COURSE WORK REPORT
In Intelligent Systems
Image classification system

Prepared by: Artemyev Dmitriy
Makhalin Nikolay
Romanova Evgeniya
Tyugay Nikolay

Submitted to: Bahrami Amir Hossein

May, 2023

St. Petersburg

Table of contents

1. Introduction

The invention of digital photography has given mankind many new opportunities. One of them is the use of a digital image to recognize objects, places, people, writing and actions.

Image recognition is used to perform many machine-based visual tasks, such as labeling the content of images with meta tags, performing image content search and guiding autonomous robots, self-driving cars and accident-avoidance systems.

Image recognition is gaining immense popularity and can lead to a variety of new applications in the future, including the following:

- medical diagnosis;
- driverless cars;
- visual search;
- people identification
- quality control;
- facial recognition.

This course work paper presents the results of an investigation into the use of machine learning for landscape image classification. Our research can be a useful starting point for areas such as:

- drone inspection cultural heritage sites;
- replacing traditional multiple and sometimes costly small sensors with a single camera to determine the state of the environment. This will allow you to accept the correct settings for the operating system.
- inspection of forests;
- exploration of shipping lanes for the presence of glaciers;
- augmented reality in tourism, urban planning;
- checking for compliance with building plans after the fact;
- fixing the location of the coastline of water resources for the analysis of flood situations;
- train a neural network to classify diseases or symptoms, for example, for MRI diagnostics
- and so on…

As a result of this course work, we will achieve the following milestones:

1) Verified and cleaned dataset with data analysis

2) Forging the first CNN model with its own hyperparameters

3) Using Bayesian optimization to determine optimal hyperparameters

4) Creation of an Ensemble of models to implement a more confident classification of images

5) Telegram bot capable of receiving images as messages and classifying them according to the 6 mentioned classes.

The goal of the current project is to develop an image classification system. This should consist of a server application and a client application. The server will classify the image received from the client through the Telegram bot using machine learning methods. The methods used in this work should allow achieving a high percentage of image classification accuracy, up to 95% and higher.

In the project, we will not only classify images, but also monitor metrics such as: precision, recall, F1-score, support and confusion matrix. A confusion matrix is a table that is used to define the performance of a classification algorithm.

		Actual class	
		P	**N**
Predicted class	**P**	**TP**	FP
	N	FN	**TN**

Where:

TP (True Positive) is a number of *right predictions* that are *correct* when label is *True and* predicted as *True*.

TN (True Negative) is a number of *right predictions* that are *incorrect* when label is *False and* predicted as *False*.

FP (False Positive) is a number of *not right predictions* that are *incorrect* when label is *False but* predicted as *True***.**

FN (False Negative) is a number of *not right predictions* that are *correct* when label is *True but* predicted as *False***.**

Due to the fact that our Accuracy should aim for 100% and above, we get that:

$$Accuracy = \frac{TN + TP}{TN + FP + TP + FN} \quad \rightarrow 1(100\%)$$

What is the accuracy of the machine learning model for this classification task? Accuracy represents the number of correctly classified data instances over the total number of data instances.

Precision is an accuracy of positive predictions
Precision represents **percent of correct predictions.** In other words, it is **ability not to label** an image **as positive** that is actually **negative.**
Precision is calculated by following equation:

$$\text{Precision} = TP / (TP + FP)$$

Recall is a fraction of positive predictions among all True samples. In other words, it is **ability to find all positive samples.**
Recall is calculated by following equation:

$$\text{Recall} = TP / (TP + FN)$$

F1-score is a so called **weighted harmonic mean of the Precision and Recall**
F1-score also known as balanced F-score or F-measure, as it incorporates Precision and Recall into computation, and, therefore, contributions of Precision and Recall to F1-score are equal
F1-score reaches its best value at 1 and worst score at 0
F1-score is calculated by following equation:

$$\text{F1-score} = 2\ (Recall\ \text{Precision}) / (\text{Recall} + \text{Precision})$$

Support is a number of occurrences of each class in a dataset

2. Approach for solution

2.1. The dataset description

So, firstly, the dataset is public and contains many elements --(https://www.kaggle.com/datasets/puneet6060/intel-image-classification). This is image data of Natural Scenes around the world. Dataset contains around 25k images of size 150x150 pixels distributed under 6 categories - buildings, forest, glacier, mountain, sea, street. There are around 14k images in Train, 3k in Test and 7k in Prediction. It took not so much time to delete mismatched images in classes - it is when we can see mountains in sea image class etc.

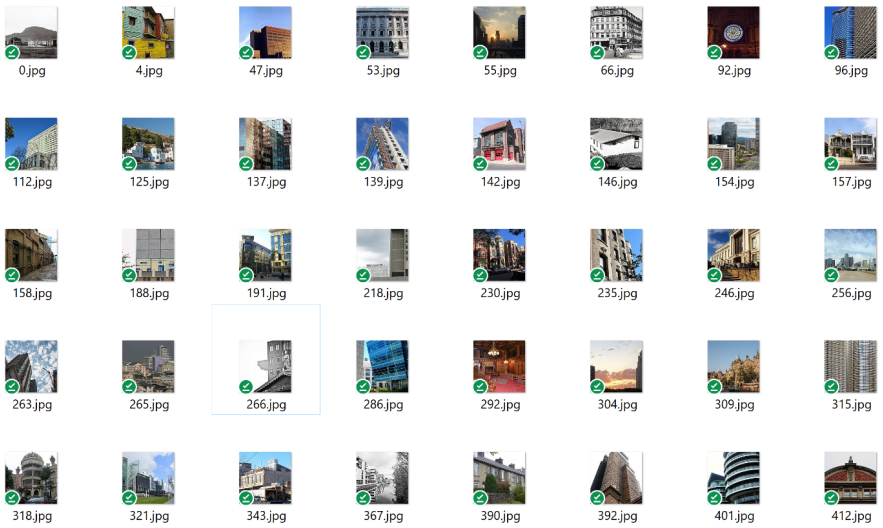
Training dataset contains – 14 034 images:

- buildings -2191;
- forest -2271;
- sea -2274;
- street -2382;
- glacier -2404;
- mountain -2512.

Testing dataset contains – 3 000 images:

- buildings -525;
- forest - 474;
- sea -510;
- street -437;
- glacier -501;
- mountain -553.

The training set will be used from the “seg_train” folder to compile and fit in our machine models. For the training set we will use images from the “seg_train” folder.



for example this is “building” class folder with 2192 images

Figure 1.

2.2. Digital Twin

A digital twin is used - this is a virtual copy of a physical product, process or system. It acts as a bridge between physical and virtual worlds. Digital twin consists of three parts: the physical product, the virtual product and the linkage between physical and virtual product.

In our case, the developed system, according to image classification, can be an integral part of the digital twin of the city.

2.3. Swimlane Diagram

A swimlane diagram is a type of flowchart that describes activities that group them by roles, resources, and elements. This diagram graphically represents the steps in the process. The Swimlane diagram divides the process into sections to show what steps need to be taken to solve the problem of image recognition and create a Telegram bot app. The swimlane diagram for the project is presented on fig. 2.

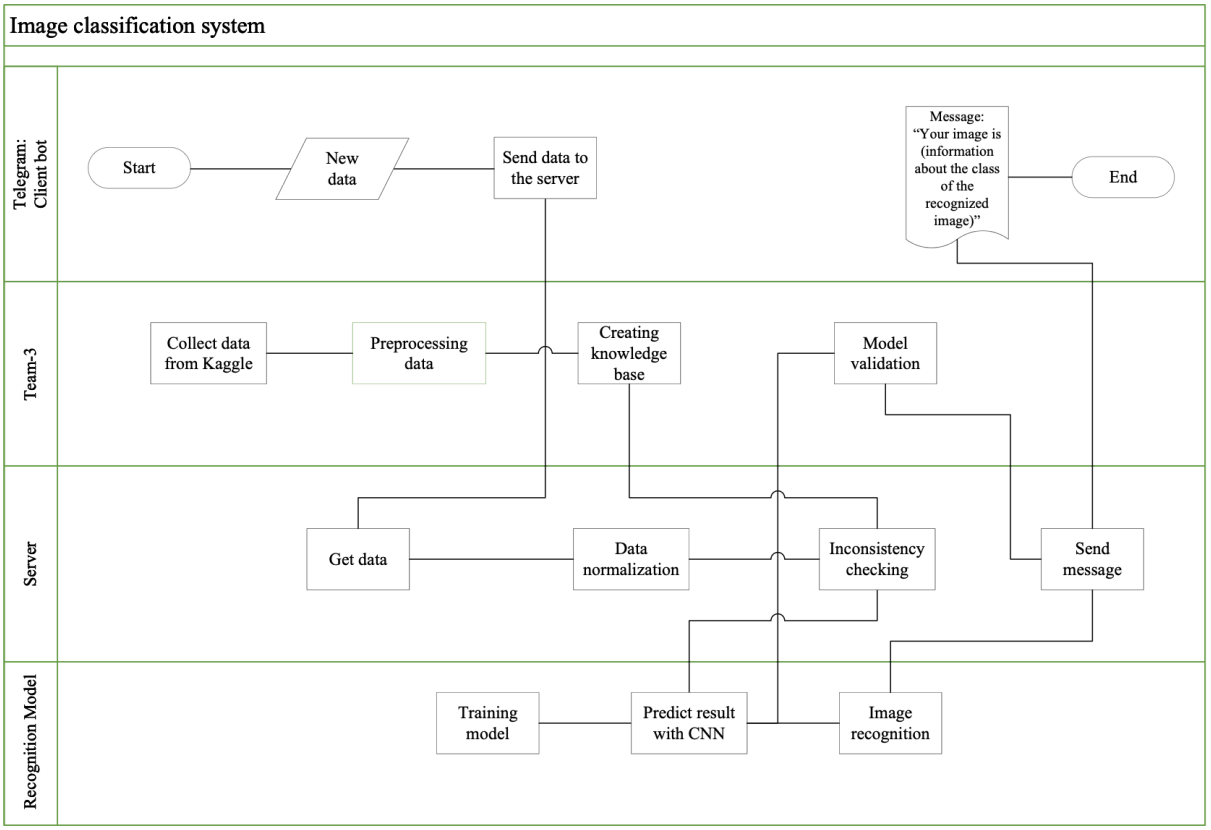


Figure 2. Swimlane diagram

2.4. Detailed description of the processes (IDEF0)

The Integration Definition for Function Modeling (IDEF0) is a function modeling methodology for describing manufacturing functions, which offers a functional modeling language for the analysis, development, reengineering and integration of information systems, business processes or software engineering analysis.

IDEF0 below shows the functional flow of data and main process for the image recognition task (fig. 3.).

