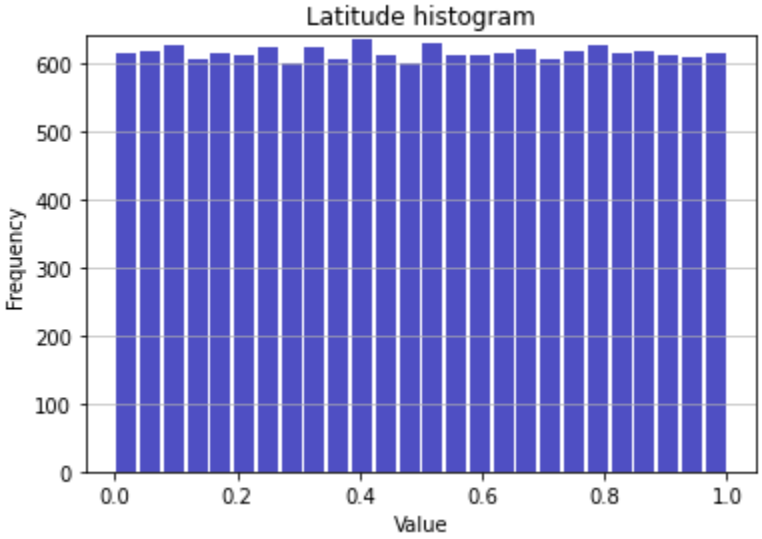

Image 8 - New latitude histogram

4.5 Data augmentation

Data augmentation is a way of increasing the dataset by adding slightly modified images to the dataset. Most common data augmentation methods are randomly cropping an image, randomly zooming in on part of the image, and randomly flipping the image vertically or horizontally.

In chapter 2.3.1 we mentioned that features we feel are important are mostly located at the edge of an image. We chose not to use random crop or random zoom because we thought there was a high chance it could crop out vegetation or zoom in on irrelevant part of the image once again removing important features from the image. We also decided that vertical flip was not going to be useful as that would mean that the image would be upside down which doesn't make much sense and is not expected as input.

In the end we ended up trying randomly changing brightness and randomly flipping the image horizontally. Random brightness did not have any result on the output and horizontal flip improved it by 1% (classification model). Since horizontal flip was the only augmentation method that had a positive impact on the result, we decided to only use horizontal flip.

5 Modeling

5.1 Introduction

Machine learning problems are usually divided into two groups: classification problems and regression problems. To solve the problem we decided to try multiple model architectures for both classification and regression. Two types of models that we tried are Transformer

models and Convolutional neural network (CNN) models. Transformer models that were tried were vanilla Vision transformer (ViT) and Shifted Window Transformer (Swin). The CNN model that was tried was EfficientNetB4. For all of our classification models we used sparse categorical cross-entropy as our loss function and softmax as our activation function. For all of our regression models we used mean square error as our loss function and sigmoid as our activation function (we can do this as our data is scaled from 0 to 1).

5.2 Creating The Dataset

To run any of our models we first need to load our dataset. First we split the data in two sets: training set containing 90% of the images and validation set containing 10% of the images. Our original attempt was to compress the data with Python's pickle module; however, that would require all of the dataset to be loaded into the GPU at once. This was not possible as the dataset is too large causing our code to keep crashing. Next we tried tensorflow's ImageDataGenerator but it slowed down training extremely. Finally, we settled for TfRecods - tensoflow's custom data format. With TfRecords files are stored sequentially, enabling fast streaming due to low access times. This had improved the training time significantly compared to ImageDataGenerator.

5.3 Convolutional neural networks

5.3.1 Introduction to Convolutional networks

Convolutional neural networks are a type of artificial intelligence that uses a mathematical operation called convolution instead of general matrix multiplication in at least one of its layers. They are most commonly used to analyze images and have been the most popular type of neural network for that task since release of AlexNet in 2012. CNNs usually use multiple convolutional layers with learnable filters to extract features from the image that it thinks are important. To extract the features, layers "constructively distort" the image using convolution and feed it to the next layer. In *Image 10*, a number recognition CNN was used to visualize what happens inside a CNN.

