LUMEN DATA SCIENCE 2022
Project documentation

GeoGuesser AI Agent

Zagreb, May 2022

1. Problem understanding

In this paper, we will describe our solution to the problem presented in the 2022 LUMEN Data Science competition organized by eSTUDENT association. The task was proposed by it's premium partner Photomath. The task is to develop an AI agent for deciding on a geolocation based on a 360 view of a place - anywhere in Croatia. The view is presented with 4 images, therefore, each set of 4 images is a sample of our data.

The task is inspired by GeoGuessr, a geography game where you get taken somewhere in the world and are presented with a street view 360. Your task is to decide where on the map of the world were you taken to. GeoGuessr provides several versions and game modes differentiating in (among other things) means of possible locations you are taken to. For example, you might choose to play only on the territory of Spain, or the city of Paris, or you can choose to guess where the UNESCO heritage site you are presented with is. The use-case for this competition is bound only to guessing locations in Croatia. Figure 1.1 shows the Geoguessr game interface.[1]

Apart from getting the best score on this game and beating your friend's high-score, this solution could be useful for other tasks, including recognition of historical sites, recognition of places for crime investigation or others.

As you probably realize by now, this is an image recognition problem. While human and animal senses recognize locations and objects with ease, computers have different abilities and require additional resources to perform this kind of task. Some kind of Artificial intelligence is required to acquire information from images, and nowadays Convolutional neural networks perform best on problems like this one. These models need to be trained on a large amount of data and they usually work differently from living brains. They use pixels, specific parts and features in an image to come up with a decision, whether it's a classification or a regression problem.

Image recognition is a subcategory of computer vision and artificial intelligence. It represents a set of methods for detecting and analyzing images to enable the automation

[1] `https://tinyurl.com/mrys35ze`

Figure 1.1: Geoguessr interface.

of a specific task. In theory, image recognition is based on deep learning. Similar to a human brain, a deep network consists of connected neurons, but these neurons compute outputs of mathematical functions. During the learning process, they try to assign the best fit for the parameters of these functions to perform best on the given task. The complexity and the architecture of a model depend on the complexity of the problem, and image recognition tasks require rather complex models.

2. Data understanding

Data provided for this use case is a set of images - geolocation pairs. Each sample consists of 4 images (4 cardinal directions) and a corresponding label (geographical latitude and longitude). Considering our data is labeled, the best solution for this problem would probably be a supervised machine learning algorithm.

As mentioned, 4 images represent one sample of data and map to one geolocation. Geolocation is represented with geographical coordinates - width and length.

Most of the images are of roads and the sky (Figure 2.1 and Figure 2.2).

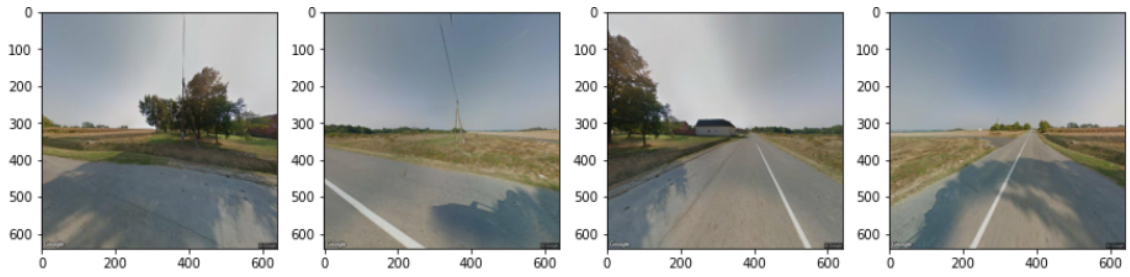

Figure 2.1: Photos of a road and the sky.

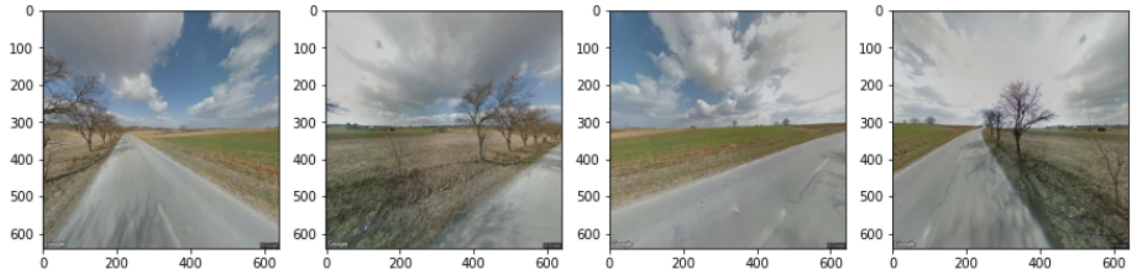

Figure 2.2: Photos of a road and the sky.

Some images are taken in nature; an example can be seen in Figure 2.3). Rarely, images are taken in an urban or a rural area where you can see buildings or houses (Figure 2.4).

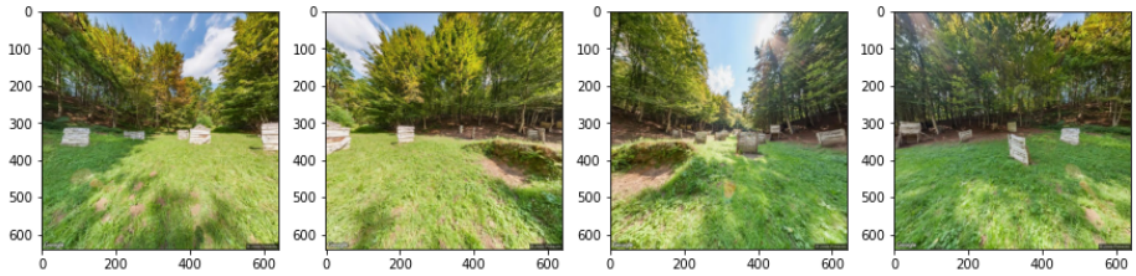

Figure 2.3: Photos taken in nature.

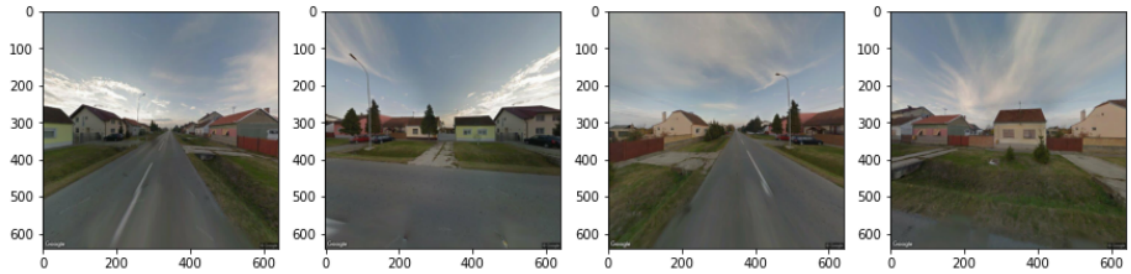

Figure 2.4: Photos taken in a populated area.

A few images are taken inside houses or apartments, possibly taken from listings on housing sites. An example of these kinds of images can be seen in Figure 2.5.

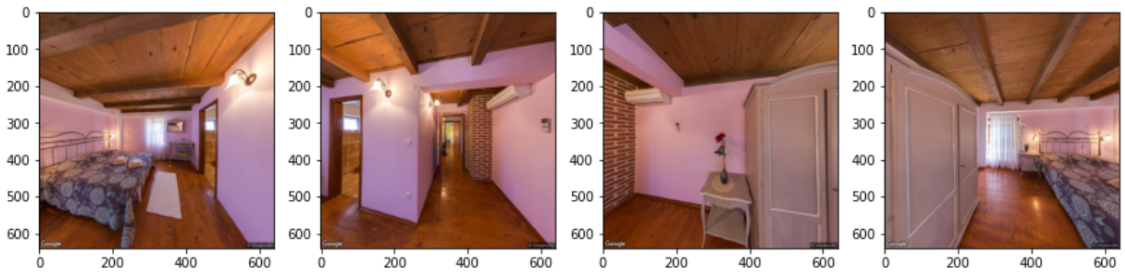

Figure 2.5: Images taken in an interior space.

As mentioned, 4 images represent one sample of data and map to one geolocation. Geolocation is represented with geographical coordinates - latitude and longitude. We were given a data folder containing sub-folders with 4 images each. The following image 2.6 represents a sample of a *.csv* file with the corresponding labels for each sample of 4 images. Images are given an identifier which can be seen as the name of the folder they are stored in. This identifier is stored in this CSV file under the column "uuid".

	uuid	latitude	longitude
0	69387a76-b6f6-4a76-9d82-59367e14cb12	45.552228	18.538397
1	83fd0354-8781-4325-9139-653ba0ce718f	45.116326	14.821817
2	5e2f692d-a2e6-45b1-b18b-3cec90b31b64	45.424986	18.767853
3	b3447ea2-8ea2-4c4e-b2a8-8611c8253995	46.154501	17.057093
4	93c8620f-2e97-4fe1-999a-c4c423b3d878	43.651482	16.323894
...	...	...	...
15995	717d023f-52d3-447c-8746-730d7e555fe6	45.424661	13.931598
15996	0ad4bcfa-4aa4-4826-88ec-2692a2906132	45.420424	15.348075
15997	abd34a3c-30cf-462b-938a-9ef2ac9b9a8b	45.632118	17.207108
15998	848a0d8b-286b-4b78-b694-57924eaf52aa	45.065947	14.123027
15999	8871c47b-9dae-4f0d-a373-f685cfda28e9	46.200279	17.060700

16000 rows × 3 columns

Figure 2.6: A preview of a given csv file with labels.

A few maps of Croatia were created to visualize data and see potential outliers, and spots located outside of borders. The original shapefile of Croatia can be seen in Figure 2.7.

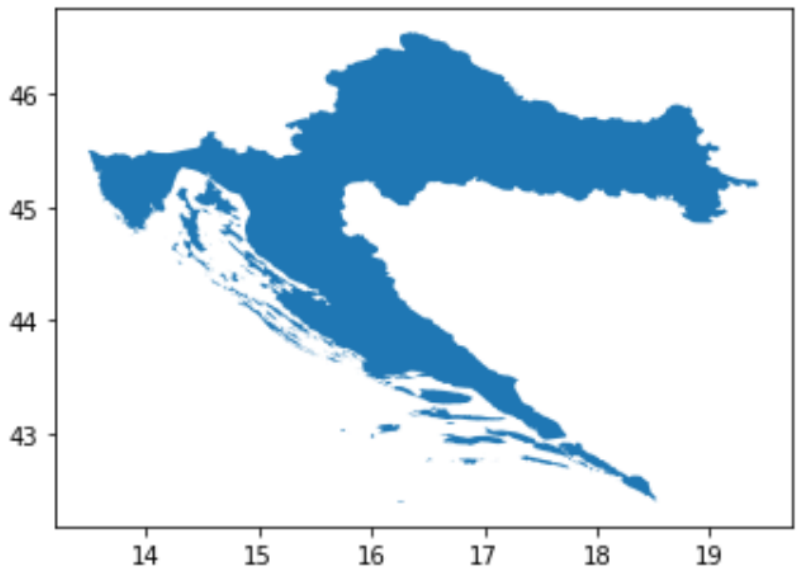

Figure 2.7: A map of Croatia from an original shapefile.

Croatian coast is one of the most indented coasts in the world and there are many islands. None of the data present in the dataset is located on any of the islands, so we decided to exclude them from our analysis. Firstly, we detected polygons containing the mainland of Croatia. In Figure 2.8 marked with red 'x' markers, you are able to see the polygons chosen for the further analysis. As Croatia's mainland is separated in Dubrovnik-Neretva county bordering Bosnia and Herzegovina, there are two polygons we find interesting to consider in our analysis.

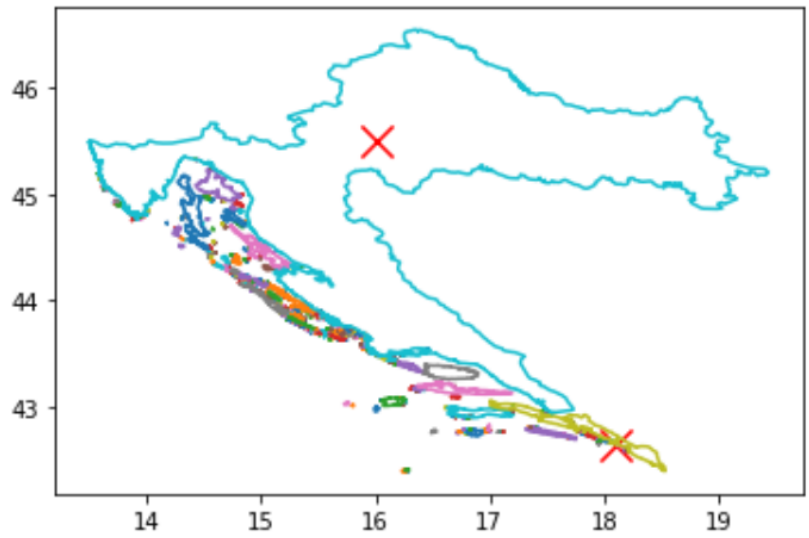

Figure 2.8: A map of Croatia with polygons detected for exclusion.