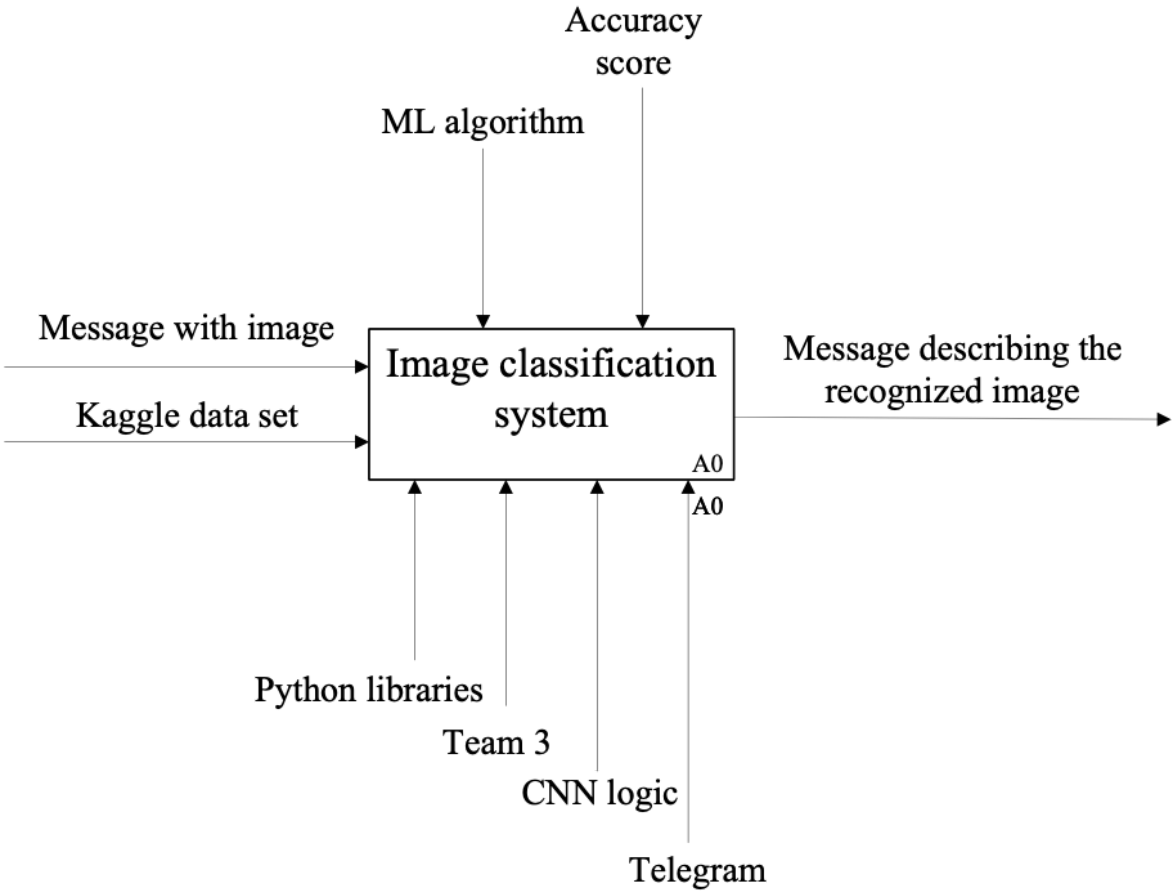


Figure 3. IDEF0: System overview

Figure 4 displays the decomposition of the main process (node A0). The main process consists of 4 steps:

- image preparation;
- model creation;
- image recognition;
- result output.

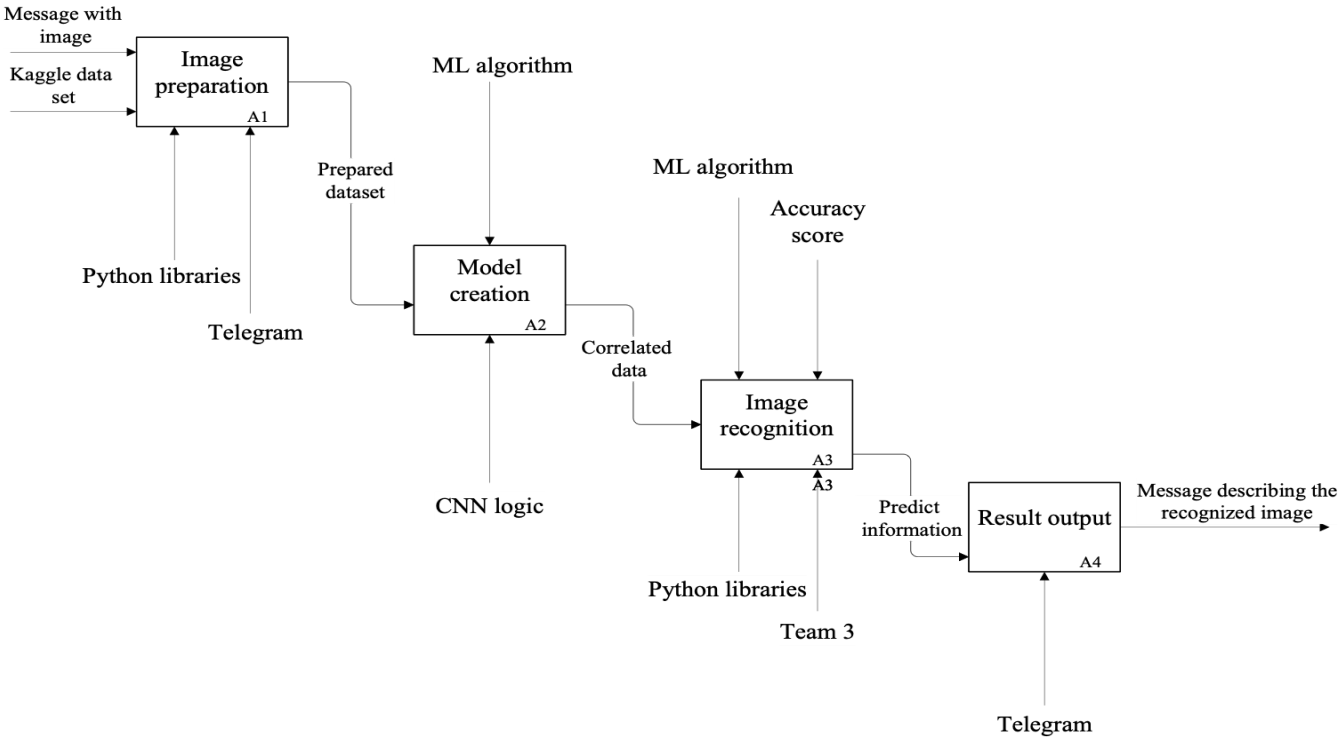


Figure 4. IDEF0: Stages of creating a model for landscape image recognition

Figure 5 shows the decomposition of node A3, which consists of 3 common steps for any machine learning algorithms.

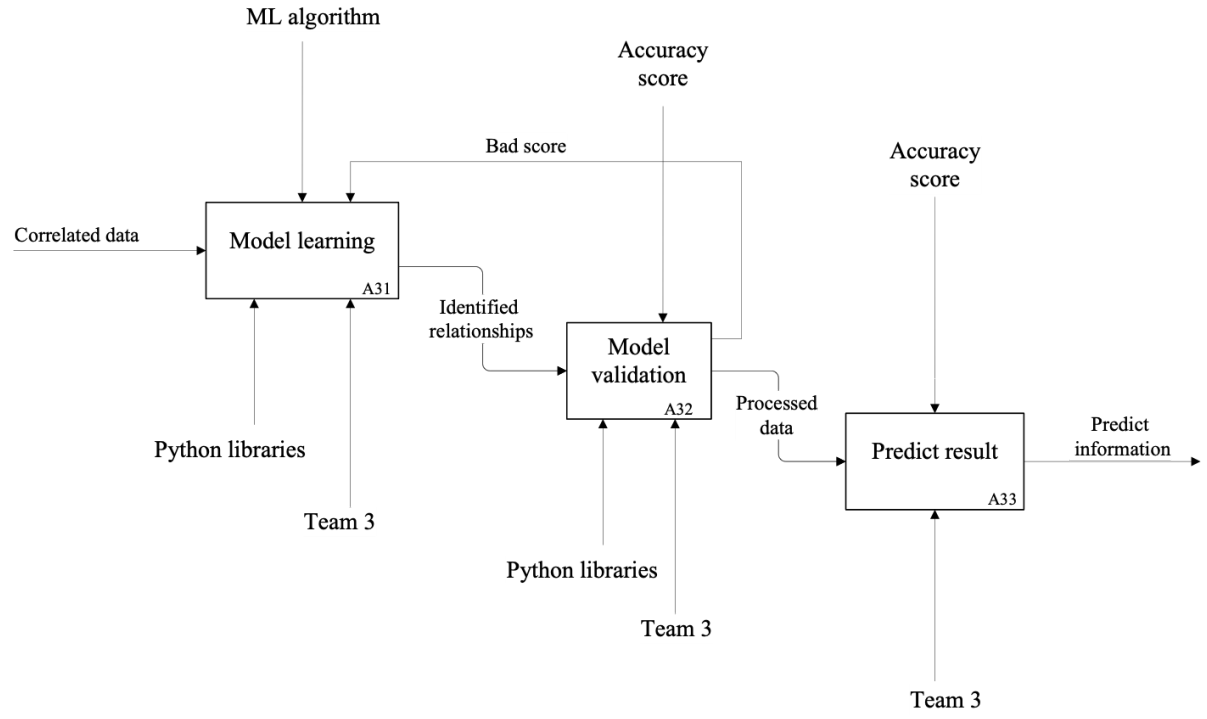


Figure 5. IDEF0: decomposition node - A3

2.5. Algorithm for the solution

2.5.1. General algorithm for the solution

The algorithm for finding a solution involves several steps:

1. Identify important features that will help identify possible methods to use;
2. Identify the knowledge needed to solve the problem;
3. Choose a technique for solving the problem;
4. Create a description of the problem and solution (IDEF0 and Swimlane diagrams);
5. Examine the data set and evaluate it;
6. Develop the first representation of the solution (knowledge base, application, etc.);
7. Evaluate the results;
8. Refine and improve the solution by adding more details and customizing it.

2.5.2. What is CNN?

In deep learning, a convolutional neural network (CNN) is a class of artificial neural network most commonly applied to analyze visual imagery. CNNs use a mathematical operation called convolution in place of general matrix multiplication in at least one of their layers. They are specifically designed to process pixel data and are used in image recognition and processing. They have applications in image and video recognition, recommender systems, image classification, image segmentation, medical image analysis, natural language processing, brain–computer interfaces, and financial time series.

Yann LeCun et al. (1989)[45] used back-propagation to learn the convolution kernel coefficients directly from images of hand-written numbers. Learning was thus fully automatic, performed better than manual coefficient design, and was suited to a broader range of image recognition problems and image types.