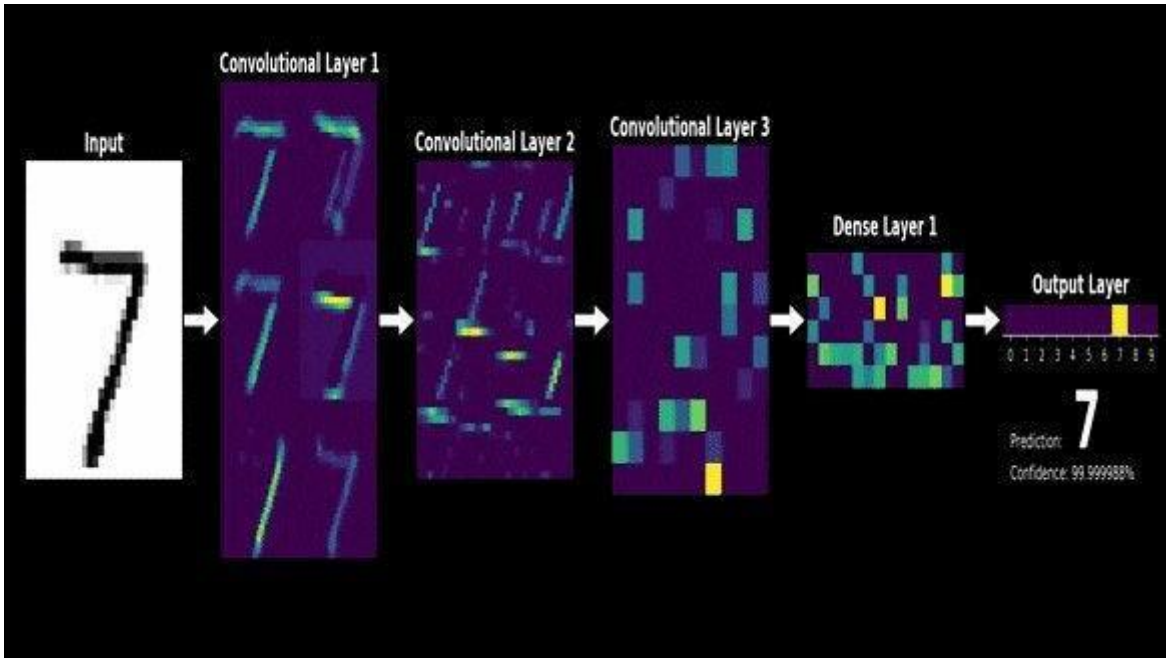

Image 9 - Visualization of a number recognition CNN

We can see that in the last layer the image is completely distorted and meaningless to humans, however the network is able to use it to correctly classify images. Due to convolution, CNNs only focus on the pixels close to each other. This inductive bias is both the main advantage and the main disadvantage of CNNs. Because of this bias CNNs have much less parameters then other networks like multilayer perceptrons making them easier to train. It also means they need less data to learn then models like transformers, however the bias somewhat limits their ability to generalize which can lead to poorer results.

5.3.2 EfficientNet

EfficientNet is a state of the art convolutional neural network architecture designed by Google. Its biggest advantage is that it is optimized for having the least amount of parameters necessary not to lose performance. Comparison of EfficientNets parameters and other popular CNN architectures is given in *Image 11*.