Figure 2.9 presents only the mainland of Croatia used in our model.

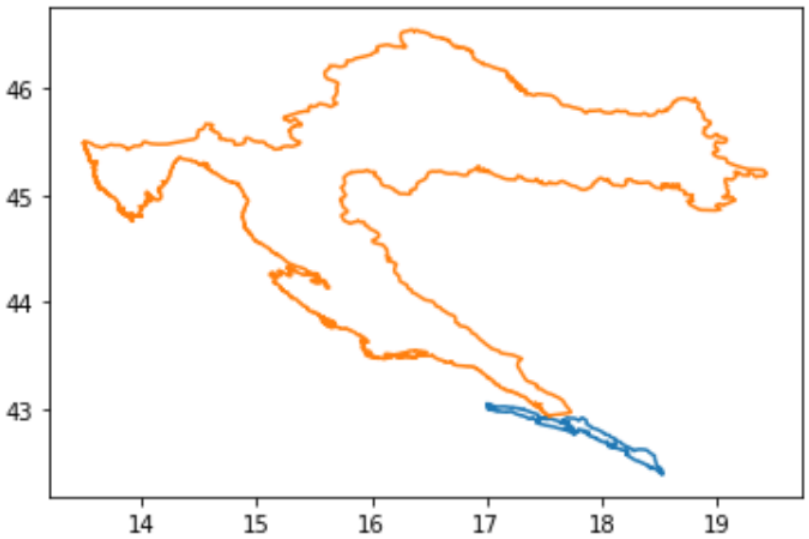

Figure 2.9: The final map of Croatia used in the model.

Finally, Figure 2.10 shows the map with data scattered along the surface. The blue dots are located outside of the borders of the polygon, and they are considered to be

outliers.

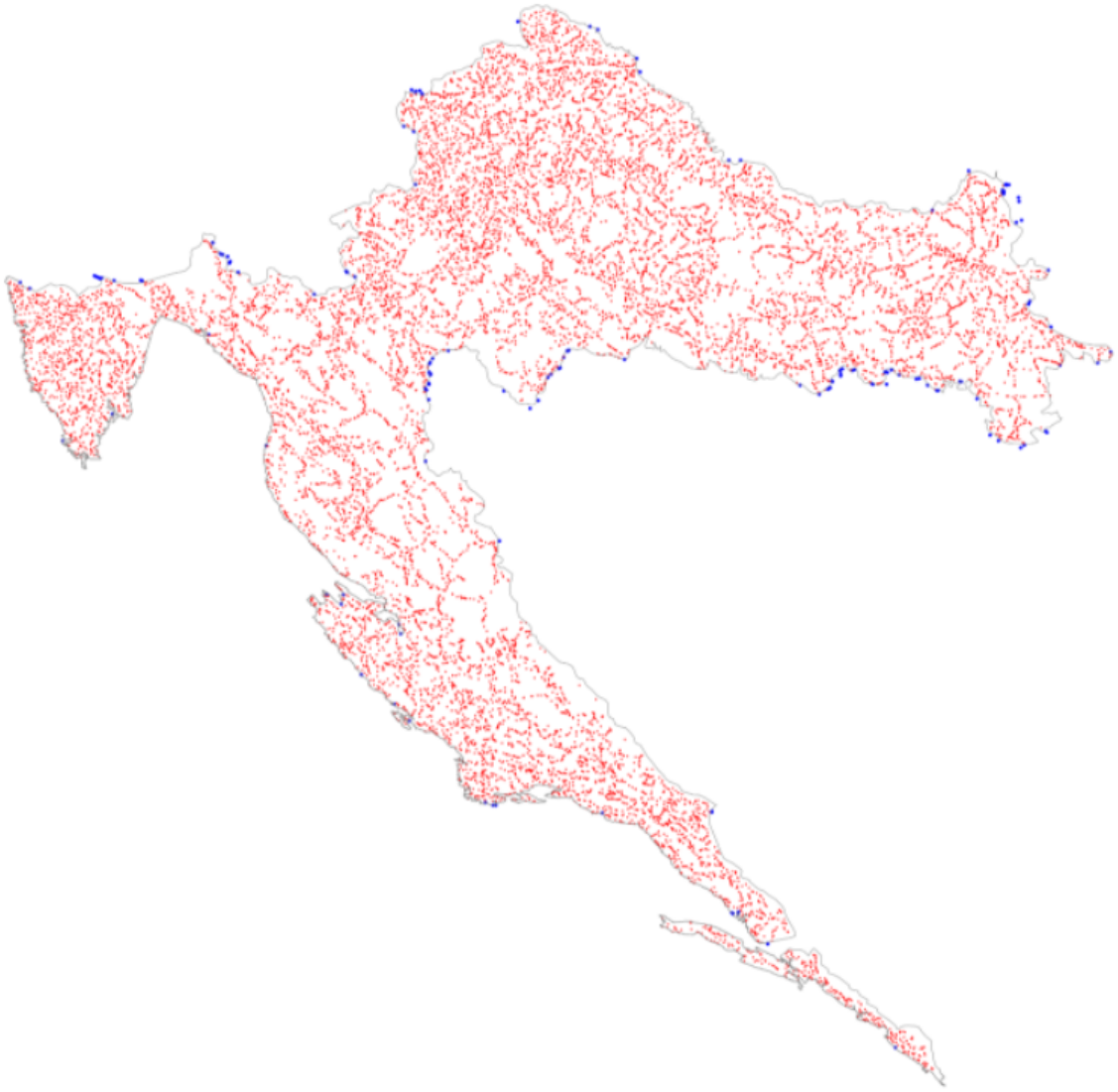

Figure 2.10: The final map of Croatia with scattered data.

3. Data preparation

Throughout the previous chapter, we analyzed the given dataset and data structure. In this chapter, we will remove outliers and crooked images from the dataset and describe all the necessary steps to prepare data for model training.

The first step in data preparation was the removal of outliers and images taken in interior spaces. We did that by removing those rows from the "data.csv" file, which connects images with their labels by column "uuid". Some of those images were kept just to ensure enough noise in our data. Finally, a total of 181 samples of data (4 images per sample equals to a total of 724 images) is eliminated and we are left with 15819 rows in the "data.csv" file (63276 images).

The next step was to divide Croatia's surface into smaller areas. We choose to use K-means algorithm for this task and the choice for the number of areas landed at 8. This decision was obtained by using the elbow and silhouette analysis, alongside our intuition and knowledge of Croatia's landscape. Also, splitting data into more classes could maybe yield better results, but as you will realize later in this paper, each of these classes, or say data in these classes, will be trained on a different model. Having too many partitions would then lead to having a large number of models, which is something not feasible to us considering the available resources but also the amount of data we have.

Figure 3.1 shows silhouette analysis, the partition of the land, and elbow analysis curve for 8 clusters, respectively.

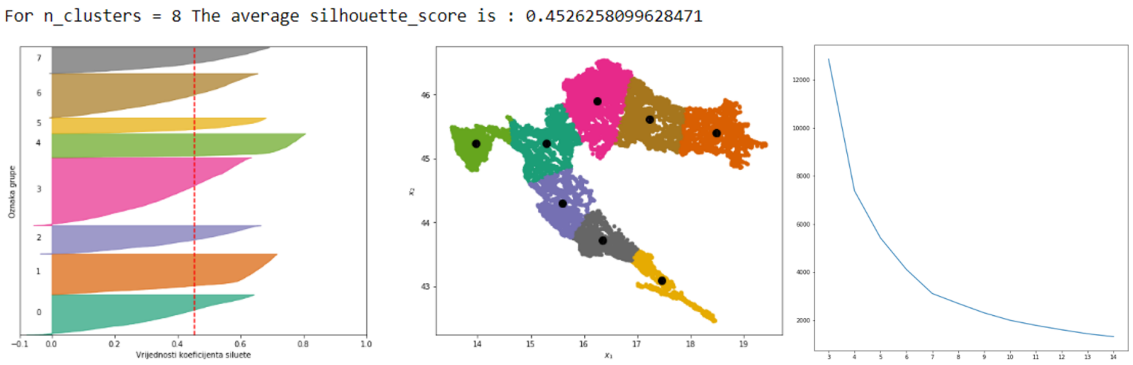

Figure 3.1: The final silhouette.

Silhouette analysis is a graphical method. The method is implemented by graphically displaying the "silhouette value" for each example from set D, and then obviously determining the optimal number of groups based on it. Silhouette values for x are defined as:

$$s(i) = \frac{b(i) - a(i)}{max(a(i), b(i))}$$

Where a(i) and b(i) are the average distances from x(i) to the example of the same or closest group. We calculate and plot the s(i) for all examples of each group.

The value of the silhouette is in the interval [-1, +1]. If s(i) = 1, it means that the example is very far from the example from the neighboring group, and very close to the examples from its group. Examples for which s(i) = 0 are somewhere on the border between two or more groups.

In the Elbow method, we graphically show the dependence of the criterion function on the parameter K and look for the "elbow" of the curve (the place where the function first falls sharply and then stagnates or falls very slowly).

When assessing the K with silhouette analysis, poor groupings are those groupings in which all silhouette values for a group are less than the average silhouette value for the whole set of examples. With K = 8 we can see that is not the case and can be considered a fine grouping. Although it is desirable to have groups of approximately equal size it sometimes does not make much sense to divide a bigger group just so groups could be of the same size. In the elbow method graph we can see the criterion function falls somewhat sharply to K = 7, and then more stagnantly.

Keeping in mind the results of these methods, we decided on the K mostly with the manual quality control of the groups. We picked the division which made sense, referring to the knowledge of the problem we are solving.

The final partition of the area with labeled classes is shown in Figure 3.2.

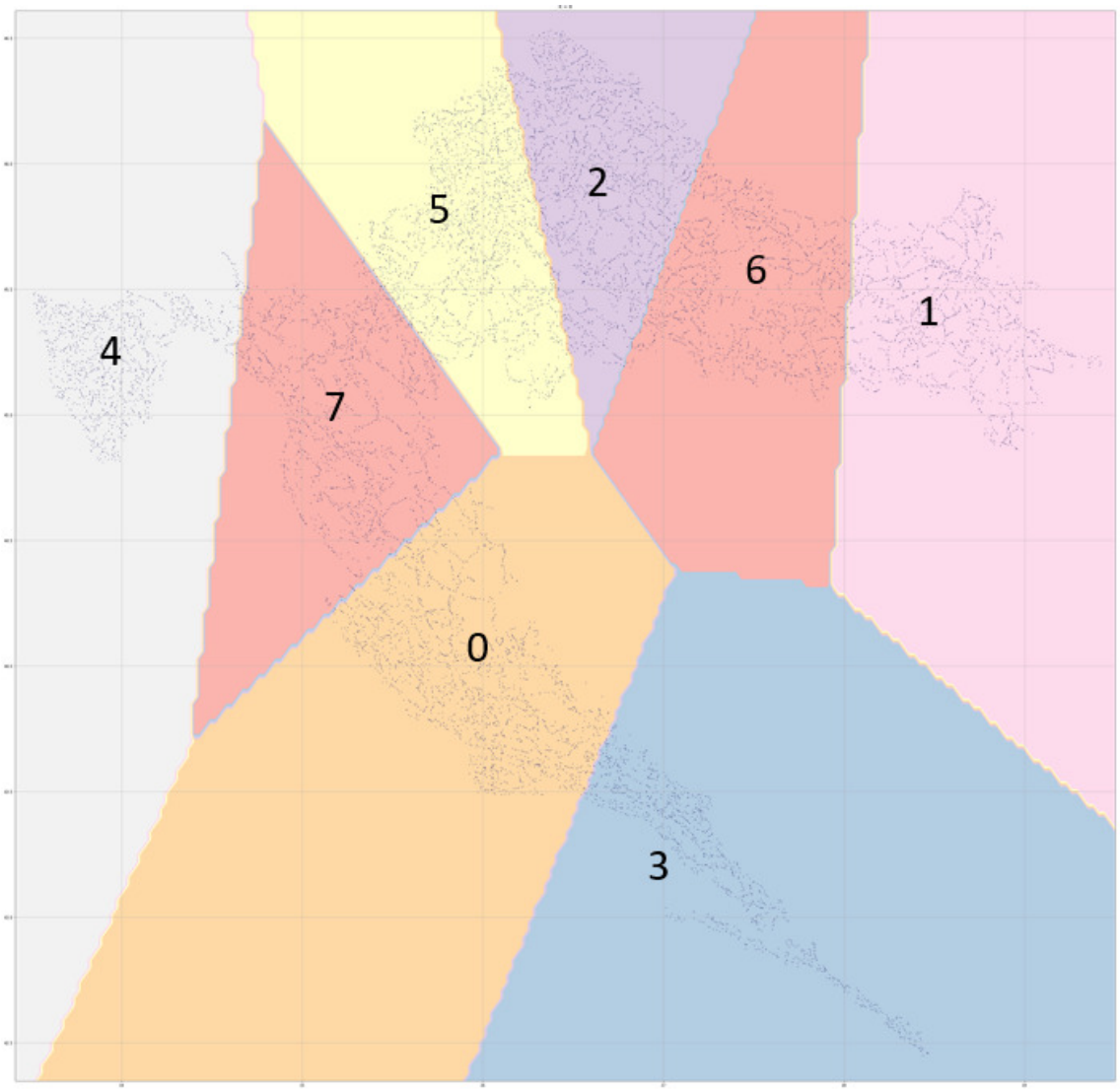

Figure 3.2: Partition of Croatia's mainland into 8 surfaces with labeled classes.