The human brain processes a huge amount of information the second we see an image. Each neuron works in its own receptive field and is connected to other neurons in a way that they cover the entire visual field. Just as each neuron responds to stimuli only in the restricted region of the visual field called the receptive field in the biological vision system, each neuron in a CNN processes

data only in its receptive field as well. The layers are arranged in such a way so that they detect simpler patterns first (lines, curves, etc.) and more complex patterns (faces, objects, etc.) further along

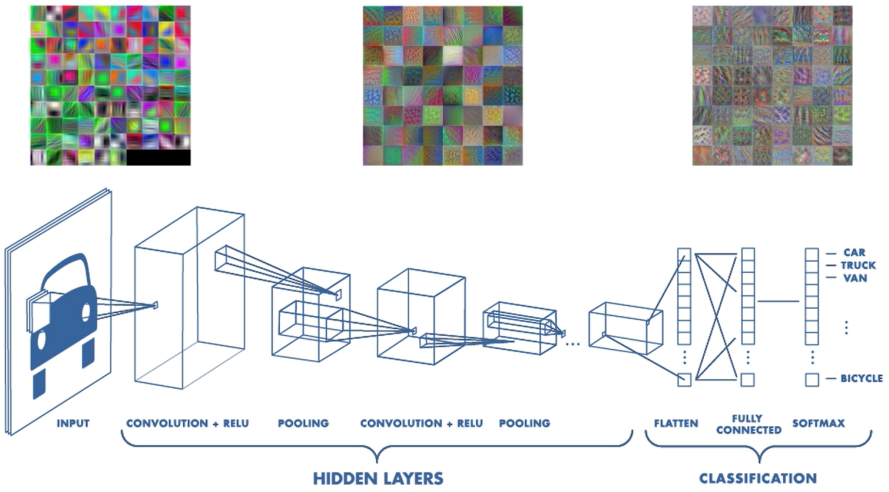


Figure 6. Typically CNN

A CNN typically has three layers: a convolutional layer, a pooling layer, and a fully connected layer (fig.6) .

Convolution Layer

The convolution layer is the core building block of the CNN. It carries the main portion of the network's computational load.

This layer performs a dot product between two matrices, where one matrix is the set of learnable parameters otherwise known as a kernel, and the other matrix is the restricted portion of the receptive field. The kernel is spatially smaller than an image but is more in-depth. This means that, if the image is composed of three (RGB) channels, the kernel height and width will be spatially small, but the depth extends up to all three channels.

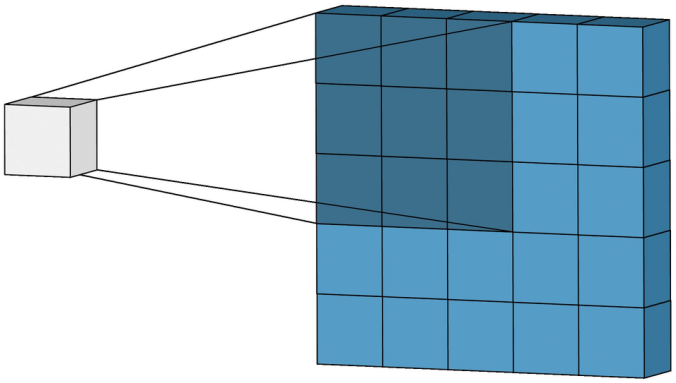


Figure 7. Illustration of Convolution Operation

During the forward pass, the kernel slides across the height and width of the image-producing the image representation of that receptive region(fig.7). This produces a two-dimensional representation of the image known as an activation map that gives the response of the kernel at each spatial position of the image. The sliding size of the kernel is called a stride.

Motivation behind Convolution

Convolution leverages three important ideas that motivated computer vision researchers: sparse interaction, parameter sharing, and equivariant representation. Let's describe each one of them in detail.

Trivial neural network layers use matrix multiplication by a matrix of parameters describing the interaction between the input and output unit. This means that every output unit interacts with every input unit. However, convolution neural networks have sparse interaction. This is achieved by making kernel smaller than the input e.g., an image can have millions or thousands of pixels, but while processing it using kernel we can detect meaningful information that is of tens or hundreds of pixels. This means that we need to store fewer parameters that not only reduces the memory requirement of the model but also improves the statistical efficiency of the model.

If computing one feature at a spatial point (x1, y1) is useful then it should also be useful at some other spatial point say (x2, y2). It means that for a single two-dimensional slice i.e., for creating one activation map, neurons are

constrained to use the same set of weights. In a traditional neural network, each element of the weight matrix is used once and then never revisited, while convolution network has shared parameters i.e., for getting output, weights applied to one input are the same as the weight applied elsewhere.

Due to parameter sharing, the layers of convolution neural network will have a property of equivariance to translation. It says that if we changed the input in a way, the output will also get changed in the same way.

Pooling Layer

The pooling layer replaces the output of the network at certain locations by deriving a summary statistic of the nearby outputs. This helps in reducing the spatial size of the representation, which decreases the required amount of computation and weights. The pooling operation is processed on every slice of the representation individually.

There are several pooling functions such as the average of the rectangular neighborhood, L2 norm of the rectangular neighborhood, and a weighted average based on the distance from the central pixel. However, the most popular process is max pooling, which reports the maximum output from the neighborhood (fig. 8).

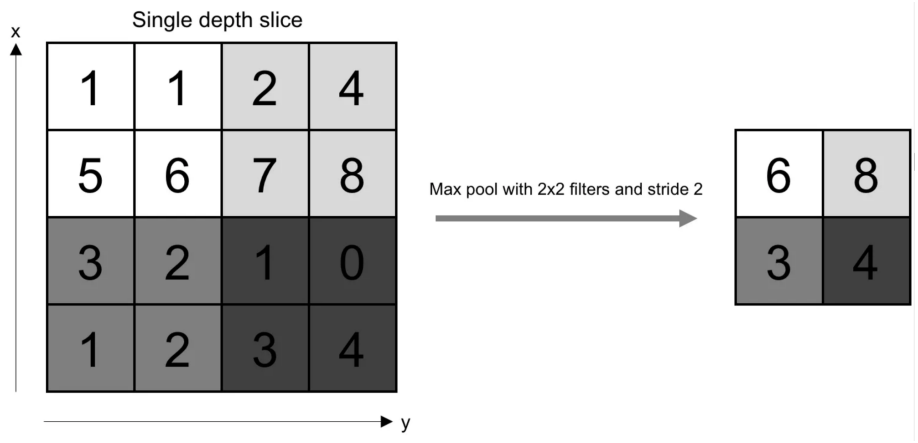


Figure 8. Pooling Operation

If we have an activation map of size W x W x D, a pooling kernel of spatial size F, and stride S, then the size of output volume can be determined by the following formula:

$$W_{out} = \frac{W - F}{S} + 1$$

This will yield an output volume of size Wout x Wout x D.

In all cases, pooling provides some translation invariance which means that an object would be recognizable regardless of where it appears on the frame.

Fully Connected Layer

Neurons in this layer have full connectivity with all neurons in the preceding and succeeding layer as seen in regular FCNN. This is why it can be computed as usual by a matrix multiplication followed by a bias effect.

The FC layer helps to map the representation between the input and the output.

Non-Linearity Layers

Since convolution is a linear operation and images are far from linear, non-linearity layers are often placed directly after the convolutional layer to introduce non-linearity to the activation map.

There are several types of non-linear operations, the popular ones being:

1. Sigmoid

The sigmoid non-linearity has the mathematical form:

$$\sigma(\kappa) = 1/(1+e^{-}\kappa).$$

It takes a real-valued number and "squashes" it into a range between 0 and 1.

However, a very undesirable property of sigmoid is that when the activation is at either tail, the gradient becomes almost zero. If the local gradient becomes very small, then in backpropagation it will effectively "kill" the gradient. Also, if the data coming into the neuron is always positive, then the output of sigmoid will be either all positives or all negatives, resulting in a zig-zag dynamic of gradient updates for weight.

2. Tanh

Tanh squashes a real-valued number to the range [-1, 1]. Like sigmoid, the activation saturates, but — unlike the sigmoid neurons — its output is zero centered.

3. ReLU

The Rectified Linear Unit (ReLU) has become very popular in the last few years. It computes the function:

$$f(\kappa)=\max(0,\kappa).$$

In other words, the activation is simply threshold at zero.

In comparison to sigmoid and tanh, ReLU is more reliable and accelerates the convergence by six times.

Unfortunately, a con is that ReLU can be fragile during training. A large gradient flowing through it can update it in such a way that the neuron will never get further updated. However, we can work with this by setting a proper learning rate.

2.5.3. Bayesian Optimization

Bayesian optimization is a sequential design strategy for global optimization of black-box functions that does not assume any functional forms. It is usually employed to optimize expensive-to-evaluate functions. Bayesian Optimization has been widely used for the hyperparameter tuning purpose in the Machine Learning world.

The Overview of Hyperparameter Optimization

There are 4 main types of methods of hyperparameter optimization: Manual Search, Random Search, Grid Search, and Bayesian Optimization.

Bayesian Optimization differs from Random Search and Grid Search in that it improves the search speed using past performances, whereas the other two methods are uniform (or independent) of past evaluations. In that sense, Bayesian Optimization is like Manual Search. Let’s say you are manually

optimizing the hyperparameter of a Random Forest regression model. Firstly, you would try a set of parameters, then look at the result, change one of the parameters, rerun, and compare the results, so that way you know whether you are going towards the right direction. Bayesian Optimization does a similar thing — the performance of your past hyperparameter affects the future decision. In comparison, Random Search and Grid Search do not take into account past performance when determining new hyperparameters to evaluate. Thus, Bayesian Optimization is a much more efficient method.

So, Bayesian Optimization builds a probability model of the objective function and uses it to select hyperparameter to evaluate in the true objective function.

Let's separate this sentence into three parts.

First part: Bayesian Optimization builds a probability model of the objective function

The true objective function is a fixed function. Let's say it is supposed to look like the picture below (fig.9), but as we mentioned, we don't know this at the beginning of the hyperparameter tuning.