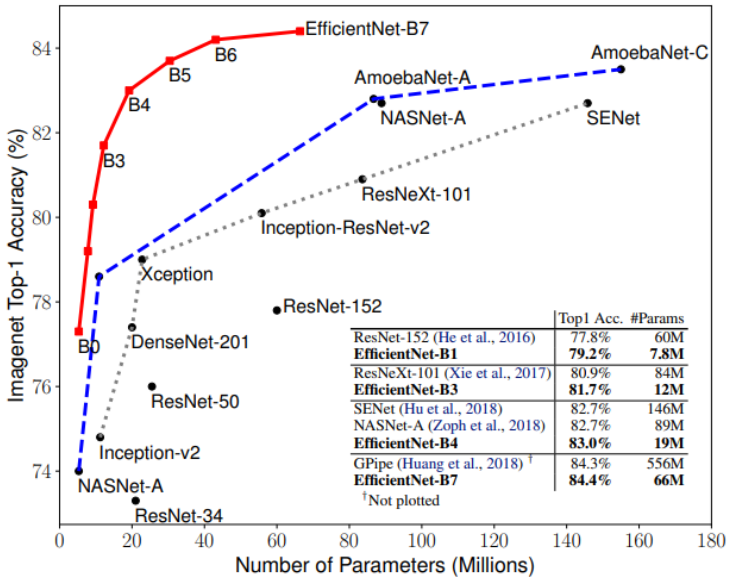

Image 10 - Comparison of EfficientNets parameters and other CNN architectures

EfficientNet has eight versions ranging from EfficientNetB0 to EfficientNetB7 with B0 being the smallest and B7 being the largest model. Following our experiments, we chose EfficientNetB4 as our architecture. For both the classification and regression model we used Adam as the optimizer with a learning rate of 0.01. We also added a dropout layer with a dropout rate of 0.1 to prevent overfitting.

5.4 Transformers

5.4.1 Introduction to Transformers

While CNNs dominated the computer vision field since 2012., a new type of model architecture was invented in 2021. called Vision Transformers. Transformers are a machine learning architecture designed in 2017. that was most commonly used in natural language processing. In 2021. a paper was released that managed to adapt the Transformer architecture to computer vision problems. Ever since the various Transformer architectures are replacing CNNs on an increasing number of problems. Transformers' biggest advantage is that they can generalize much better than any other known model type due to a lack of bias. The same lack of bias that makes transformers so good at generalization also means the model can easily overfit if not fed enough data. Since our dataset is quite large, we figured this should not pose a problem in our case. A transformer architecture is given in *Image 12*.