We have created an additional column "label_8" and tagged each row (coordinate) in the dataset with the assigned class. A preview of the dataset file with the new column added can be seen in Figure 3.3.

	uuid	latitude	longitude	label_8
0	69387a76-b6f6-4a76-9d82-59367e14cb12	45.552228	18.538397	1
1	83fd0354-8781-4325-9139-653ba0ce718f	45.116326	14.821817	7
2	5e2f692d-a2e6-45b1-b18b-3cec90b31b64	45.424986	18.767853	1
3	b3447ea2-8ea2-4c4e-b2a8-8611c8253995	46.154501	17.057093	2
4	93c8620f-2e97-4fe1-999a-c4c423b3d878	43.651482	16.323894	0
...	...	...	...	...
15814	717d023f-52d3-447c-8746-730d7e555fe6	45.424661	13.931598	4
15815	0ad4bcfa-4aa4-4826-88ec-2692a2906132	45.420424	15.348075	7
15816	abd34a3c-30cf-462b-938a-9ef2ac9b9a8b	45.632118	17.207108	6
15817	848a0d8b-286b-4b78-b694-57924eaf52aa	45.065947	14.123027	4
15818	8871c47b-9dae-4f0d-a373-f685cfda28e9	46.200279	17.060700	2

15819 rows × 4 columns

Figure 3.3: Dataset structure with the column "label_8" added.

At this point, we can obtain the distribution of data per class. Figure 3.4 shows a histogram of this distribution.

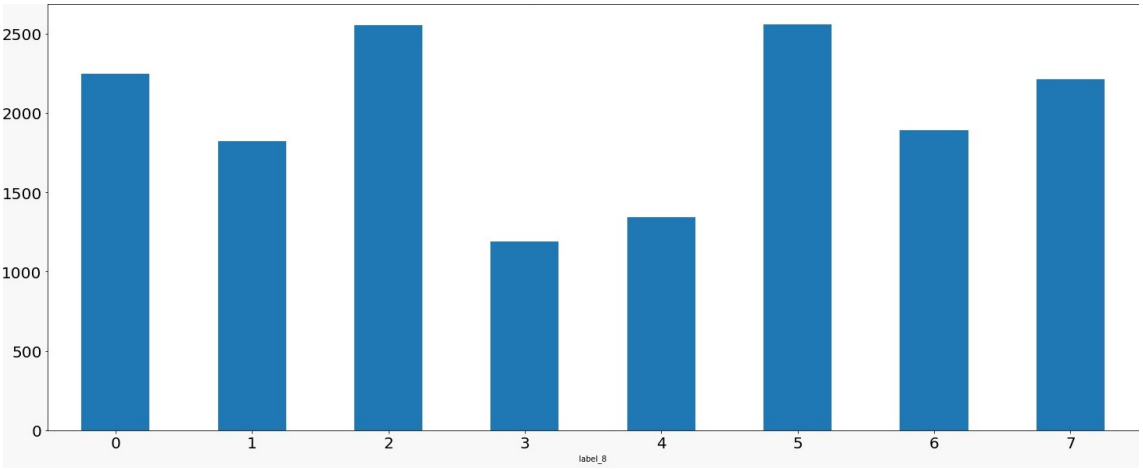

Figure 3.4: Distribution of data by class.

As can be seen in this histogram, some of the classes are assigned less data than others, so we would expect our model to perform slightly better on the more "populated" classes. Again, a thorough analysis was conducted before splitting data in this way, and we believe it was the best possible solution.

The next step was to split data into train, validation, and test sets keeping the

dataset distribution consistent. The ratio we used for the division was 70/15/15, respectively. Figures 3.5, 3.6, 3.7 show distribution of data on train, validation, and test sets, respectively.

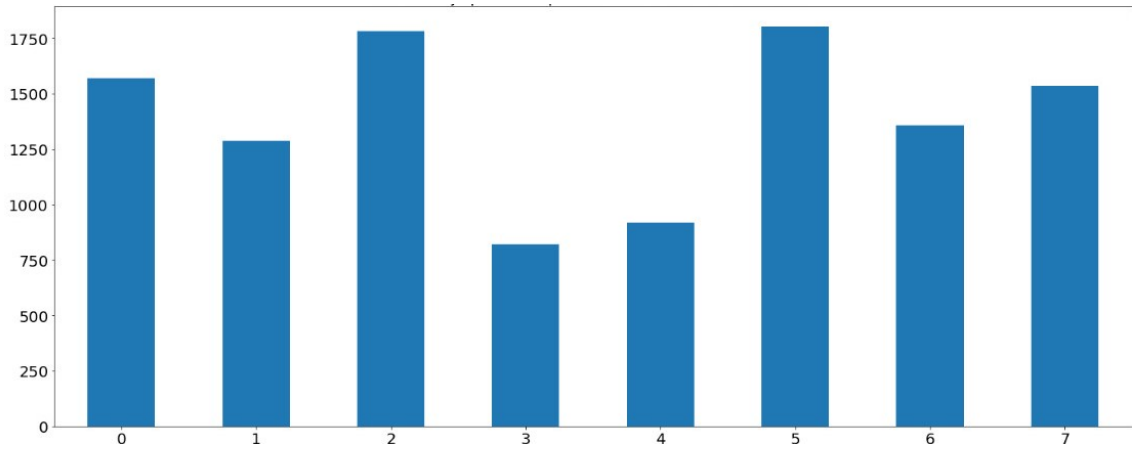

Figure 3.5: Data distribution of train data.

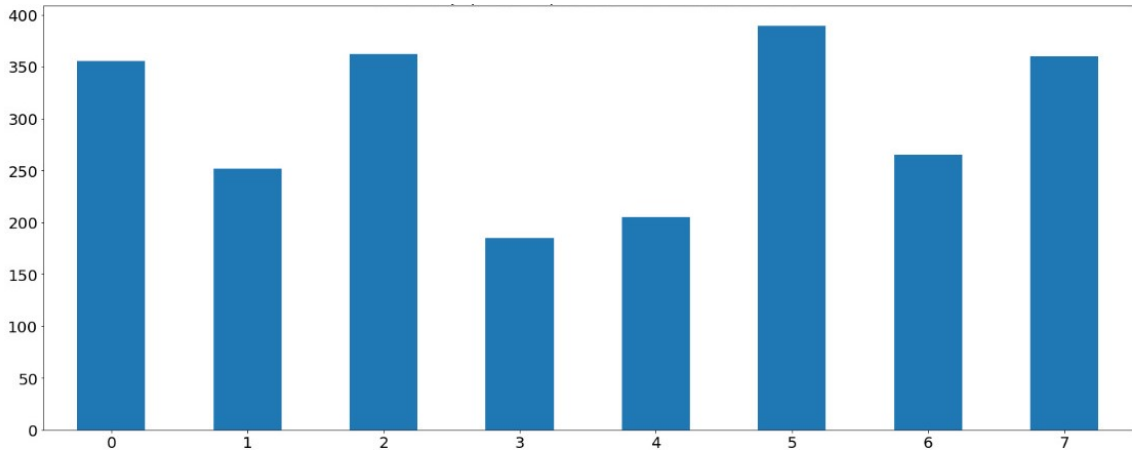

Figure 3.6: Data distribution of validation data.

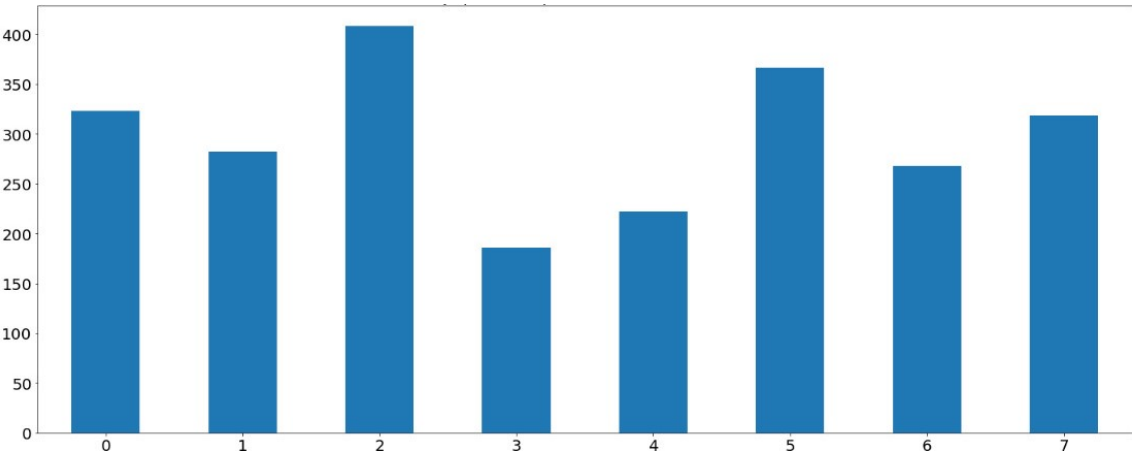

Figure 3.7: Data distribution of test data.

Next, we have repeated the previous process of splitting areas into subclasses for each of the eight classes.

Using K-means algorithm and elbow and silhouette analysis, we considered the optimal division of each area. The final division and the number of subclasses for each class can be seen in Table 3.1.

Table 3.1: Number of subclasses per each class division.

Class ID	Number of subclasses
0	10
1	11
2	14
3	7
4	14
5	15
6	12
7	15

Figures 3.8, 3.9, 3.10, 3.11, 3.12, 3.13, 3.14, 3.15 show partitions into subclasses for each of the 8 classes.

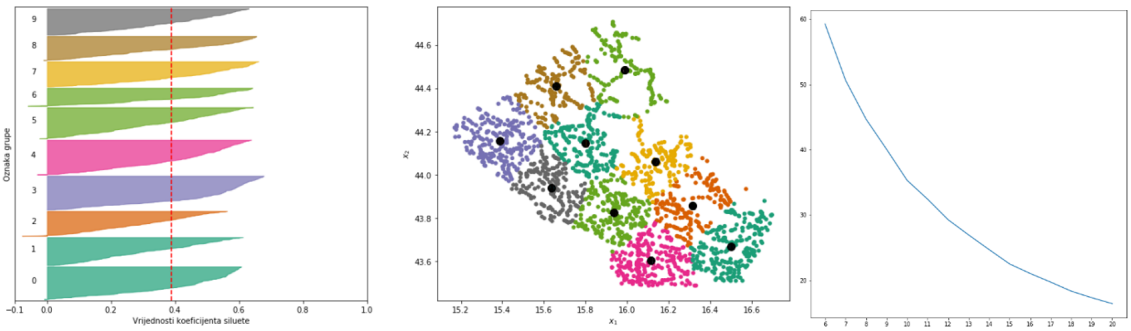

Figure 3.8: Partition of class 0.

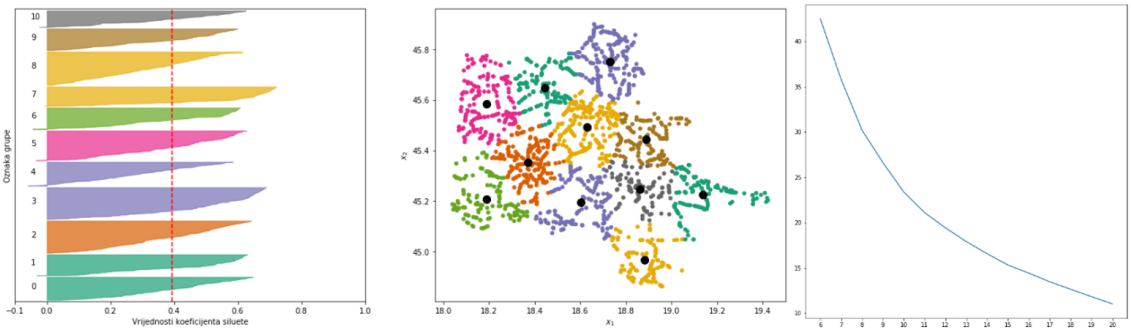

Figure 3.9: Partition of class 1.