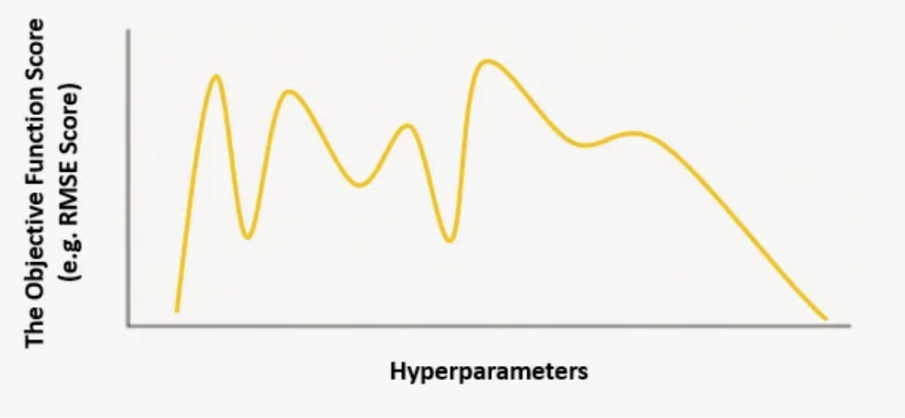


Figure 9. The true objective function

If there are unlimited resources, we would compute every single point of the objective function so that we know its actual shape (In our example, keep calling the Random Forest Regression model until we have the RMSE scores for all possible hyperparameter combinations). However, that's impossible. So let's say we only have 10 samples from the true objective function, represented as black circles in next picture (fig.10):

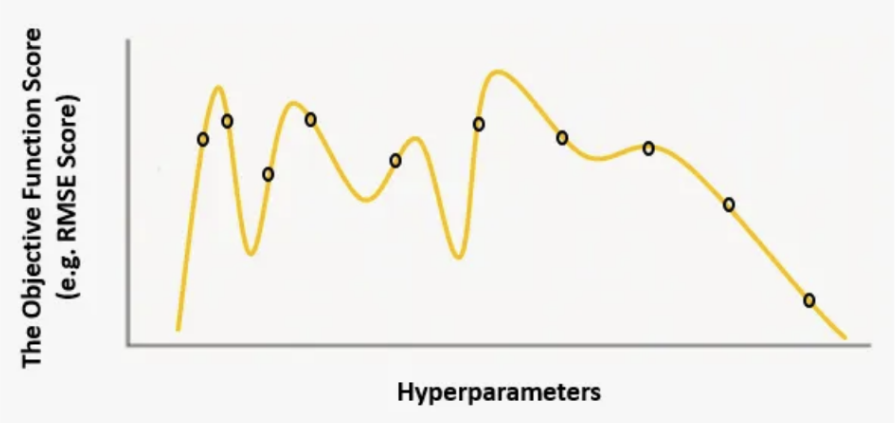


Figure 10. The true objective function

Using these 10 samples, we need to build a surrogate model (also called the response surface model) to approximate the true objective function.

Second part: And use it to select hyperparameters

Now we have 10 samples of the objective function and how should we decide which parameter to try as the 11th sample? We need to build an acquisition function (also called the selection function). The next hyperparameter of choice is where the acquisition function is maximized. In Fig 11, the green shade is the acquisition function and the red straight line is where it is maximized. Therefore the corresponding hyperparameter and its objective function score, represented as a red circle, is used as the 11th sample to update the surrogate model.

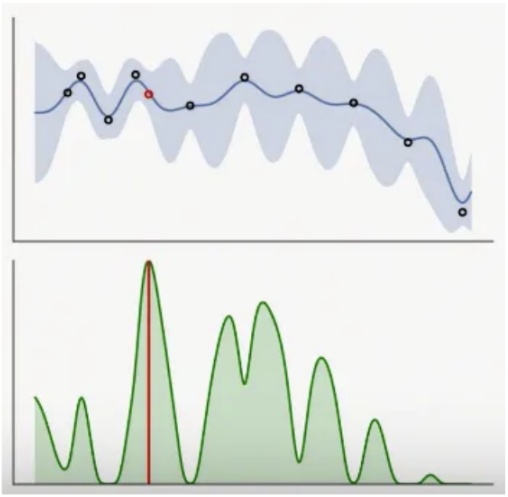


Figure 11. Maximize acquisition function to select the next point

Third part: To evaluate in the true objective function

As described above, after using an acquisition function to determine the next hyperparameter, the true objective function score of this new hyperparameter is obtained. Since the surrogate model has trained on the (hyperparameter, true objective function score) pairs, adding a new data point updates the surrogate model.

Repeat the above steps until the max time or max iteration is reached. Now we have an accurate approximation of the true objective function and can easily find the global minimum from the past evaluated samples.

2.5.4. VGG16

VGG16 is a 16-layer deep neural network. VGG16 is thus a relatively extensive network with a total of 138 million parameters — it's huge even by today's standards. However, the simplicity of the VGGNet16 architecture is its main attraction.

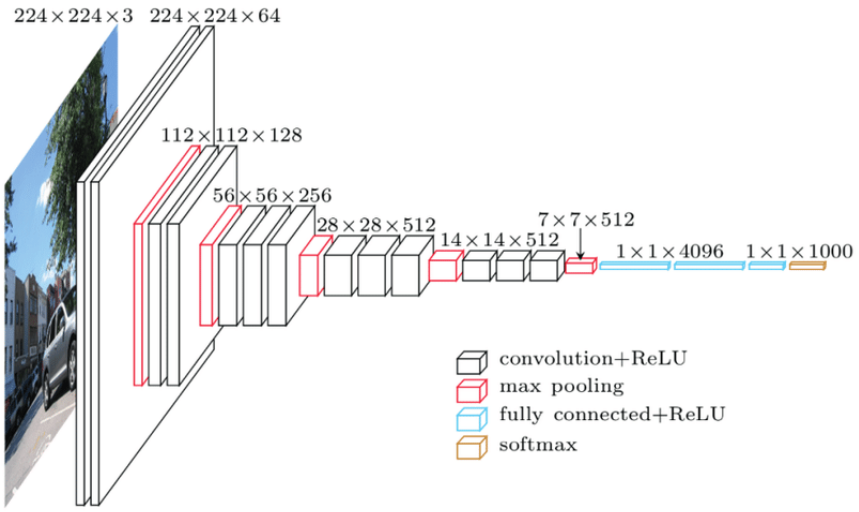


Figure 12. VGG16

2.5.5. Ensemble model

Anytime we're trying to make an important decision, we try to collect as much information as possible and reach out to experts for advice. The more information we can gather, the more we (and those around us) trust the decision-making process.

Machine learning predictions follow a similar behavior. Models process given inputs and produce an outcome. The outcome is a prediction based on what pattern the models see during the training process.

Ensemble models are a machine learning approach to combine multiple other models in the prediction process. These models are referred to as base estimators. Ensemble models offer a solution to overcome the technical challenges of building a single estimator.

There is 4 types of ensemble modeling techniques (Bagging, Boosting, Stacking, Blending)

2.5.5.1 Bagging

The idea of bagging is based on making the training data available to an iterative learning process. Each model learns the error produced by the previous model using a slightly different subset of the training data set. Bagging reduces variance and minimizes overfitting. One example of such a technique is the random forest algorithm.

Bootstrap Aggregation (Bagging)

This technique is based on a bootstrapping sampling technique. Bootstrapping creates multiple sets of the original training data with replacement. Replacement enables the duplication of sample instances in a set. Each subset has the same equal size and can be used to train models in parallel.

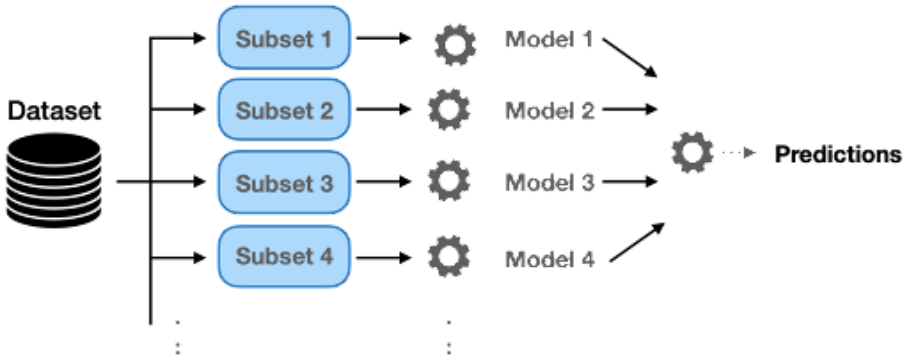


Figure 13. Bagging technique

Bagging technique to make final predictions by combining predictions from multiple models (fig.13).

Random forest

This technique uses a subset of training samples as well as a subset of features to build multiple split trees. Multiple decision trees are built to fit each training set. The distribution of samples/features is typically implemented in a random mode.

Extra-trees Ensemble

Here's another ensemble technique where the predictions are combined from many decision trees. Similar to random forest, it combines a large number of decision trees. However, the extra trees use the whole sample while choosing the splits randomly.

2.5.5.2 Boosting

Adaptive Boosting (AdaBoost)

This is an ensemble of algorithms, where we build models on the top of several weak learners . As we mentioned earlier, those learners are called weak because they are typically simple with limited prediction capabilities. The adaptation capability of AdaBoost made this technique one of the earliest successful binary classifiers.

Sequential decision trees

These were the core of such adaptability where each tree adjusts its weights based on prior knowledge of accuracies. Hence, we perform the training in such a technique in a sequential (rather than parallel) process. In this technique, the process of training and measuring the error in estimates can be repeated for a given number of iterations or when the error rate is not changing significantly.

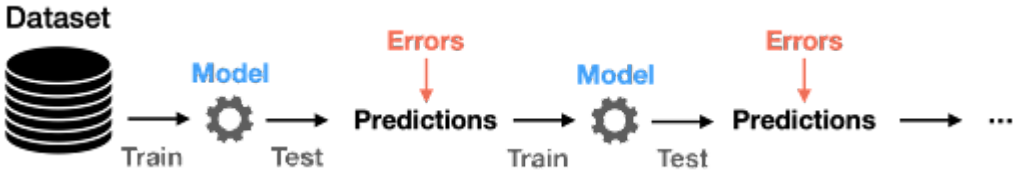


Figure 14. Boosting technique

Gboosting

Gradient boosting algorithms are great techniques that have high predictive performance. Xgboost, LightGBM and CatBoost are popular boosting algorithms you can use for regression and classification problems.

2.5.5.3 Stacking

Stacking is similar to boosting models; they produce more robust predictors. Stacking is a process of learning how to create such a stronger model from all weak learners' predictions.

Figure 15. Stacking technique

2.5.5.4 Blending

Very similar to the stacking approach, except the final model is learning the validation and testing dataset along with predictions. Hence, the features used is extended to include the validation set.

2.6. Technical platform

Choosing the right technical platform is crucial when implementing a project because it directly impacts the system performance, scalability, and reliability. Here are some reasons why it is important to carefully choose a technical platform:

1. Performance: The platform can affect the speed and responsiveness of our project. A poorly designed platform may cause our system to respond slowly or even crash during peak usage times. A robust platform, on the other hand, can ensure that our system can handle high volumes of requests and respond quickly.

2. Scalability: As our project becomes more popular, it will need to handle an increasing number of users and messages. Choosing a platform that can scale with our project's growth is crucial to ensuring that our system can continue to operate smoothly as it grows.

3. Integration: Our project will need to integrate with other systems and platforms, such as company's CRM or social media accounts. Choosing a platform that can integrate easily with these systems can save time and effort and make it easier to build a fully functional AI-system.

4. Security: Our project may handle sensitive user data such as personal information and photos. Choosing a platform with robust security features is essential to protecting user data and ensuring compliance with data protection regulations.

5. Customization: Our AI-system's functionality and design should match business needs. At the initial stage, it is necessary to have the widest possible opportunities for further choice.

6. Support: Technical issues can arise at any time during the development and operation of our system. Choosing a platform with reliable support and documentation can help us resolve issues quickly and minimize downtime.

Overall, choosing the right technical platform is essential to building a successful project. Careful consideration of factors such as performance, scalability, integration, security, customization, and support can help ensure that our AI-system is reliable, effective, and meets our business needs (fig.16).