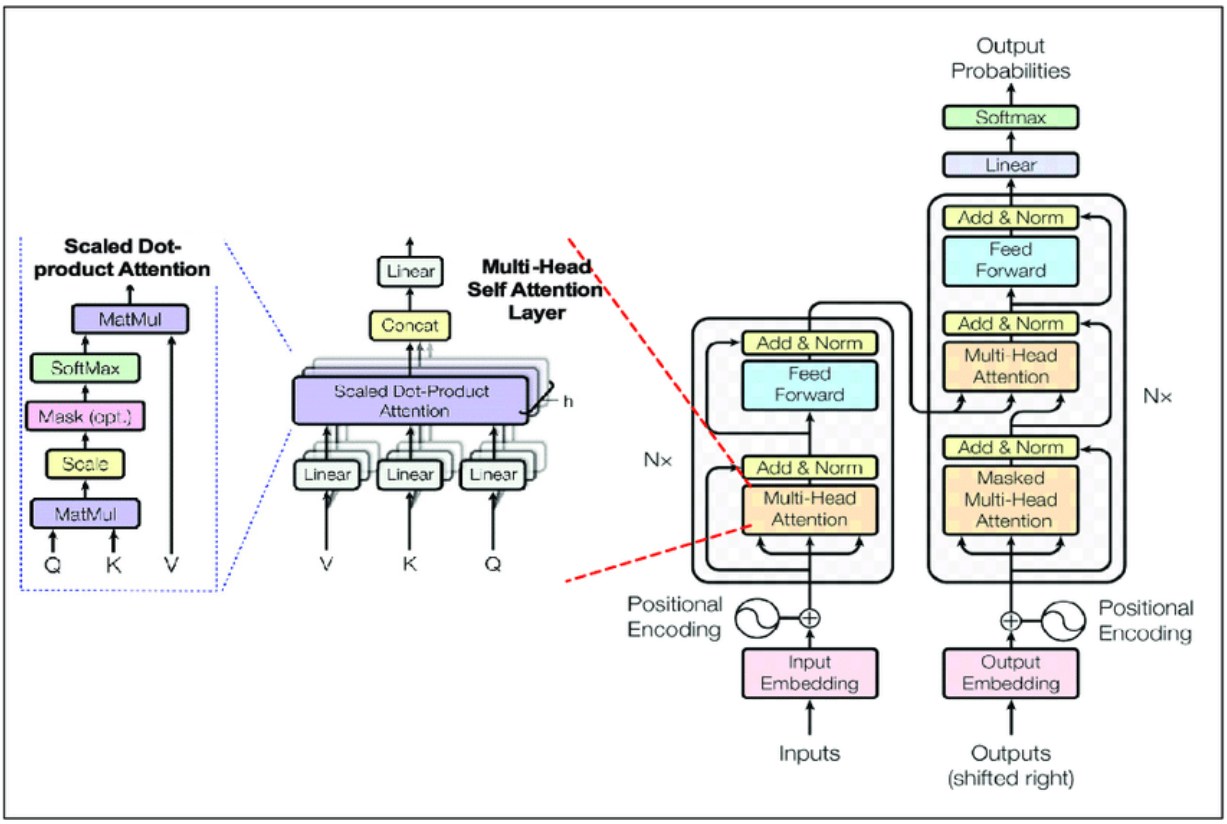

Image 11 - Transformer architecture

Transformer mechanics are quite complicated but an adequate simplification is that the model takes an input of data and creates some internal features called tokens. It builds a key, value, query matrix that is used to calculate self-attention – a matrix of numerical values representing the connections between each pair of tokens. In the end when we want to make a prediction we feed data to the model which creates a query matrix. Query matrix is then multiplied by the keys matrix to obtain the value matrix that is used to represent our prediction.

5.4.2 Vision Transformer

One of the most popular Transformer architectures for computer vision is the ViT. This is also the first good Transformer meant for computer vision tasks. The reason why it was hard to implement a Transformer that takes images as inputs was because the cost for creating tokens is quadratic. This property makes it impossible to implement Transformers for realistic input sizes. To combat this ViT uses something called patch tokenization. Instead of calculating self-attention for each pixel, the image is divided into patches of custom size and relations are created in between these patches. *Image 13* visualizes patch tokenization on a random picture from our dataset.

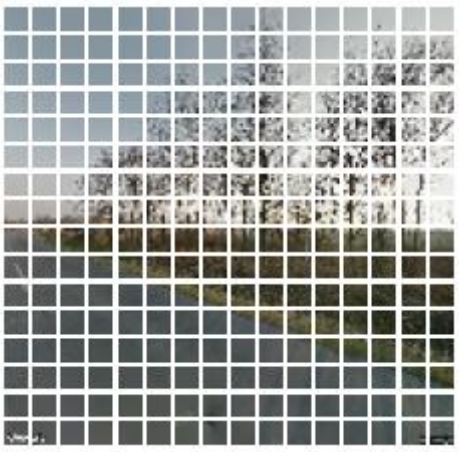

Image 12 - Patch tokenization example

Instead of having 16384 tokens ($128 * 128$ pixels) we have 256 ($\frac{16384}{8*8}$) (patches are of size 8 x 8). These patches are then fed into the original Transformer architecture described earlier.