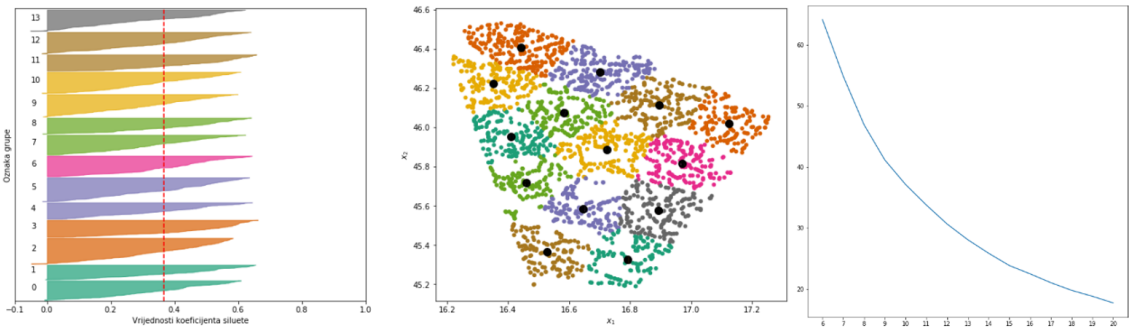

Figure 3.10: Partition of class 2.

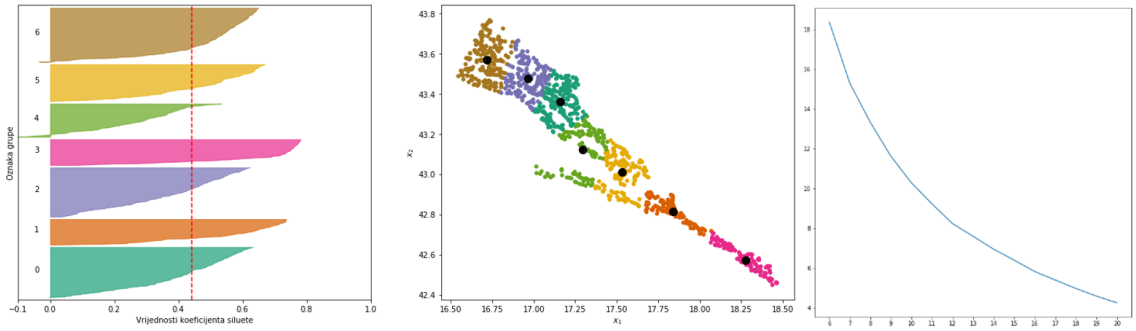

Figure 3.11: Partition of class 3.

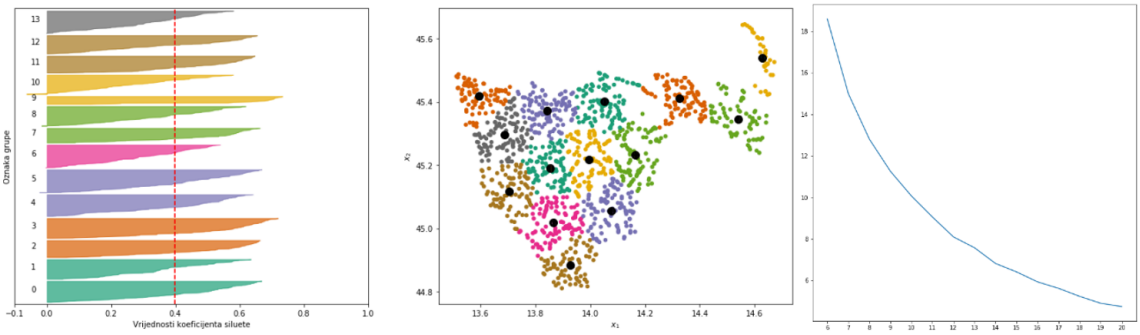

Figure 3.12: Partition of class 4.

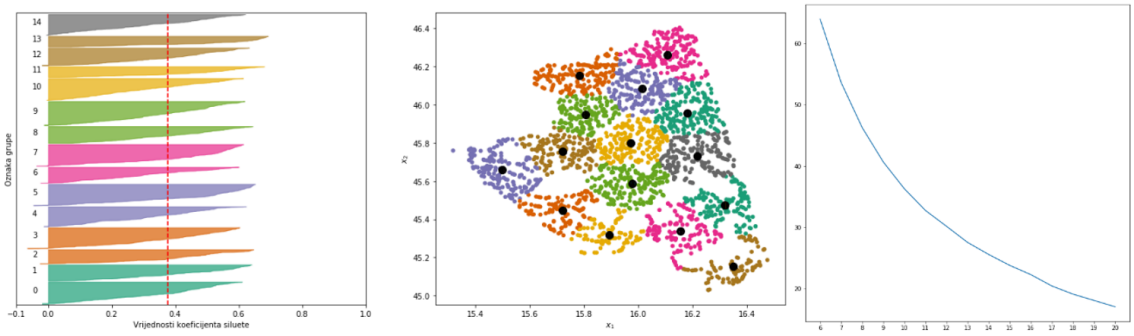

Figure 3.13: Partition of class 5.

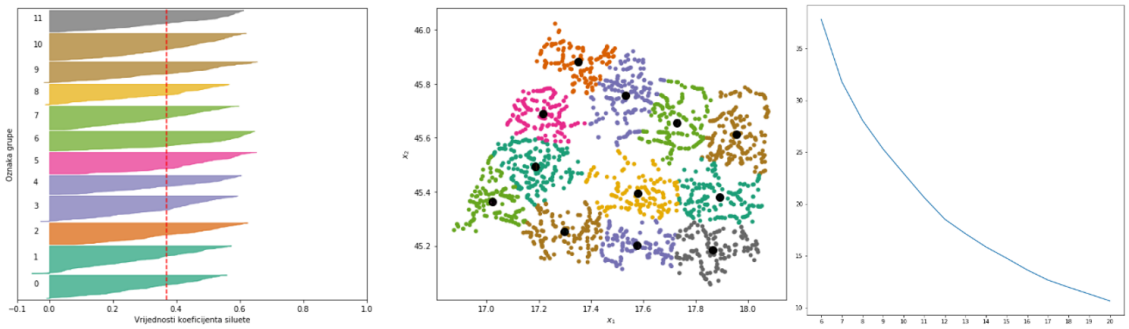

Figure 3.14: Partition of class 6.

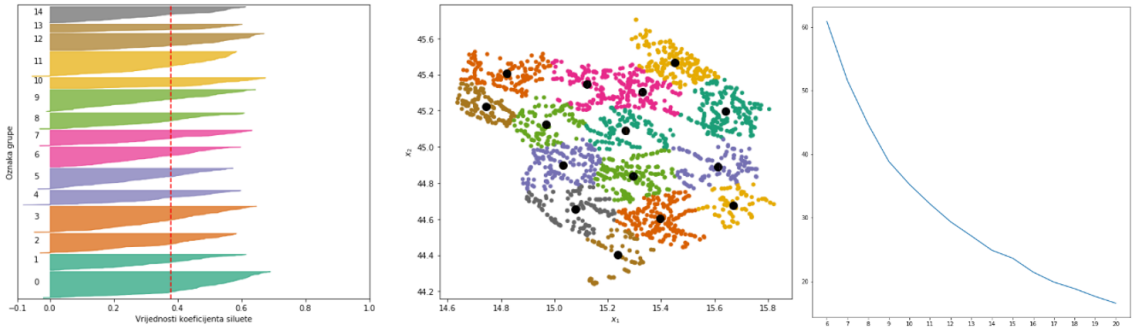

Figure 3.15: Partition of class 7.

The final partition of Croatia's mainland used for training is shown in Figure 3.16 resulting in a total of 98 classes.

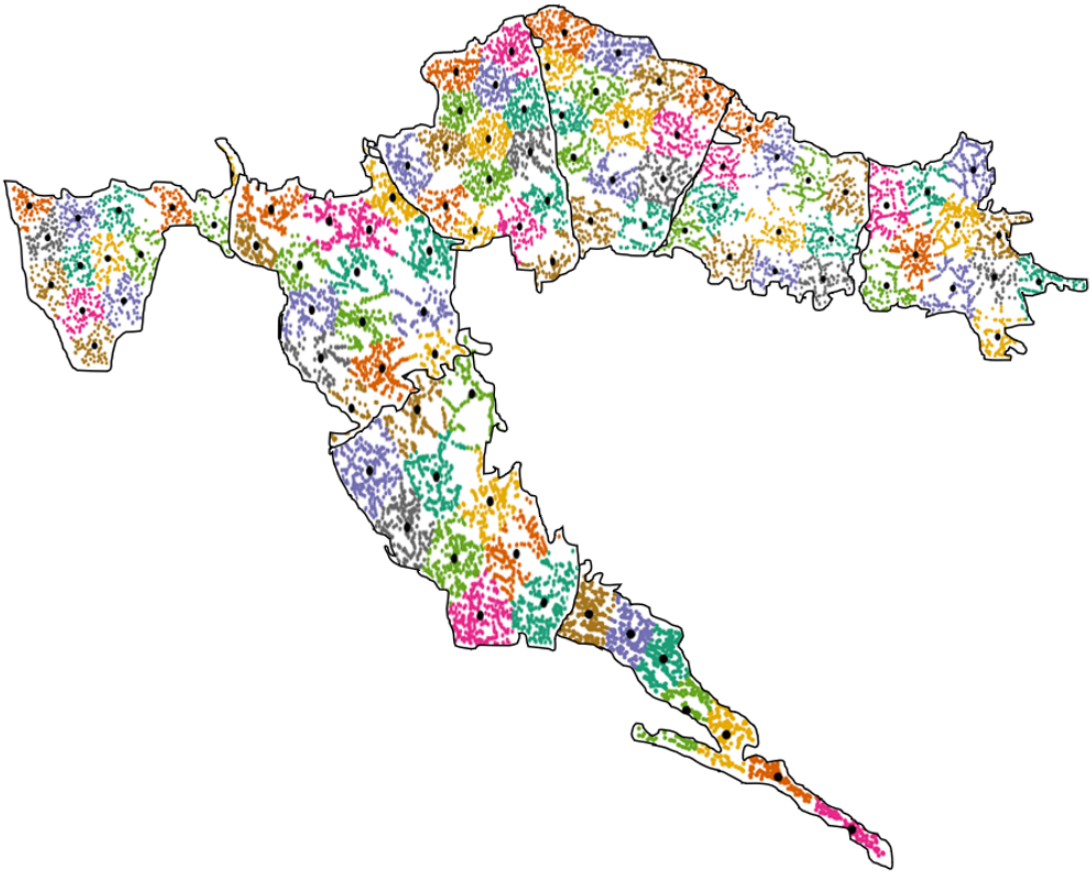

Figure 3.16: Partition into subclasses.

Table 3.2 shows average area and radius measures for the classes.

Table 3.2: Area and radius measures for 98 subclasses.

Area of Croatia	56594 km^2
Average area of a segment	577.49 km^2
Average radius of a segment	13.56 km

After finding the optimal subdivisions, we have assigned a subclass to each row in the dataset based on coordinate and "label_8" class label. A preview of the final dataset file can be seen in Figure 3.17.

	uuid	latitude	longitude	label_8	label_8_subclasses
0	69387a76-b6f6-4a76-9d82-59367e14cb12	45.552228	18.538397	1	10
1	83fd0354-8781-4325-9139-653ba0ce718f	45.116326	14.821817	7	14
2	5e2f692d-a2e6-45b1-b18b-3cec90b31b64	45.424986	18.767853	1	1
3	b3447ea2-8ea2-4c4e-b2a8-8611c8253995	46.154501	17.057093	2	1
4	93c8620f-2e97-4fe1-999a-c4c423b3d878	43.651482	16.323894	0	7
...	...	...	...	...	...
15814	717d023f-52d3-447c-8746-730d7e555fe6	45.424661	13.931598	4	8
15815	0ad4bcfa-4aa4-4826-88ec-2692a2906132	45.420424	15.348075	7	8
15816	abd34a3c-30cf-462b-938a-9ef2ac9b9a8b	45.632118	17.207108	6	8
15817	848a0d8b-286b-4b78-b694-57924eaf52aa	45.065947	14.123027	4	7
15818	8871c47b-9dae-4f0d-a373-f685cfda28e9	46.200279	17.060700	2	1

Figure 3.17: Dataset structure with added "label_8_subclasses".