Figure 16. Technical platform requirements

2.6.1. User interface

The user interface of the AI-based image recognition system on Telegram messenger is designed to be user-friendly and easy to navigate. It consists of a chatbot that users can interact with to submit images for recognition and receive results.

To use the system, users simply need to open a chat with the chatbot and follow the prompts to submit an image. The chatbot will guide the user through

the process, providing instructions for image submission and any necessary technical requirements.

Once the image is submitted, the chatbot will use AI-based image recognition technology to analyze the image and provide results. The results include information both about objects and allows you to track the technical parameters of the model and system.

The user interface may also include additional future features, such as real-time recognition, voice input capabilities, or chatbot integration with other apps or services. These features are designed to enhance the user experience and provide a seamless and efficient image recognition process.

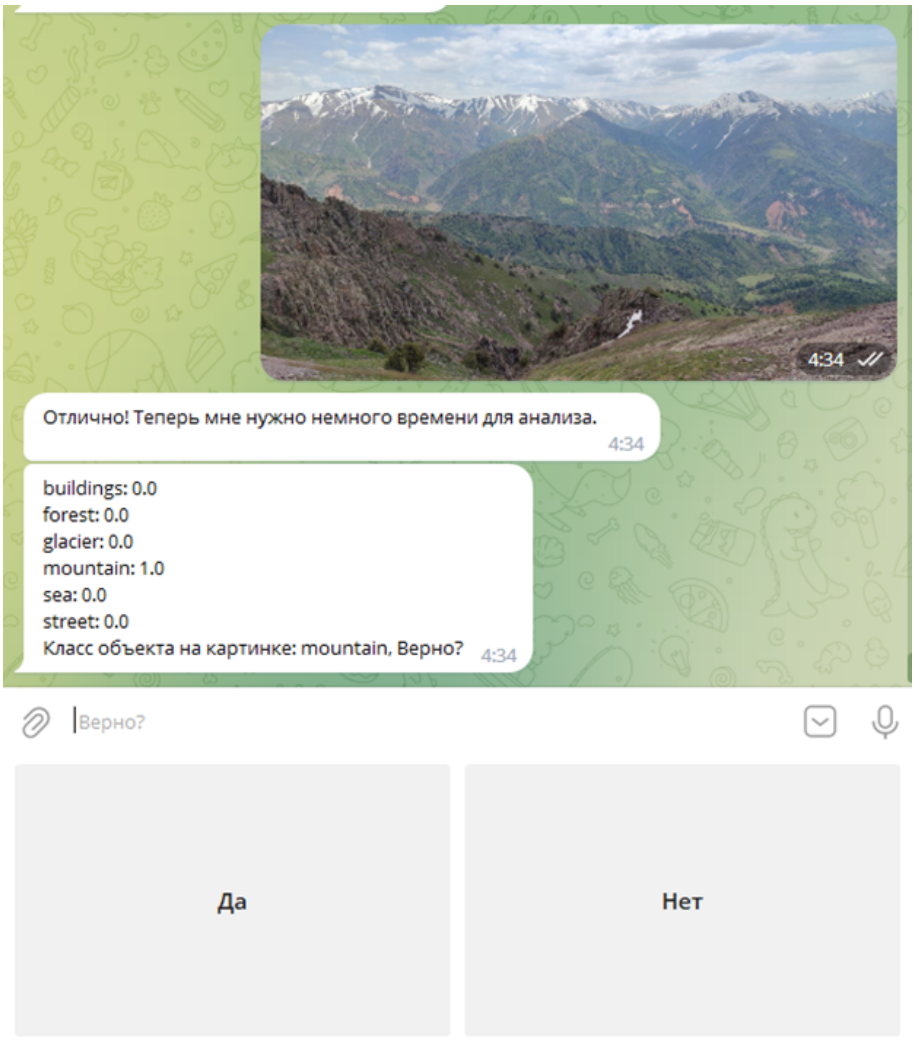


Figure 17. Working application example

2.6.2. Messenger platform

Telegram messenger was chosen as the platform for the AI-based image recognition system for several reasons:

1. Large User Base: Telegram messenger has a large and growing user base, with over 500 million active users worldwide. This provides a significant audience for the AI-based image recognition system.

2. Messaging Capabilities: Telegram messenger has advanced messaging capabilities, including group chats, channels, and file sharing. This makes it an ideal platform for users to submit images for recognition and receive results.

3. Secure and Private: Telegram messenger has a reputation for being a secure and private messaging platform, which is essential when handling sensitive information like images.

4. Bot API: Telegram messenger provides a Bot API that allows developers to create custom chatbots for various purposes. This was a crucial factor in developing the AI-based image recognition system on Telegram, as it allows for seamless integration with the messaging platform.

5. Open Platform: Telegram messenger is an open platform that encourages developers to create and innovate new features. This provides ample opportunities for future enhancements and collaborations.

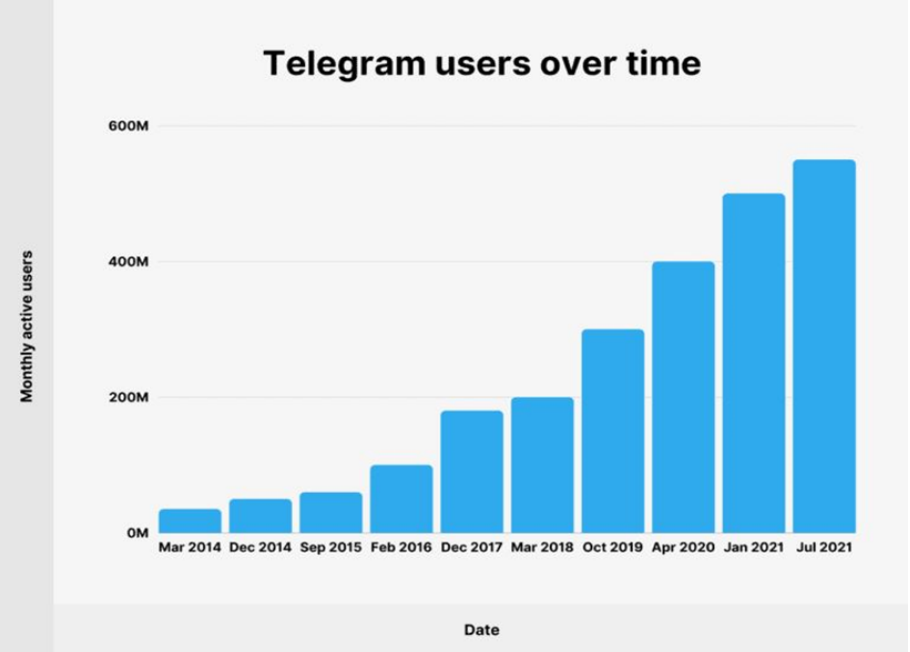


Figure 18. Telegram users over time

2.6.3. Telegram bot architecture

The architecture of a Telegram bot based on the python-telegram-bot library typically consists of several components that work together to receive and process incoming messages from users. Here's a breakdown of the key components:

1. Telegram Bot API: The Telegram Bot API is a RESTful API that enables communication between our bot and the Telegram messaging platform. Our bot uses this API to send and receive messages from users.
2. python-telegram-bot library: The python-telegram-bot library is a Python wrapper for the Telegram Bot API that makes it easy to build Telegram bots. It provides an interface for handling incoming messages, sending responses, and managing bot updates.
3. Webhook or polling: The python-telegram-bot library supports two methods for receiving incoming messages: webhook and polling. With a webhook, the Telegram server sends messages directly to our bot's endpoint. With polling, our bot periodically checks the Telegram server for new messages. Webhooks are generally faster and more efficient, but require a web server with SSL certificate. Polling is simpler to set up, but may cause more delays.
4. Message handlers: Message handlers are functions that are executed when our bot receives a specific type of message from a user. For example, we might have a message handler that responds to text messages with

a specific keyword, or one that sends a menu of options when a user starts a new conversation with the bot.

5. Conversation handler: The ConversationHandler is a special type of message handler that enables us to manage multi-step conversations with users. It allows us to define a set of states, transitions between states, and functions to handle each state.

6. Database: Depending on the complexity of our bot, we may need to store data in a database. This can include user preferences, session data, or other information. The python-telegram-bot library includes support for several database backends, including SQLite and PostgreSQL.

Overall, the architecture of a Telegram bot based on the python-telegram-bot library is modular and flexible, allowing us to easily build and customize our bot's functionality.

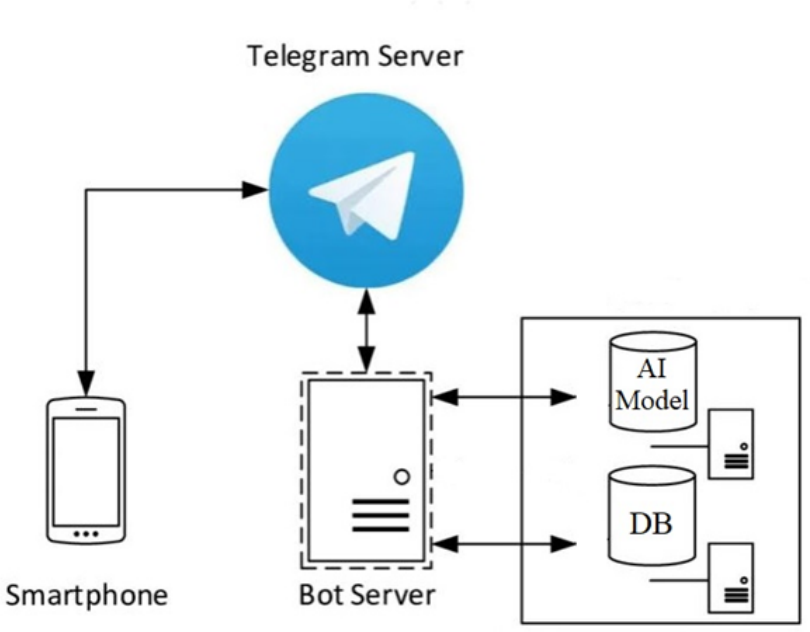


Figure 19. Bot architecture

2.6.4. VPS

A Virtual Private Server (VPS) is a type of hosting service that provides a virtual machine with dedicated resources on a shared physical server. VPS servers offer several benefits for creating small projects, including:

1. Cost-effective: VPS servers are typically more affordable than dedicated servers, making them a cost-effective solution for small projects that require dedicated resources.

2. Scalability: VPS servers can be easily scaled up or down depending on our project's needs. This means that we can add or remove resources like CPU, RAM, or storage as needed without having to move to a new server.

3. Control and flexibility: With a VPS server, we have full root access and control over the environment, which allows us to install and configure the software we need for our project. This flexibility is ideal for small projects that may have specific requirements or need customization.

4. Security: VPS servers are isolated from other virtual machines on the same physical server, providing an extra layer of security for our project. Additionally, many VPS hosting providers offer security features such as firewalls, DDoS protection, and regular backups.