5.4.3 Shifted Window Transformer

Another popular transformer architecture is the Swin Transformer. A problem with ViT can arise when trying to look at small details. Due to the way patches are extracted they can often be too big for these small details, meaning that attention between patches cannot be properly formed. For example, if our goal was to identify people that are far away in a large image choosing too big of a patch size can end up with all the people grouped in one patch making our model unable to create relations for each person. Choosing to small of a patch size means our model can have too many tokens making it too big. To combat this the Swin architecture doesn't use a fixed patch length. It starts with small patches and merges them as it gets to deeper transformer layers. This property allows Swin models to focus both on the small details and the overall context of the image. Merging of smaller patches into bigger ones is similar to the way CNNs work which is why Swin is often described as something in between a Transformer and a CNN.

5.5 Small Regression Model

For our classification models we still need to find a way to turn classes into coordinates. We tried two approaches, one being setting a fixed point in the center of every region to be outputted every time that region is predicted. The second approach is to create a small regression model that will take classification models output as input. Instead of giving the model only the predicted class as input we decided it would be better to give the model the distribution for each class as it contains more information. The model itself has an input layer, one dense layer, and an output layer. We used Adam as our optimizer with a learning rate of 0.001.

5.6 Visualizing Results

5.6.1 Visualizing predictions

In the end we decided to visualize a few predictions for the EfficientNet model. They are shown in *Image 14*. The red dot represents the predicted location and the blue dot represents the true location.