The distribution of data for each subclass and train, validation, and test set, respectively, can be seen in Figures 3.18, 3.19 and 3.20. These distributions are not kept after splitting, as the sets were already split when dividing into 8 classes previously.

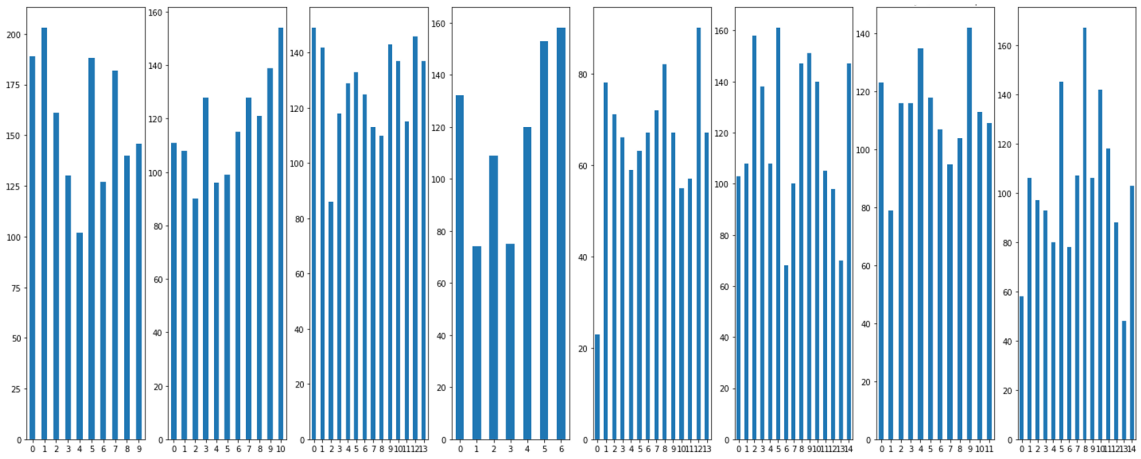

Figure 3.18: Data distribution of subclasses on train data.

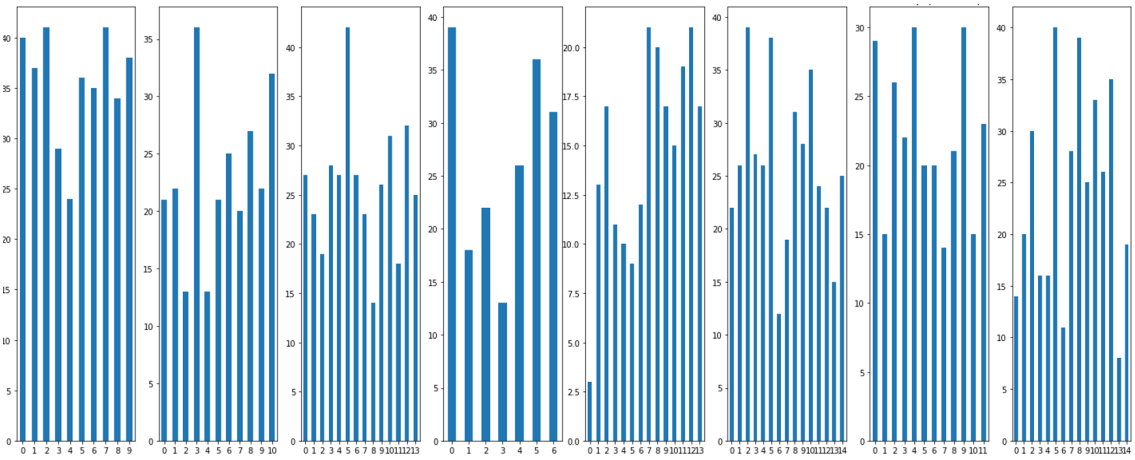

Figure 3.19: Data distribution of subclasses on validation data.

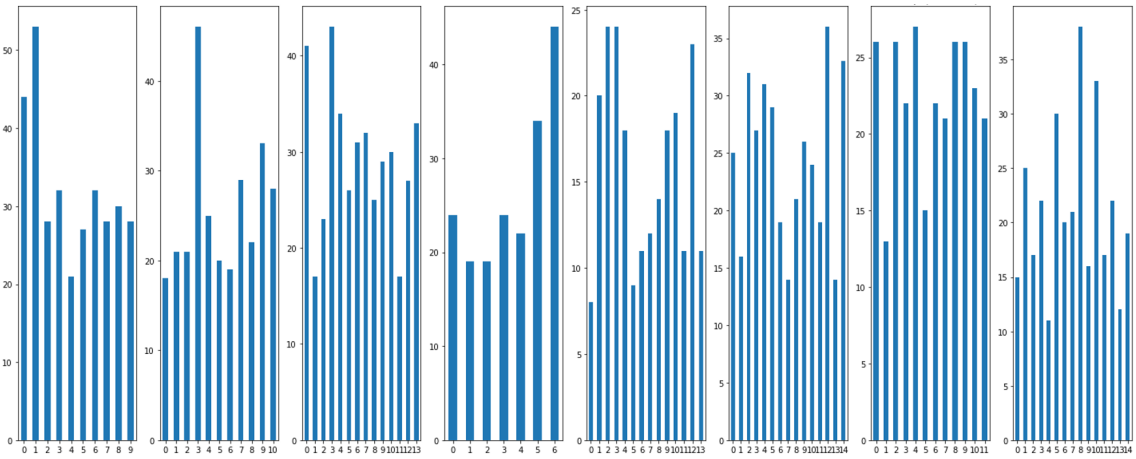

Figure 3.20: Data distribution of subclasses on test data.

We calculated the centroids for both geographical latitude and longitude for each subclass in order to use them later in the model's prediction. They can be seen in Figure 3.21 where columns mark the classes obtained in the first K-means division, and rows correspond to the index of a subclass in each class. Some cells contain `NaN` value, which is present if the class contains fewer subclasses than the index states. So, if the selected image in the first space division belongs to the area with label 2, only the centroids in the column 'Subclass_2' will be considered. If the specific image belongs to a sub-area with label 12, the result of the prediction will be the centroid values [45.55291321, 16.5286481] for latitude and longitude respectively.

	Subclass_label	Subclass_0	Subclass_1	Subclass_2	Subclass_3	Subclass_4	Subclass_5	Subclass_6	Subclass_7
0	0	[44.15612668 15.38626806]	[45.65119832 18.4459707]	[45.83072026 16.44796636]	[43.48499821 16.94250338]	[45.56076109 14.62608621]	[45.4751124 16.31978747]	[45.75703435 17.53163149]	[44.67800611 15.67133216]
1	1	[43.60823548 16.12501561]	[45.43284605 18.89062567]	[46.10343241 16.91463085]	[42.81289171 17.84109742]	[45.08656415 13.9071937]	[46.14193168 15.74552805]	[45.26136894 17.22981669]	[45.1789077 15.30902074]
2	2	[44.14304393 15.80042127]	[44.96254418 18.88888925]	[45.3186414 16.78898616]	[43.13954581 17.28419166]	[45.29452802 13.68326539]	[45.58813459 15.9772626]	[45.18978555 17.79138103]	[45.13150082 14.97876168]
3	3	[43.85890984 16.32333588]	[45.75347298 18.73127773]	[46.39991662 16.35920766]	[42.57352839 18.27785276]	[45.411221 14.32799458]	[46.26428119 16.1168249]	[45.6599875 17.72793901]	[44.65549834 15.11254367]
4	4	[44.48330164 15.98685025]	[45.20280696 18.18434278]	[46.25331559 16.74677179]	[43.00959494 17.52244383]	[45.2419928 14.00312094]	[45.75664682 15.72285586]	[45.4958832 17.19417682]	[44.84964116 14.96217599]
5	5	[43.82243087 15.93687392]	[45.21720702 18.48814182]	[45.7896163 16.97667141]	[43.57284552 16.71276151]	[45.35297877 14.55026954]	[45.97886819 16.16525166]	[45.3364006 17.94099305]	[45.20046876 15.64159186]
6	6	[44.40811252 15.65735964]	[45.22669828 19.1181571]	[46.12779041 16.35191771]	[43.37456969 17.14294363]	[45.20584447 13.82629043]	[45.31788884 15.89499209]	[45.88162818 17.35063959]	[44.89017786 15.6248029]
7	7	[43.66960238 16.5062493]	[45.59303223 18.18020095]	[45.60910982 16.58436696]	NaN	[45.05770085 14.08712566]	[45.44682501 15.72101967]	[45.3678177 17.01667864]	[45.34880282 15.174607]
8	8	[44.05540448 16.14046899]	[45.22006848 18.75768866]	[46.01206169 17.12839771]	NaN	[45.37320042 13.84117008]	[45.74062899 16.22380098]	[45.68796621 17.21628755]	[45.4481106 15.43285149]
9	9	[43.93874096 15.63609271]	[45.39402579 18.34495874]	[46.04846339 16.58219853]	NaN	[45.40400018 14.05170461]	[45.80275332 15.98141956]	[45.606692 17.94955831]	[44.94254186 15.15120669]
10	10	NaN	[45.50059851 18.63014198]	[46.33351276 16.54312601]	NaN	[45.09710142 13.70642161]	[45.65828588 15.4989668]	[45.39784696 17.629158]	[44.60840166 15.3952331]
11	11	NaN	NaN	[45.36477935 16.5286481]	NaN	[45.41751946 13.5939549]	[45.3381485 16.15609441]	[45.23058429 17.47443211]	[45.40921235 14.83406581]
12	12	NaN	NaN	[45.55291321 16.85840129]	NaN	[44.91558131 13.90663581]	[45.94453137 15.80844477]	NaN	[44.8606703 15.34540437]
13	13	NaN	NaN	[45.88635438 16.75183918]	NaN	[45.23842404 14.16751597]	[45.15463765 16.34999926]	NaN	[44.4059999 15.240095]
14	14	NaN	NaN	NaN	NaN	NaN	[46.12826654 15.96604096]	NaN	[45.22897137 14.7403904]

Figure 3.21: Structure of the prediction table.

3.1. Data augmentation

To seemingly augment our data set, we used horizontal rotation. The images were flipped horizontally while training with a probability of 0.5. None of the other augmentation techniques were used because we concluded that all of the images would be taken from a similar perspective.

All of the images included in the model were resized to dimensions (256, 256).

4. Modelling

This chapter describes the models we tested for this task and the final model we decided to use in more detail.

4.1. Loss functions

This section briefly describes the loss functions used in evaluating the models' performances.

4.1.1. Haversine

To compute the distance between two locations on Earth, the haversine formula is used and the same measure is used as a loss function to evaluate the final model's performance. Haversine formula calculates the great-circle distance between two points; that is, the shortest distance over the earth's surface. The final haversine distance (d) is calculated using the following formulas:

$$a = sin^2(\Delta\phi/2) + cos\phi 1 \cdot cos\phi 2 \cdot sin^2(\Delta\lambda/2)$$

$$c = 2 \cdot atan2(\sqrt{a}, \sqrt{1-a})$$

$$d = R \cdot c$$

Here ϕ is the latitude, λ is the longitude, and R is the Earth's radius (approximately 6,371km). The angles are calculated in radians.

4.1.2. Cross-Entropy Loss

Cross-entropy loss is a metric used to measure how well a classification model in machine learning performs. The loss (or error) is measured as a number between 0 and 1, with 0 being a perfect model. The goal is generally to get your model as close to 0 as possible. Cross-entropy loss measures the difference between the discovered probability

distribution of a machine learning classification model and the predicted distribution. We calculate a separate loss for each class label per observation and sum the result.

$$-\sum_{c=1}^{M} y_{o,c} \log(p_{o,c})$$

Here, M is a number of classes, y is a binary indicator (0 or 1) if class label c is the correct classification for observation o, and p is the predicted probability observation o is of class c.

4.1.3. Custom Cross-Entropy Loss

Our custom loss additionally punishes misclassifications. It works by multiplying the calculated cross-entropy loss by an additional constant which grows with the distance of centroids of individual classes in the first layer model. Distances were calculated using the haversine loss function. The results are shown in Figure 4.1.