5. Reliability: VPS servers are typically more reliable than shared hosting services, as they provide dedicated resources and are less susceptible to performance issues caused by other users on the same server.

Overall, VPS servers are an excellent choice for small projects that require dedicated resources, flexibility, and security, while also being cost-effective and reliable.

2.6.5. Database

SQLite is a lightweight, self-contained database management system that has several benefits for small projects:

1. Easy to use: SQLite is simple and easy to use, especially for small projects with low complexity. Its simple design allows users to create and manage databases with minimal effort.

2. Low overhead: SQLite has a small footprint, which means that it requires minimal system resources to run. This makes it ideal for small projects with limited system resources.

3. Self-contained: SQLite is a self-contained database system that stores all data in a single file, making it easy to manage and transport between different systems.

4. No server required: Unlike other database management systems, SQLite does not require a separate server to run. This means that developers can create and test their applications on a local machine without the need for a dedicated server.

5. Cross-platform compatibility: SQLite is compatible with a wide range of operating systems, including Windows, Linux, macOS, and mobile operating systems like iOS and Android.

6. ACID-compliant: SQLite is an ACID-compliant database, which means that it guarantees data consistency and integrity even in the event of system failures or power outages.

7. Low maintenance: Due to its self-contained nature, SQLite requires minimal maintenance compared to other database management systems. It does not require regular backups or maintenance tasks like optimization or indexing.

Overall, SQLite is a good choice for small projects that require a lightweight and easy-to-use database management system. Its low overhead, self-contained nature, cross-platform compatibility, and ACID compliance make it a popular choice among developers who want to build applications quickly and efficiently.

3. Application development

3.1. Dataset review and preparing

As was mentioned in the previous paragraph, the dataset contains a lot of images and before using them we were able to check all images. All images stored in 6 folders corresponding to the name of the category (buildings, forest, glacier, mountain, sea, street). That's why image of the "sea" cannot be in folder named "buildings" and so on. After checking and deleting all irrelevant images we can move on.

```
from keras.layers import Input
from bayes_opt import BayesianOptimization
from tqdm import tqdm
from sklearn.utils import shuffle
from sklearn.metrics import classification_report
from keras.utils import to_categorical
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Dense, Flatten, Dropout
from tensorflow.keras.callbacks import EarlyStopping
```

```
<ipython-input-3-5af3f7e24a7a>:11: DeprecationWarning: `import kerastuner` is depr
  import kerastuner as kt
```

```
[4] class_names=['buildings', 'forest', 'glacier','mountain','sea','street']
    class_names_label = {class_name:i for i, class_name in enumerate(class_names)}
    nb_classes=len(class_names)
    print(class_names_label)
    image_size=(150,150)
    work_path='/content/drive/Othercomputers/Thinkpad/Project'
```

```
{'buildings': 0, 'forest': 1, 'glacier': 2, 'mountain': 3, 'sea': 4, 'street': 5}
```

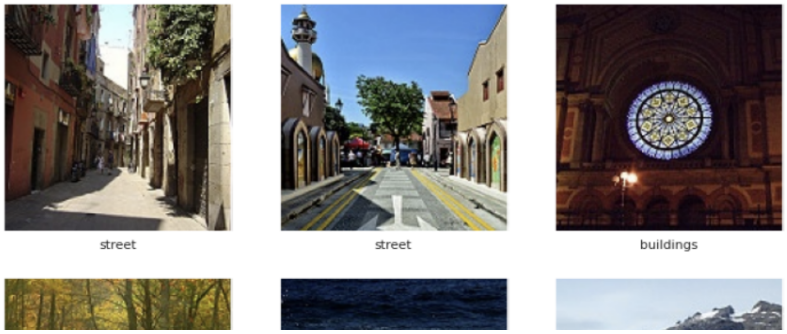
Let's prepare the function ***load_data()*** to load the dataset and its labels.

```
#load dataset
def load_data():
    #directory=os.path.abspath("")
    directory=work_path + '/dataset'
    category=['seg_train','seg_test']
    output=[]
    for eachcategory in category:
        path=os.path.join(directory,eachcategory)
        images=[]
        labels=[]

        for folder in os.listdir(path):
            label=class_names_label[folder]
            for file in os.listdir(os.path.join(path, folder)):
                img_path=os.path.join(os.path.join(path,folder),file)
                image=cv2.imread(img_path)
                image=cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
                image=cv2.resize(image,image_size)

                images.append(image)
                labels.append(label)
        images=np.array(images, dtype='float32')
        labels=np.array(labels, dtype='int32')

        output.append((images,labels))
    return output
```

```
(train_images, train_labels), (test_images, test_labels) = load_data()
train_x, val_x, train_y, val_y = train_test_split(train_images,
                                                  train_labels,
                                                  stratify=train_labels,
                                                  random_state=48,
                                                  test_size=0.05)
(test_x, test_y)=(test_images, test_labels)
```

Let's define another function called ***display_example*** to show collected data.

```
def display_example(class_name, images, labels):
    figsize=(15,15)
    fig=plt.figure(figsize=figsize)
    #fig.subtitle("Some examples of images from the datset", fontsize=16) - depr
    for i in range(10):
        plt.subplot(5,5, i+1)
        plt.yticks([])
        plt.xticks([])
        plt.grid(False)
        #image=cv2.resize(images[i], figsize)
        #plt.imshow(image.astype(np.uint8))
        plt.imshow(images[i].astype(np.uint8))
        plt.xlabel(class_names[labels[i]],fontsize = 8)
    plt.show()

display_example(class_names,train_images,train_labels)
```



So, at this moment we have to convert a class vector to a binary class matrix.

```
train_x = train_x / 255.0
val_x = val_x / 255.0
test_x = test_x / 255.0

train_y = to_categorical(train_y)
val_y = to_categorical(val_y)
test_y = to_categorical(test_y)
```

And let’s see what we have in our datasets

```
print(train_x.shape)
print(train_y.shape)
print(val_x.shape)
print(val_y.shape)
print(test_x.shape)
print(test_y.shape)
```

```
(13281, 150, 150, 3)
(13281, 6)
(700, 150, 150, 3)
(700, 6)
(2964, 150, 150, 3)
(2964, 6)
```

3.2. First model

It's a very simple model with 2 convolutional layers.

```
model = Sequential([
     tf.keras.layers.Conv2D(32,(3,3), activation='relu', input_shape=(150,150,3)),
     tf.keras.layers.MaxPooling2D(2,2),
     tf.keras.layers.Conv2D(32,(3,3), activation='relu'),
     tf.keras.layers.MaxPooling2D(2,2),
     tf.keras.layers.Flatten(),
     tf.keras.layers.Dense(128,activation=tf.nn.relu),
     tf.keras.layers.Dense(6,activation=tf.nn.softmax)
     ])
```

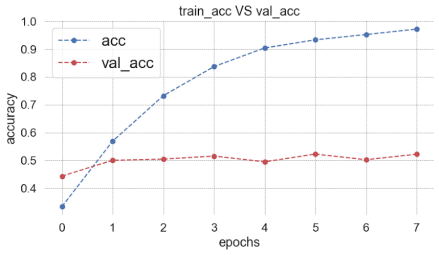
compile with Adam optimizer:

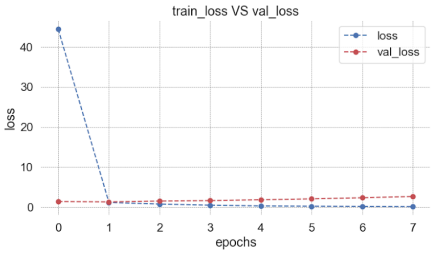
```
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
```

and look at the fit and results:

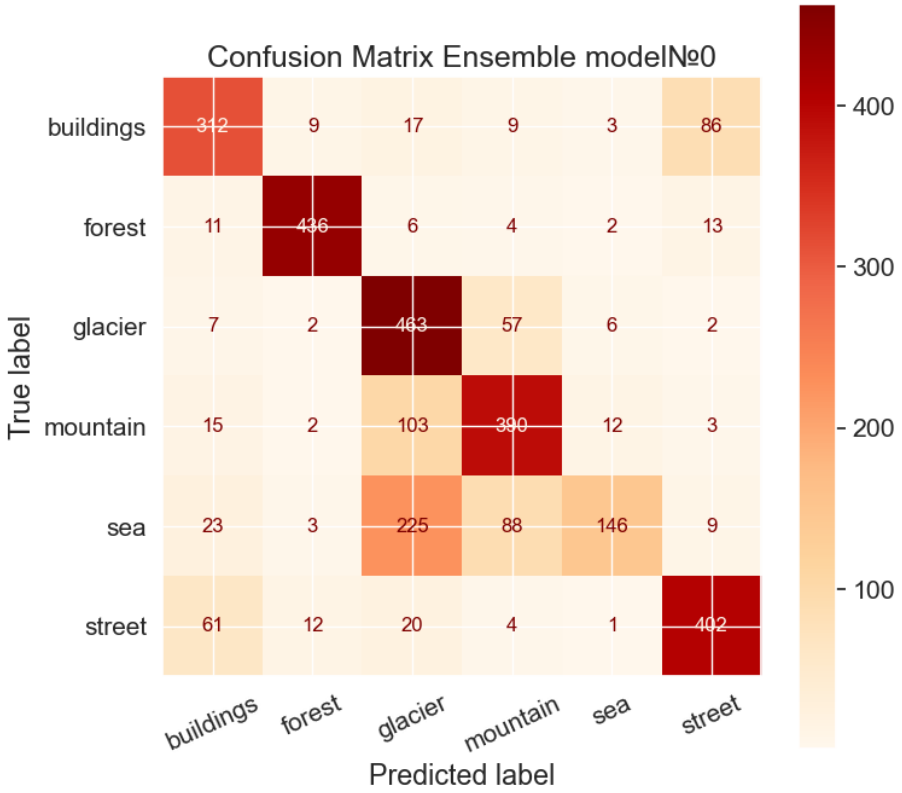
```
history=model.fit(train_images, train_labels, batch_size=128,epochs=8,validation_split=0.2)
```

```
Epoch 1/8
88/88 [==============================] - 163s 2s/step - loss: 44.4261 - accuracy: 0.3337 - val_loss: 1.3838 - val_accuracy: 0.4
435
Epoch 2/8
88/88 [==============================] - 170s 2s/step - loss: 1.1502 - accuracy: 0.5688 - val_loss: 1.2960 - val_accuracy: 0.50
02
Epoch 3/8
88/88 [==============================] - 170s 2s/step - loss: 0.7527 - accuracy: 0.7323 - val_loss: 1.4970 - val_accuracy: 0.50
45
Epoch 4/8
88/88 [==============================] - 156s 2s/step - loss: 0.4711 - accuracy: 0.8370 - val_loss: 1.6201 - val_accuracy: 0.51
55
Epoch 5/8
88/88 [==============================] - 155s 2s/step - loss: 0.2903 - accuracy: 0.9040 - val_loss: 1.8251 - val_accuracy: 0.49
48
Epoch 6/8
88/88 [==============================] - 149s 2s/step - loss: 0.2165 - accuracy: 0.9332 - val_loss: 2.0503 - val_accuracy: 0.52
26
Epoch 7/8
88/88 [==============================] - 168s 2s/step - loss: 0.1610 - accuracy: 0.9523 - val_loss: 2.3260 - val_accuracy: 0.50
23
Epoch 8/8
88/88 [==============================] - 196s 2s/step - loss: 0.1052 - accuracy: 0.9718 - val_loss: 2.6486 - val_accuracy: 0.52
```






```
93/93 [==============================] - 1s 4ms/step
              precision    recall  f1-score   support

           0       0.55      0.81      0.65       436
           1       0.96      0.75      0.84       472
           2       0.62      0.88      0.73       537
           3       0.72      0.57      0.64       525
           4       0.74      0.47      0.57       494
           5       0.74      0.67      0.71       500

    accuracy                           0.69      2964
   macro avg       0.72      0.69      0.69      2964
weighted avg       0.72      0.69      0.69      2964
```



Now we need to try to use Bayesian Optimization to find much more suitable hyperparameters.