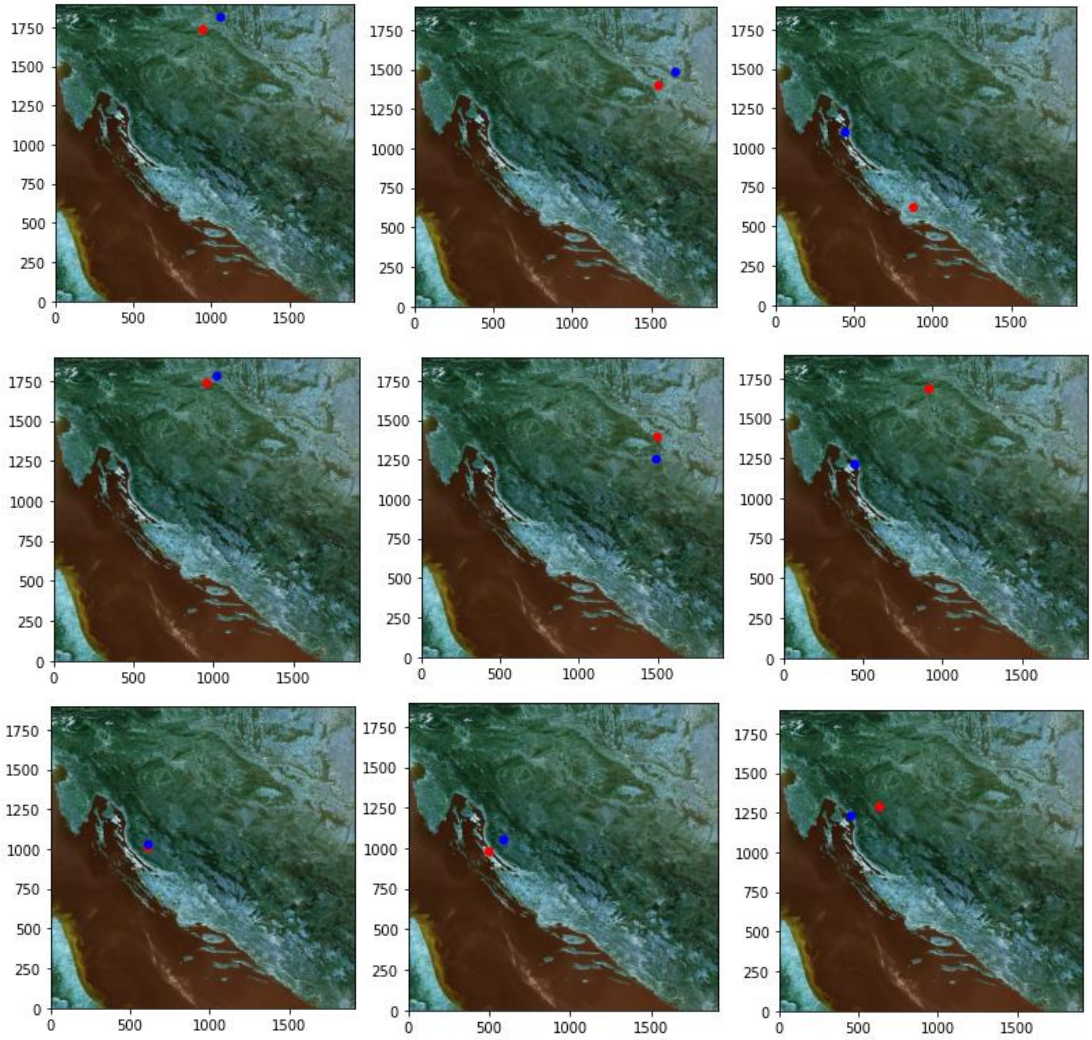

Image 13 - EfficentNet model prediction examples

In the end we were pleasantly surprised by how precisely the model is able to predict the locations. The images visualized were nine random locations and were not handpicked.

5.6.2 GradCam

GradCam is a popular visualization technique in machine learning. We can use it to visualize what our network thinks is important on an image.

Image 14 - GradCam visualization

Color represents how much attention our model gives to a part of the image with red representing the most important parts and blue representing the least important parts. In the first row we can see that the model focuses on the vegetation as we suspected in chapter 3.2.1. In the second row there is no vegetation that can be used to identify the location so interestingly the model chose to focus on the sky.

6 Evaluation

6.1 Results

In the end directly using regression turned out to give better results than using classification with a small regression model. CNNs gave better results than transformers. Multiple setups were tried for both Transformers. Best classification results for every architecture are listed in *Table 2*.

Architecture	Accuracy
EfficientNet	70%
Swin	60%
ViT	58%
EfficientNet with transfer learning	40%

Table 2 - Classification results

Since transformers performed so much worse on classification than EfficientNet we decided not to apply a small regression model on it after classification. Regression results are depicted in *Table 3*.

Architecture	Classification (MSE)	Regression (MSE)
EfficientNet	0.038	0.033
ViT	n/a	0.048

Table 3 - Regression results

We can see that, on this problem, for both classification and regression EfficientNet has the best performance by far. Using fixed locations for every class resulted in a mean squared error (MSE) of 0.13.

6.2 Review Process

In retrospect the process seems to be generally properly executed. What we wished we did differently is to use a combination of upsampling and downsampling in balancing data instead of just downsampling. Also, we would have loved to try even more Transformer setups as properly choosing parameters tends to play a more important role in Transformer architectures than in CNNs. We were unfortunately limited in this regard by a lack of GPU time.

6.3 Next Steps

We still believe that Transformers have a lot more to offer than shown. Our next steps would be to try more configurations for both transformer architectures. We would also try to increase the dataset even more if possible.

7 Sources

Image 1 - produced by team members

Image 2 - produced by team members

Image 3 - produced by team members

Image 4 - produced by team members

Image 5 - produced by team members

Image 6 - produced by team members

Image 7 - produced by team members

Image 8 - produced by team members

Image 9 - https://external-preview.redd.it/eh8jmzR3w7uyTlrNlsPHXXBfgYhqRNLSVER1sdeCZD4.jpg?auto=webp&s=fa22e670fa22dc60236dfe29b67b0dd3ddf6fcc4, accessed 5.05.2022.

Image 10 - https://miro.medium.com/max/662/0*09AED_CjE-PUFxKC.png, accessed 5.05.2022

Image 11 - https://www.researchgate.net/publication/342045332/figure/fig2/AS:900500283215874@1591707406300/Transformer-Model-Architecture-Transformer-Architecture-26-is-parallelized-for-seq2seq.png, accessed 5.05.2022

Image 12 - produced by team members

Image 13 - produced by team members

Image 14 - produced by team members