	Distance between cluster centers	Class 0	Class 1	Class 2	Class 3	Class 4	Class 5	Class 6	Class 7
0	Class 0	0.000000	262.059078	225.301739	135.844382	209.361655	195.305552	208.595817	129.533506
1	Class 1	262.059078	0.000000	162.973707	264.456241	360.452763	208.792489	87.121952	265.191036
2	Class 2	225.301739	162.973707	0.000000	307.056662	221.001014	58.385632	81.564500	148.885859
3	Class 3	135.844382	264.456241	307.056662	0.000000	344.914749	298.646500	254.722974	258.139367
4	Class 4	209.361655	360.452763	221.001014	344.914749	0.000000	162.761283	274.794121	101.617046
5	Class 5	195.305552	208.792489	58.385632	298.646500	162.761283	0.000000	121.903248	95.740480
6	Class 6	208.595817	87.121952	81.564500	254.722974	274.794121	121.903248	0.000000	182.862158
7	Class 7	129.533506	265.191036	148.885859	258.139367	101.617046	95.740480	182.862158	0.000000

Figure 4.1: Distances between centroids.

As seen in Figure 4.1, the lowest distance is the one between classes 2 and 5 (58,385632 km) and the largest distance is between class 1 and 4 (360,452763 km). Therefore, the punishment will be larger if the model misclassifies classes 1 and 4 than for the misclassification of classes 2 and 5. The distances are normalized to range [1, 2] and the final table of constants is shown in Figure 4.2.

	0	1	2	3	4	5	6	7
0	1.000000	1.674266	1.552580	1.256429	1.499810	1.453276	1.497274	1.235537
1	1.674266	1.000000	1.346241	1.682201	2.000000	1.497925	1.095132	1.684634
2	1.552580	1.346241	1.000000	1.823231	1.538342	1.000000	1.076734	1.299603
3	1.256429	1.682201	1.823231	1.000000	1.948561	1.795389	1.649979	1.661289
4	1.499810	2.000000	1.538342	1.948561	1.000000	1.345538	1.716425	1.143119
5	1.453276	1.497925	1.000000	1.795389	1.345538	1.000000	1.210276	1.123664
6	1.497274	1.095132	1.076734	1.649979	1.716425	1.210276	1.000000	1.412082
7	1.235537	1.684634	1.299603	1.661289	1.143119	1.123664	1.412082	1.000000

Figure 4.2: Scaled distances between centroids.

4.2. Baseline

As a reference for our future and hopefully better results, firstly, we explored a baseline model described in this section.

This model works by predicting the output for each example in the dataset as a centroid of a random region. Croatia's surface was split into 98 regions as seen in Figure 3.16. Each of the examples from the dataset (4 images) was assigned a random region and the final output is calculated as a centroid pair (latitude, longitude) of the corresponding region.

4.3. Final model

Our main goal is to group areas of Croatia into smaller groups to minimize the distance between the real coordinates and the ones predicted by the model. This is a classification task. Firstly, we will describe the final model architecture we chose for this task, and later, the different approaches we tried will be described briefly.

4.3.1. Model architecture

This section describes the final model chosen for this task. The model is fairly complex, but the structure is repetitive.

If you observe the structure of our model in Figure 4.3, you will realize it resembles a tree-like model. When these models are mentioned in data science, the first thing that comes to mind are decision trees and random forests. Our model has a decision tree structure, which means a decision made earlier affects all later decisions and thus

reduces search space. The circles in Figure 4.3 will be described in detail later in the text.

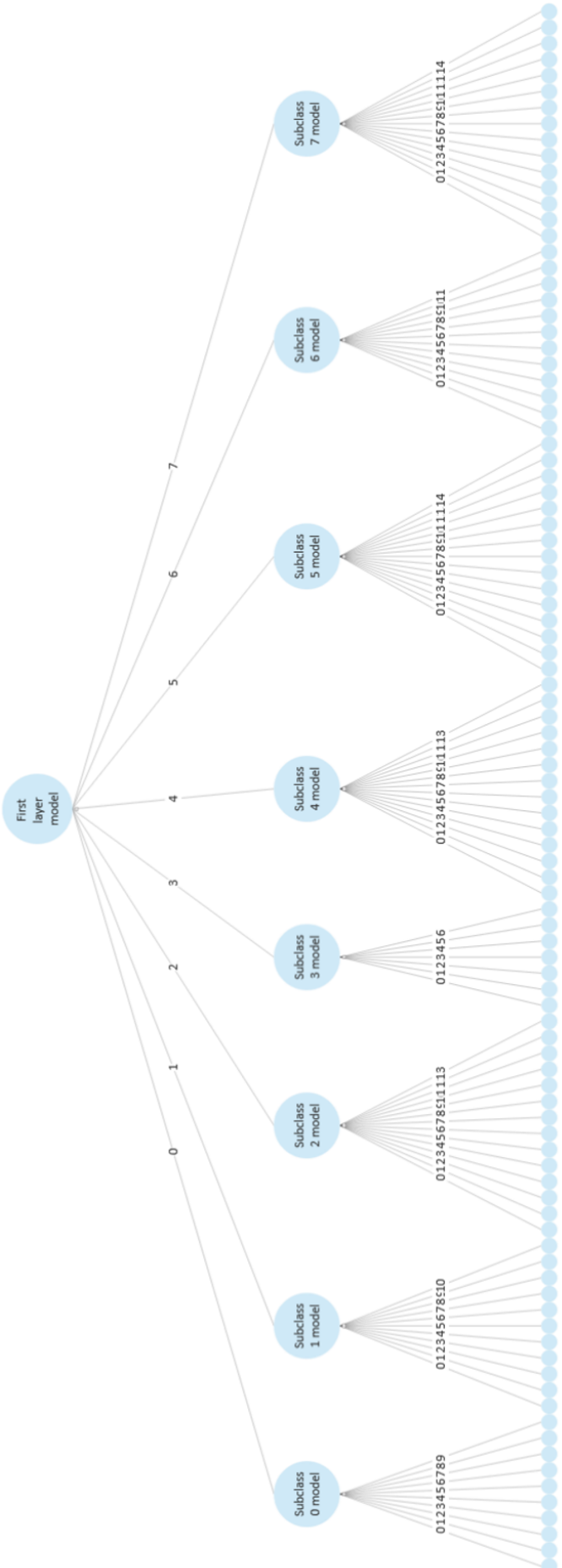

Figure 4.3: High-level model structure.

Briefly, an image arrives at the input of the first part of the model (First layer model) and the model classifies it into one of the eight regions described in the previous chapter. After that, the same image is forwarded to the corresponding Second layer model where it is classified into a smaller sub-area of Croatia's surface.

All 9 models have the same architecture. The only difference is in the last layer of the network, where the model's output size depends on the number of classes for each area. The first part of each model is a ResNet50 network to which 3 fully connected layers with ReLu nonlinear activation functions and dropout are connected. The final layer uses the Softmax activation function. The Softmax activation function is interpreted to give the probability of occurrence of each class. Thus, the index of the largest number in the output layer represents the most probable class. This part of the network will be referred to as MLP (Multi-Layer Perceptron).

ResNet50 is a 50 layers deep convolutional neural network. ResNet, short for Residual Networks, is a neural network used in many computer vision tasks. Deep convolutional neural networks are good for extracting image features. Stacking too many layers arises the problem of vanishing/exploding gradients, and when tackling this problem, another one arises. When neural networks start to converge, accuracy could get saturated and then degrade rapidly. This is hard to solve by adding more layers, and ResNet addresses this problem by introducing shortcut connections that simply perform identity mappings. By doing this, no additional parameters were added to the model and computational time was kept in check.[1]

Also, when initializing our model, the ResNet50 part serves to extract the features from the image, and the multi-layer perceptron (MLP) part connects them to the expected output/class.

During model initialization, the ResNet50's weights are set to the pre-trained ImageNet weights, and for the rest of the network, the parameters are initialized randomly.

Each of the circles with the label "model" in Figure 4.3 are constructed as shown in Figure 4.4.

[1] `https://blog.devgenius.io/resnet50-6b42934db431,https://iq.opengenus.org/resnet50-architecture/`

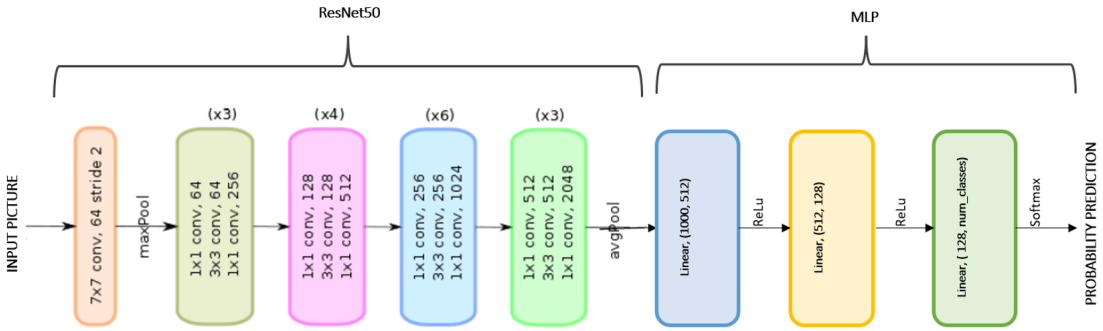

Figure 4.4: Model architecture.

4.4. First layer

Even though the expected output values for this task are continuous, the main reason we decided to use classification instead of regression is because of the natural partition of Croatia's regions. Croatia is very biodiverse; landscapes and natural features differ greatly across different regions. We expect images from the same regions to have a lot of similar features. Because of this, we agreed that the first part of our model would have to include classification. The rest of the model also happens to be a classifier, but that is chosen after trying out different approaches.

We tested two models for performing this task. The first one is ResNet50 + MLP described earlier, and the second one is ViT (Vision transformer). The vision transformer model we used splits an image into fixed-size patches, linearly embeds each of them, and adds position embeddings, and then feeds the resulting sequence of vectors to a standard Transformer encoder. In order to perform classification, an extra learnable "classification token" is added to the sequence.[2]

The ResNet50+MLP model is firsz trained on 75 epochs with cross-entropy loss. After that, the model was trained on 25 additional epochs with a lower learning rate and we used our custom Cross-Entropy loss function. This way, the most erroneous predictions are punished additionally.

The ViT model was trained on 75 epochs with cross-entropy loss.

For training and evaluation of all of the models we tested, Adam optimizer was used. Also, batch size was always set to 24 (images). The hyperparameters for both models are shown in Table 4.1 for better visualization.

[2] `https://github.com/google-research/vision_transformer`

Table 4.1: Hyperparameters for the models.

model	Learning rate	Regularization	Epochs number	Loss function
`ResNet50+MLP`	0.0003 + 0.000001	L2 = 0.000001	75 + 25	Cross Entropy Loss + Custom Loss Function
`ViT`	0.0003	L2 = 0	75	Cross Entropy Loss

4.4.1. Group decisions

Instead of training the model on a set of all four images, we decided to train it on **individual** images. Therefore, each of the four images that are expected as the input is individually passed through the model and the final class is obtained based on a group decision which will be described shortly.

By testing different approaches for these group decisions, it was concluded that the decision to select a class will be made by summing all of the model's output values with the corresponding outputs from all four images, and the index of the class with the highest group probability will be declared the winner.

The other approach we tested was to firstly find an index of the most probable class for each image and then set the most frequently emerging class as the overall winner. The problems with this approach were frequent appearances of more common classes and loss of model reliability. i.e. The first image gave 70% certainty for class 1 and 5% for class 2. The model produces an output for the other 3 images with less certainty where the probabilities are smaller for each class. Second, third and fourth image output 5% for class 1 and 20% for class 2. This way votes by image would be 1 for class 1 and 3 for class 2, but when we sum the probabilities, the highest number would be 85 for class 1, while class 2 would have 65. That is why we opted for the summation of class probabilities.

4.5. Second layer

As for the second layer of our model, that is, after the part with the classification into 8 regions, different approaches were tested. As mentioned, we used the Adam optimizer and a batch size of 24 (images) for all of the models' training.

The first model to test was a regression model. A ResNet50 network was connected to a linear regression model with 1000 input values and 512 output values with the ReLu activation function. This output was then connected to another linear regression model with 512 inputs, 2 outputs (for the expected latitude and longitude), and a

sigmoid activation function. As the sigmoid function produces values ranging from 0 to 1 - (0, 1), the expected output values were scaled to this span and later upscaled for evaluation. This model was trained on data already separated into subclasses, therefore each class had a separate model but with the same architecture. All of the models were trained with the same hyperparameters. Each region's model was trained in 100 epochs. The hyperparameters for this model are shown in Table 4.2.

Table 4.2: Hyperparameters for the second layer linear models.

model	Layers	Learning rate	Epochs	Loss function
`Second layer model (ResNet50+LR)`	50 + 1 + 1	0.0001	100	Haversine

The second approach that was tested was a classification model. This model's architecture is the same as from the ResNet50+MLP classification model described earlier, the only difference being in the number of subclasses for each of the previously split classes. The loss function used for evaluation while training is Cross-Entropy loss.

Hyperparameters used in the training of these 8 models are shown in Table 4.3.

Table 4.3: Hyperparameters for the second layer classification models.