3.3. Model with Bayesian Optimization

Let's define a function to our Bayesian Optimization.

```
def build_model(hp):
    model = Sequential()
    model.add(Input(shape=(150, 150, 3)))

    for i in range(hp.Int('num_blocks', 1, 2)):
        hp_padding=hp.Choice('padding_'+ str(i), values=['valid', 'same'])
        hp_filters=hp.Choice('filters_'+ str(i), values=[32, 64])
        model.add(Conv2D(hp_filters, (3, 3),
                         padding=hp_padding, activation='relu',
                         kernel_initializer='he_uniform',
                         input_shape=(150, 150, 3)))
        model.add(MaxPooling2D((2, 2)))
        model.add(Dropout(hp.Choice('dropout_'+ str(i), values=[0.0, 0.1, 0.2])))
    model.add(Flatten())

    hp_units = hp.Int('units', min_value=16, max_value=256, step=16)
    model.add(Dense(hp_units, activation='relu', kernel_initializer='he_uniform'))
    model.add(Dense(6,activation="softmax"))
    hp_learning_rate = hp.Choice('learning_rate', values=[1e-2, 1e-3])
    hp_optimizer=hp.Choice('Optimizer', values=['Adam', 'SGD'])

    if hp_optimizer == 'Adam':
        hp_learning_rate = hp.Choice('learning_rate', values=[1e-2, 1e-3])
    elif hp_optimizer == 'SGD':
        hp_learning_rate = hp.Choice('learning_rate', values=[1e-2, 1e-3])
        nesterov=True
        momentum=0.9
    model.compile(optimizer=hp_optimizer,loss='categorical_crossentropy', metrics=['accuracy'])
    return model
```

and run searching optimal parameters of our model for 100 trials and 30 epochs on each one:

```
#This part of code is not needed to run everytime.
#all values of necessary hyperparameters are stored in bestHP variable
#----------------
tuner_cnn = kt.tuners.BayesianOptimization(
    build_model,
    objective='val_loss',
    max_trials=100,
    directory='.',
    project_name='tuning-cnn')

tuner_cnn.search(train_x, train_y,
             validation_data= (val_x,val_y),
             epochs=30,
             callbacks=[tf.keras.callbacks.EarlyStopping(patience=2)])

print("________")
bestHP = tuner_cnn.get_best_hyperparameters(num_trials=1)[0]
print(bestHP)
```

```
Search: Running Trial #1

Value             |Best Value So Far |Hyperparameter
2                 |2                 |num_blocks
same              |same              |padding_0
64                |64                |filters_0
0                 |0                 |dropout_0
160               |160               |units
0.01              |0.01              |learning_rate
SGD               |SGD               |Optimizer
```

```
Search: Running Trial #4

Value             |Best Value So Far |Hyperparameter
1                 |2                 |num_blocks
valid             |same              |padding_0
64                |64                |filters_0
0.1               |0                 |dropout_0
160               |160               |units
0.01              |0.01              |learning_rate
Adam              |SGD               |Optimizer
same              |valid             |padding_1
32                |32                |filters_1
0.2               |0                 |dropout_1
```

```
Epoch 1/30
416/416 [==============================] - 14s 10ms/step - loss: 1.2850 - accuracy: 0.4988 - val_loss: 2.5408 - val_accuracy: 0.4229
Epoch 2/30
416/416 [==============================] - 3s 8ms/step - loss: 0.9169 - accuracy: 0.6544 - val_loss: 0.8693 - val_accuracy: 0.6757
Epoch 3/30
416/416 [==============================] - 3s 8ms/step - loss: 0.7172 - accuracy: 0.7350 - val_loss: 0.7027 - val_accuracy: 0.7557
Epoch 4/30
416/416 [==============================] - 3s 8ms/step - loss: 0.6068 - accuracy: 0.7767 - val_loss: 3.1966 - val_accuracy: 0.3843
Epoch 5/30
343/416 [=======================>......] - ETA: 0s - loss: 0.5764 - accuracy: 0.7940
```

Parameters search provided the following results of hyperparameters:

```
bestHP
```

```
{'num_blocks': 2,
 'padding_0': 'same',
 'filters_0': 32,
 'dropout_0': 0.0,
 'units': 256,
 'learning_rate': 0.001,
 'Optimizer': 'Adam',
 'padding_1': 'valid',
 'filters_1': 32,
 'dropout_1': 0.1}
```

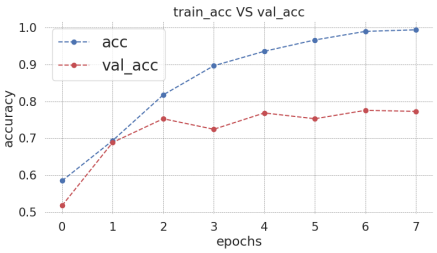
After finding optimal hyperparameters let’s do model fitting

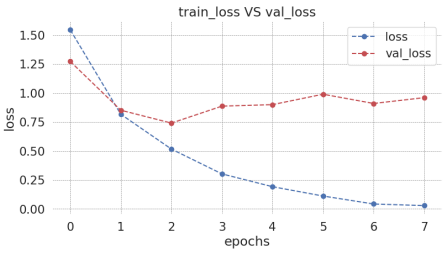
```
model_cnn = Sequential()
model_cnn.add(Input(shape=(150, 150, 3)))
for i in range(bestHP['num_blocks']):
  hp_padding=bestHP['padding_'+ str(i)]
  hp_filters=bestHP['filters_'+ str(i)]
  model_cnn.add(Conv2D(hp_filters, (3, 3), padding=hp_padding, activation='relu',
                       kernel_initializer='he_uniform', input_shape=(150, 150, 3)))
  model_cnn.add(MaxPooling2D((2, 2)))
  model_cnn.add(Dropout(bestHP['dropout_'+ str(i)]))
model_cnn.add(Flatten())
model_cnn.add(Dense(bestHP['units'], activation='relu',
                    kernel_initializer='he_uniform'))
model_cnn.add(Dense(6,activation="softmax"))
model_cnn.compile(optimizer=bestHP['Optimizer'],
                                              loss='categorical_crossentropy',
                                              metrics=['accuracy'])
print(model_cnn.summary())
history_cnn= model_cnn.fit(train_x, train_y, epochs=50, batch_size=32,
                           validation_data=(val_x, val_y),
                           callbacks=[erl_stop, mod_chk1, lr_rate])
```

```
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 conv2d_1 (Conv2D)            (None, 150, 150, 32)      896

 max_pooling2d_1 (MaxPooling  (None, 75, 75, 32)       0
 2D)

 dropout_1 (Dropout)          (None, 75, 75, 32)        0

 conv2d_2 (Conv2D)            (None, 73, 73, 32)        9248

 max_pooling2d_2 (MaxPooling  (None, 36, 36, 32)       0
 2D)

 dropout_2 (Dropout)          (None, 36, 36, 32)        0

 flatten_1 (Flatten)          (None, 41472)             0

 dense_2 (Dense)              (None, 256)               10617088

 dense_3 (Dense)              (None, 6)                 1542

=================================================================
Total params: 10,628,774
Trainable params: 10,628,774
Non-trainable params: 0
```

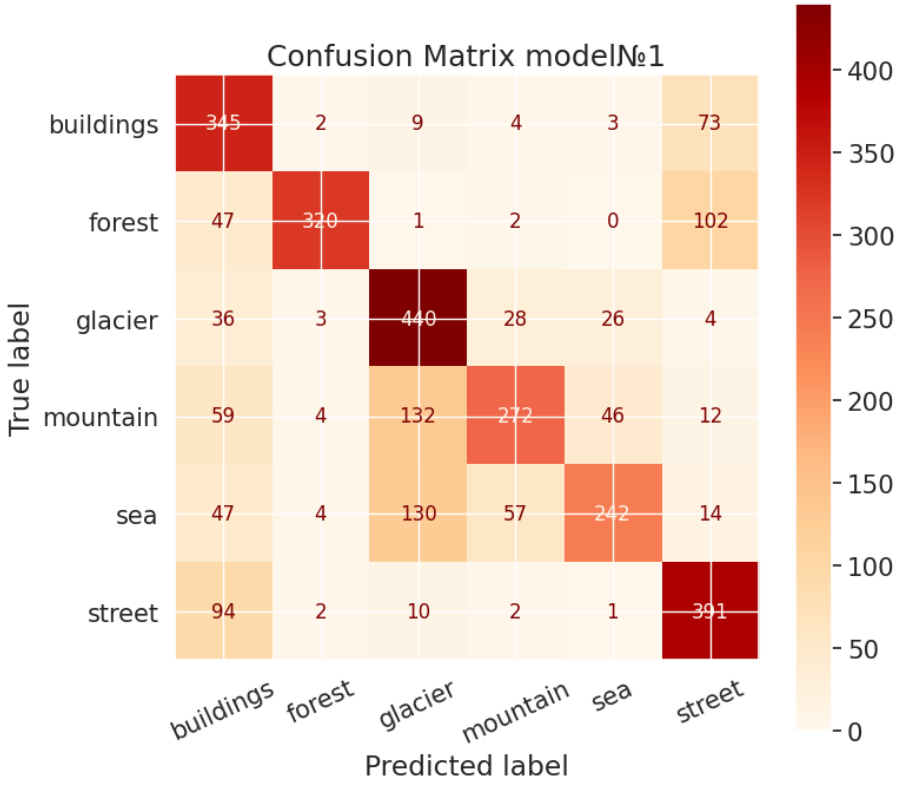
and here are our results:

```
None
Epoch 1/50
416/416 [==============================] - 6s 9ms/step - loss: 1.5469 - accuracy: 0.5844 - val_loss: 1.2737 - val_accuracy: 0.5171 - lr: 0.0010
Epoch 2/50
416/416 [==============================] - 3s 8ms/step - loss: 0.8177 - accuracy: 0.6936 - val_loss: 0.8525 - val_accuracy: 0.6886 - lr: 0.0010
Epoch 3/50
416/416 [==============================] - 3s 8ms/step - loss: 0.5171 - accuracy: 0.8176 - val_loss: 0.7414 - val_accuracy: 0.7529 - lr: 0.0010
Epoch 4/50
416/416 [==============================] - 3s 7ms/step - loss: 0.3028 - accuracy: 0.8967 - val_loss: 0.8879 - val_accuracy: 0.7243 - lr: 0.0010
Epoch 5/50
416/416 [==============================] - 3s 7ms/step - loss: 0.1922 - accuracy: 0.9360 - val_loss: 0.9008 - val_accuracy: 0.7686 - lr: 0.0010
Epoch 6/50
416/416 [==============================] - 3s 7ms/step - loss: 0.1116 - accuracy: 0.9667 - val_loss: 0.9907 - val_accuracy: 0.7529 - lr: 0.0010
Epoch 7/50
416/416 [==============================] - 3s 7ms/step - loss: 0.0428 - accuracy: 0.9901 - val_loss: 0.9108 - val_accuracy: 0.7757 - lr: 1.0000e-04
Epoch 8/50
416/416 [==============================] - 3s 8ms/step - loss: 0.0292 - accuracy: 0.9943 - val_loss: 0.9611 - val_accuracy: 0.7729 - lr: 1.0000e-04
```






```
93/93 [==============================] - 2s 15ms/step
              precision    recall  f1-score   support

           0       0.64      0.97      0.77       436
           1       0.99      0.90      0.94       472
           2       0.82      0.72      0.77       537
           3       0.72      0.85      0.78       525
           4       0.96      0.66      0.78       494
           5       0.83      0.73      0.78       500

    accuracy                           0.80      2964
   macro avg       0.83      0.81      0.80      2964
weighted avg       0.83      0.80      0.80      2964
```