Subclass ID	LR	Epochs	Regularization
`0`	0.0003	50	L2 = 0.000001
`1`	0.0003	60	L2 = 0
`2`	0.0003	40	L2 = 0
`3`	0.0003 + 0.000001	30 + 10	L2 = 0.000001
`4`	0.0003	45	L2 = 0
`5`	0.0003 + 0.000001	60 + 20	L2 = 0.000001
`6`	0.0003 + 0.000001	50	L2 = 0.000001
`7`	0.0003	35	L2 = 0

5. Evaluation

In this chapter we will firstly show the performance of each of the individual layers described in the previous chapter and for different approaches (models) that were tested.

5.1. First layer evaluation

The accuracy results for the previously described First layer models and different decision modes can be seen in Table 5.1.

Table 5.1: Accuracy for different classification models in the first layer and different decision modes.

model	Decision	Train acc	Val acc	Test acc
`First layer model - ResNet50+MLP`	group	96.44%	75.52%	75.85%
	separate	87.54%	67.78%	66.98%
`First layer model - ViT`	group	93.01%	62.79%	62.62%
	separate	81.80%	50.19%	51.90%

After evaluating the results, it was decided that the ResNet50+MLP model with group decision mode will be used for the first part of our classification model. The confusion matrix for the chosen First layer model on the test set is shown in Figure 5.1. The model struggles the most with the correct identification of classes 3 and 4. They represent the Istria and Dubrovnik regions, which are located on the outskirts of Croatia. Also, if we recall the initial distribution of data by classes, the fewest examples existed in these two classes (Figure 3.4). By further training of the model with a custom cross-entropy function, we reduced the likelihood of such confusions, but not entirely because the predictions of other classes began to deteriorate significantly. We will see later in section 5.4 how rare this phenomenon of significantly erroneous misclassifications is, but it significantly affects the mean distance error.

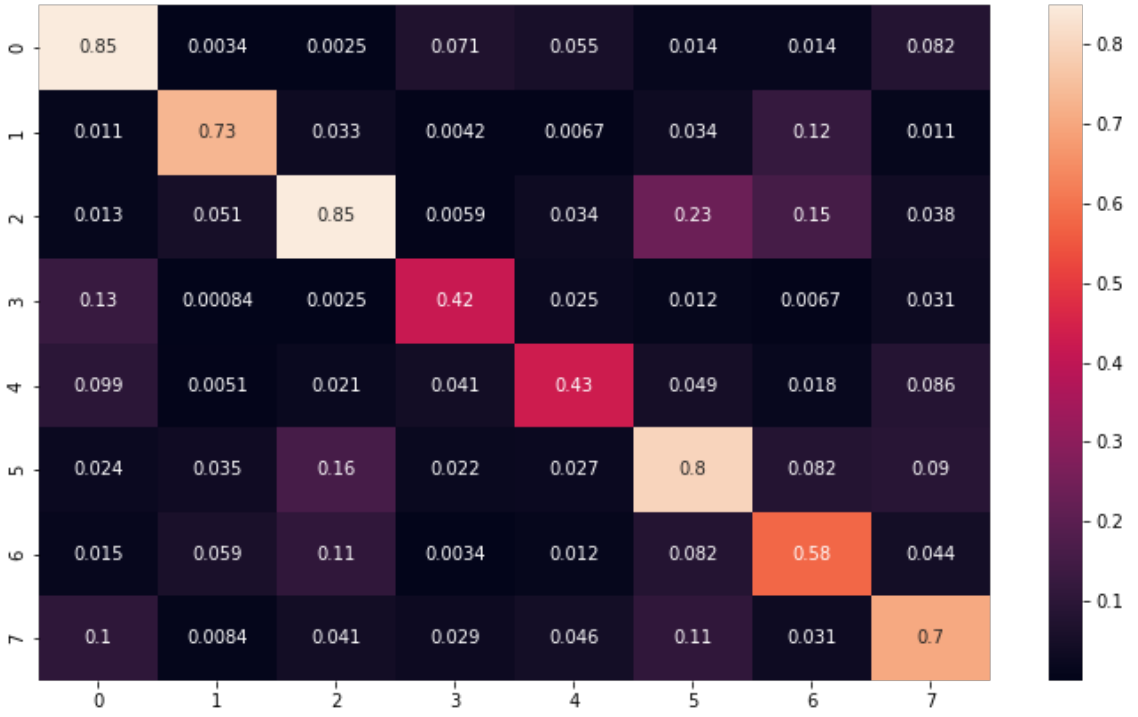

Figure 5.1: Confusion matrix for the chosen First layer model.

5.2. Second layer evaluation

We will firstly consider the performance of the regression models. To evaluate results for the Second layer regression models, we used the haversine loss function. We should keep in mind that the haversine loss was computed on each of the classes when the dataset was already split into 8 classes. So, the loss would be higher and the results would be worse if the loss function was computed after the whole process of first classifying the images and then running the regression model to predict the latitude and the longitude. The results for the validation set can be seen in Table 5.2.

Table 5.2: Haversine loss on each region's validation dataset for the regression models.

Class ID	Haversine loss [km]
0	41.53
1	34.39
2	38.32
3	51.98
4	28.97
5	36.70
6	35.46
7	37.59
Avg:	38.12

To evaluate the results from the Second layer Classification models, we calculated the accuracy (true-positive / (true-positive + false-negative)) for each of the model's predictions. These results were also computed after the initial classification. The results are shown in Table 5.3

Table 5.3: Accuracy on each region's dataset for the classification models.

Class ID	Train acc [%]	Val acc [%]	Test acc [%]
0	91.07	65.63	62.23
1	90.22	60.32	60.28
2	68.42	48.90	42.89
3	99.76	62.70	68.82
4	97.49	49.27	39.64
5	62.04	34.96	32.24
6	100.00	65.66	64.18
7	75.91	46.11	49.06

5.3. Final model evaluation

When evaluating our model, we tried different prediction approaches. At first, the resulting prediction was a centroid of the most probable class. Later, since the metrics of the second layer model were relatively poor, we decided to try to interpolate the k most probable classes by weight average. The weight in question is the probability of the occurrence of a particular class, and the average value is computed for the classes

taken into consideration as a possible solution. The formula for the weighted average is:

$$\overline{x} = \frac{\sum_i w_i x_i}{\sum_i w_i}$$

Where w_i is the weight, and x_i is the centroid.

Table 5.4 shows the final model's results.

Table 5.4: Final model results.

Final model loss	Train	Val	Test	Test WA2[1]	Test WA3[2]	Test WA4[3]
`mean`	18.29	46.88	47.16	46.33	46.17	46.20
`std`	28.67	63.97	63.83	62.64	62.37	62.16
`min`	0.08	0.35	0.48	0.08	0.14	0.27
`25%`	6.80	9.76	9.76	10.62	10.78	11.12
`50%`	10.22	19.08	19.98	21.46	21.99	21.99
`75%`	15.07	56.57	57.25	52.32	51.54	52.01
`max`	404.25	490.47	480.74	442.02	440.11	437.63

From the Table 5.4 we can see that the mean error on the test and validation set is between 46 and 47 km. Also, the introduction of weighted averages into the prediction has led to a further reduction in predicted average distances.

We can also see that the optimal number of classes to consider when interpolating the centroids is 3. Therefore, the 3 most probable classes interpolation is used in the final model for this competition.

Table 5.5 shows average losses on all of the models we tested, for comparison.

Table 5.5: Average loss in km for different models.

Model	Train	Val	Test
`Baseline`	180.71	179.50	178.80
`ViT+Classification`	20,40	61,71	61,92
`ResNet50+MLP+Regression`	39,67	58,84	59,49
`ResNet50+MLP+Classification`	18,29	46,88	46.17

5.4. Interpreting predicted values

Figure 5.2 shows points of the map with the lowest value for the loss function. All of these predictions produced a haversine loss lower than 10km.

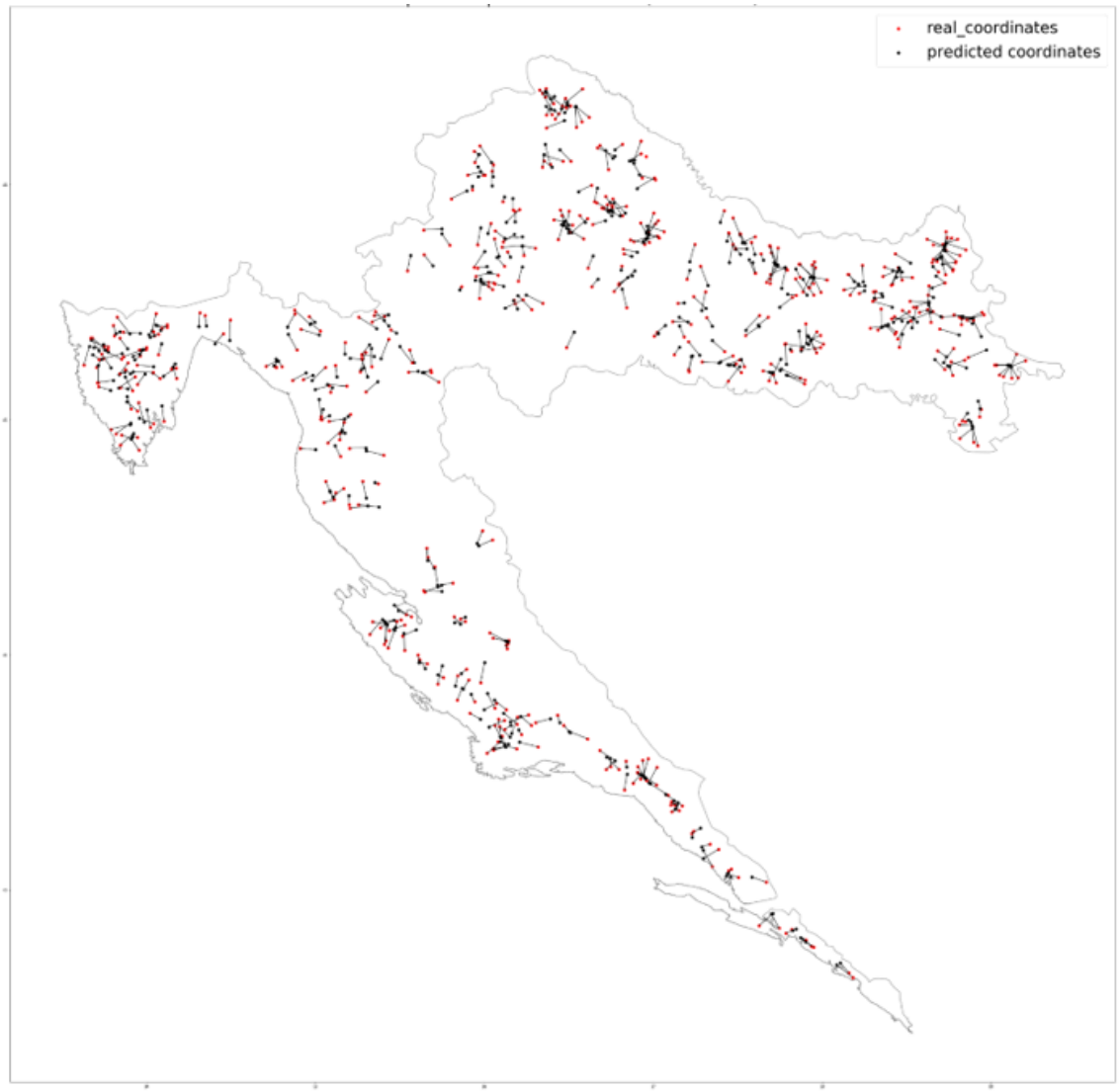

Figure 5.2: Best predicted points on the map (distance < 10km).

When we compare the plotted predictions on Figure 5.2 to the plotted points from the complete dataset on Figure 2.10, we can see that the areas where data is more dense produce more results with low loss compared to the more sparse areas. This is something that was expected. The percentage of such, under 10 km loss, well-predicted coordinates on the test set is 22.67%. Some of the image sets that produced the best results on our split test data can be observed in Figures 5.3, 5.4, 5.5 and 5.6.

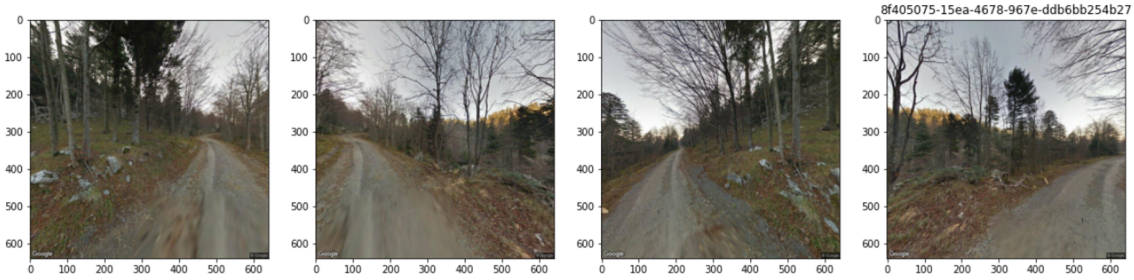

Figure 5.3: 1. set of images that produced the lowest error.

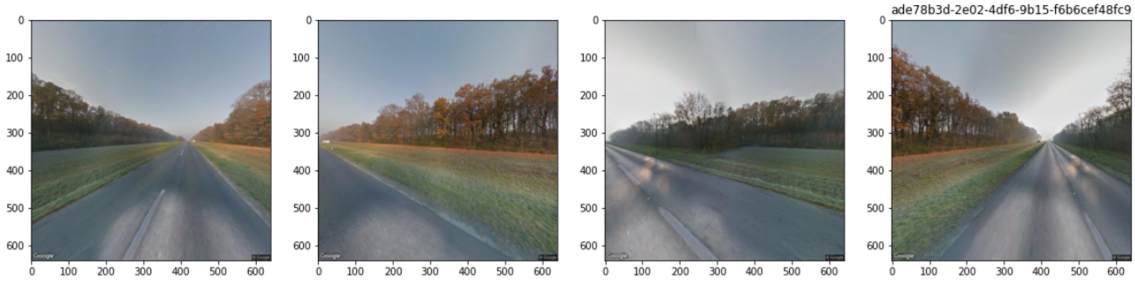

Figure 5.4: 2. set of images that produced the lowest error.

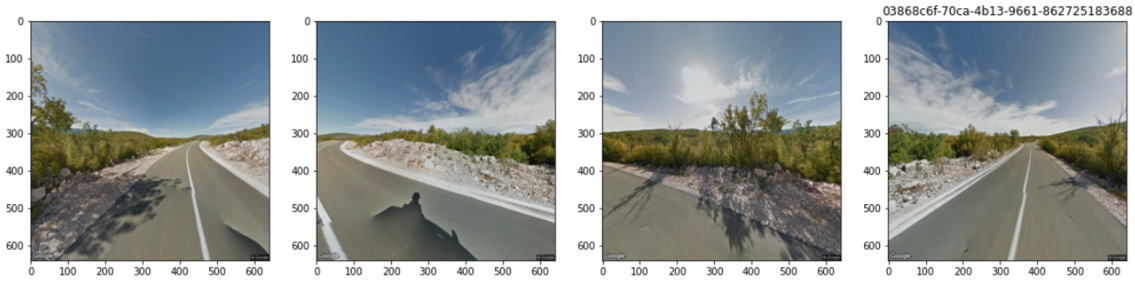

Figure 5.5: 3. set of images that produced the lowest error.

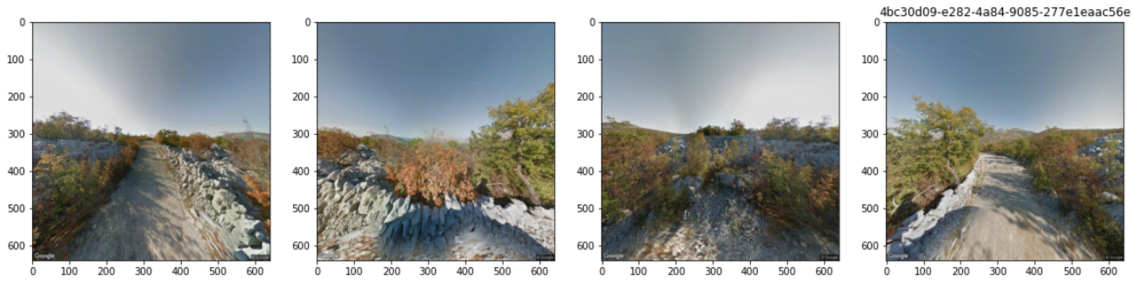

Figure 5.6: 4. set of images that produced the lowest error.

We were curious to see our model's worst predictions, and an interesting observation can be made if we take a look at Figure 5.7.

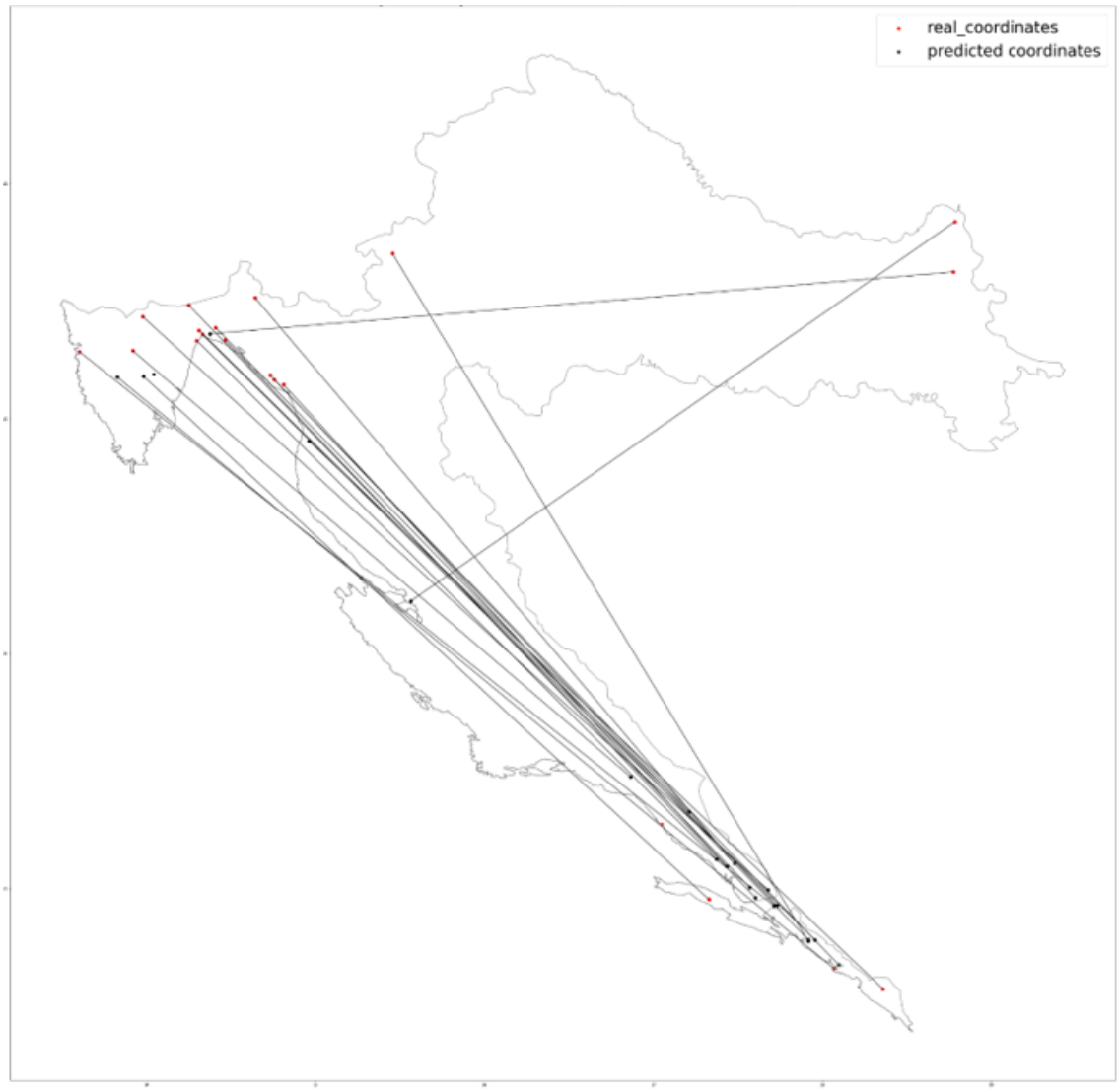

Figure 5.7: Worst predicted points on the map (distance > 300km).

Most of the predicted-real coordinate pairs are located somewhere on the coast, and we expected these sets of images to show the sea, beaches or something characteristic for those landscapes. But none of that happened to be like that. One of the samples (Figure 5.8) is an interior space that was left in the dataset to ensure some noise. But we couldn't interpret why the rest of the image sets in the collection of the worst predicted produced such high loss. The percentage of these erroneous guesses on the test dataset is 0.842%. Some of them can be seen in Figures 5.9, 5.10 and 5.11.

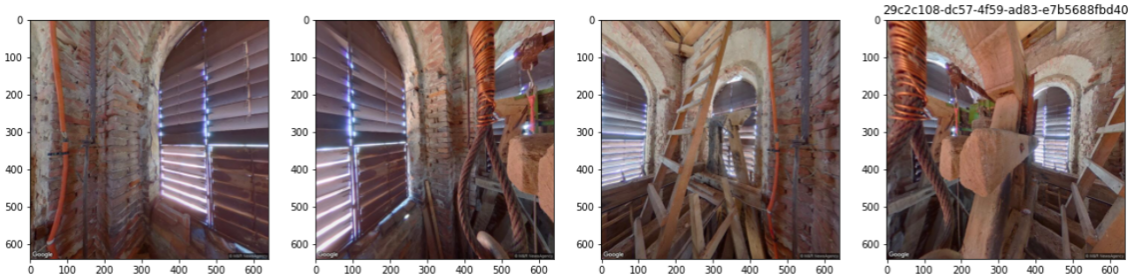

Figure 5.8: 1. set of images that produced the highest error.

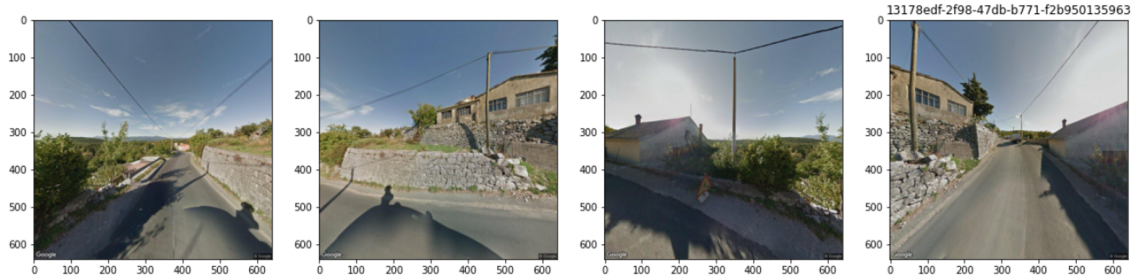

Figure 5.9: 2. set of images that produced the highest error.

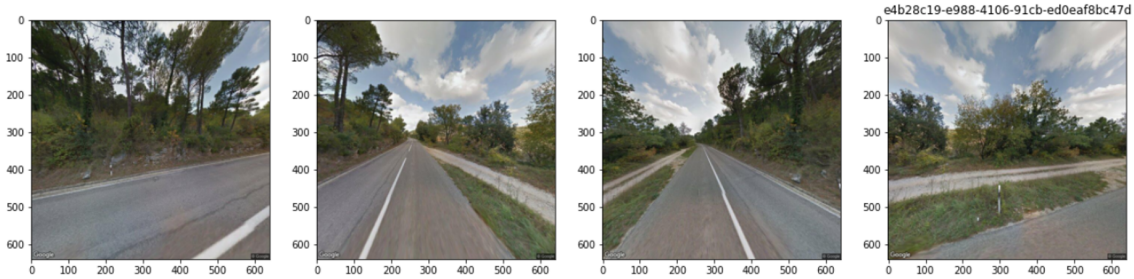

Figure 5.10: 3. set of images that produced the highest error.

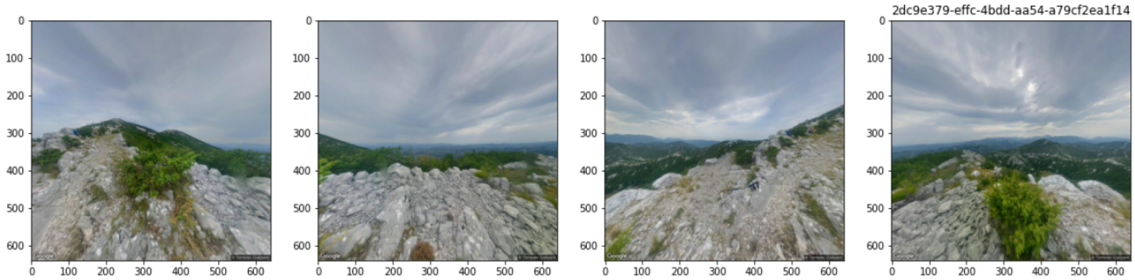

Figure 5.11: 4. set of images that produced the highest error.

List of Figures

List of Tables