Let’s do some comparison between previous model(at the left side) and model with Bayesian Optimization(right side):

```
93/93 [==============================] - 1s 4ms/step
              precision    recall  f1-score   support

           0       0.55      0.81      0.65       436
           1       0.96      0.75      0.84       472
           2       0.62      0.88      0.73       537
           3       0.72      0.57      0.64       525
           4       0.74      0.47      0.57       494
           5       0.74      0.67      0.71       500

    accuracy                           0.69      2964
   macro avg       0.72      0.69      0.69      2964
weighted avg       0.72      0.69      0.69      2964
```

without hyperparameters optimization

```
93/93 [==============================] - 2s 15ms/step
              precision    recall  f1-score   support

           0       0.64      0.97      0.77       436
           1       0.99      0.90      0.94       472
           2       0.82      0.72      0.77       537
           3       0.72      0.85      0.78       525
           4       0.96      0.66      0.78       494
           5       0.83      0.73      0.78       500

    accuracy                           0.80      2964
   macro avg       0.83      0.81      0.80      2964
weighted avg       0.83      0.80      0.80      2964
```

with Bayesian Optimization

As we can see, models fitted with Hyperparameters selected by Bayesian Optimization have higher main classification metrics. In the future, when forming our Ensemble model, one of them will be just the same on the base Bayesian Optimisation. Let’s save this model as file model1.h5. Now let’s prepare a model based on VGG16.

3.4. VGG16 model

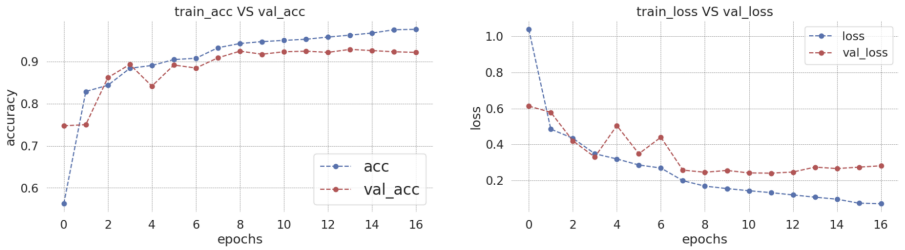
Here are code where we preparing our model:

```
from tensorflow.keras.applications import VGG16
from tensorflow.keras.models import Model
from tensorflow.keras.layers import MaxPooling2D, GlobalAveragePooling2D

base_model=VGG16(
    input_shape=(150,150,3),
    weights='imagenet',
    include_top=False
    )

for layer in base_model.layers[:10]:
    layer.trainable=False

x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(256, activation='relu')(x)
x = Dropout(0.2)(x)
predictions = Dense(6, activation='softmax')(x)
model2 = Model(inputs=base_model.inputs, outputs=predictions)
```

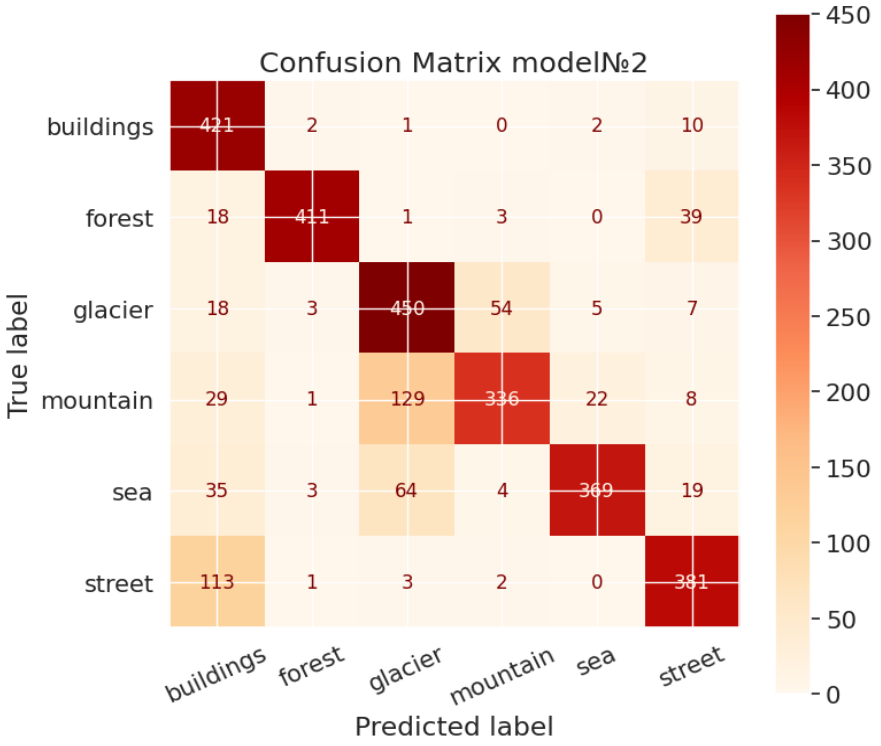
```
model2.compile(optimizer=bestHP['Optimizer'],
               loss='categorical_crossentropy', metrics=['accuracy'])
```

```
history2= model2.fit(train_x, train_y, epochs=50, batch_size=32,
                     validation_data=(val_x, val_y),
                     callbacks=[erl_stop, mod_chk2, lr_rate])
```

```
Epoch 4/50
416/416 [==============================] - 7s 17ms/step - loss: 0.3487 - accuracy: 0.8837 - val_loss: 0.3303 - val_accuracy: 0.8929
Epoch 5/50
416/416 [==============================] - 7s 16ms/step - loss: 0.3197 - accuracy: 0.8904 - val_loss: 0.5049 - val_accuracy: 0.8414
Epoch 6/50
416/416 [==============================] - 7s 16ms/step - loss: 0.2862 - accuracy: 0.9044 - val_loss: 0.3486 - val_accuracy: 0.8914
Epoch 7/50
416/416 [==============================] - 6s 16ms/step - loss: 0.2696 - accuracy: 0.9075 - val_loss: 0.4401 - val_accuracy: 0.8843
Epoch 8/50
416/416 [==============================] - 7s 17ms/step - loss: 0.1992 - accuracy: 0.9325 - val_loss: 0.2578 - val_accuracy: 0.9086
Epoch 9/50
416/416 [==============================] - 7s 17ms/step - loss: 0.1690 - accuracy: 0.9425 - val_loss: 0.2459 - val_accuracy: 0.9243
Epoch 10/50
416/416 [==============================] - 6s 16ms/step - loss: 0.1555 - accuracy: 0.9470 - val_loss: 0.2561 - val_accuracy: 0.9171
Epoch 11/50
416/416 [==============================] - 7s 17ms/step - loss: 0.1435 - accuracy: 0.9499 - val_loss: 0.2423 - val_accuracy: 0.9229
Epoch 12/50
416/416 [==============================] - 7s 16ms/step - loss: 0.1328 - accuracy: 0.9526 - val_loss: 0.2408 - val_accuracy: 0.9243
Epoch 13/50
416/416 [==============================] - 6s 16ms/step - loss: 0.1201 - accuracy: 0.9578 - val_loss: 0.2473 - val_accuracy: 0.9214
Epoch 14/50
416/416 [==============================] - 6s 16ms/step - loss: 0.1076 - accuracy: 0.9624 - val_loss: 0.2745 - val_accuracy: 0.9286
Epoch 15/50
416/416 [==============================] - 6s 16ms/step - loss: 0.0964 - accuracy: 0.9673 - val_loss: 0.2664 - val_accuracy: 0.9257
Epoch 16/50
416/416 [==============================] - 6s 16ms/step - loss: 0.0743 - accuracy: 0.9752 - val_loss: 0.2744 - val_accuracy: 0.9229
Epoch 17/50
416/416 [==============================] - 7s 16ms/step - loss: 0.0715 - accuracy: 0.9762 - val_loss: 0.2819 - val_accuracy: 0.9214
```



	precision	recall	f1-score	support
0	0.64	0.96	0.77	436
1	0.98	0.93	0.96	472
2	0.81	0.77	0.79	537
3	0.81	0.77	0.79	525
4	0.96	0.72	0.82	494
5	0.81	0.79	0.80	500
accuracy			0.82	2964
macro avg	0.84	0.82	0.82	2964
weighted avg	0.84	0.82	0.82	2964



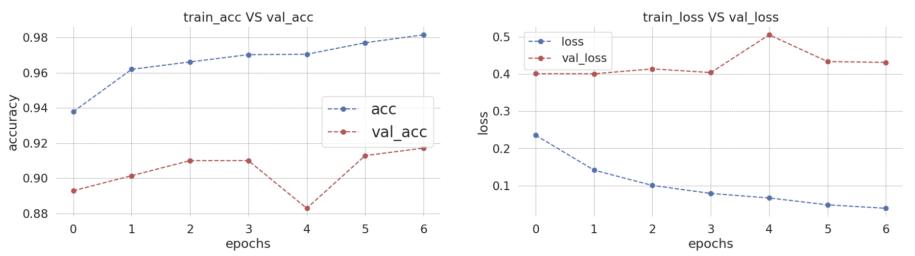
Let’s save this model as file model2.h5.

3.5. Ensemble method

We will combine two our models and see what results this new model will show to us:

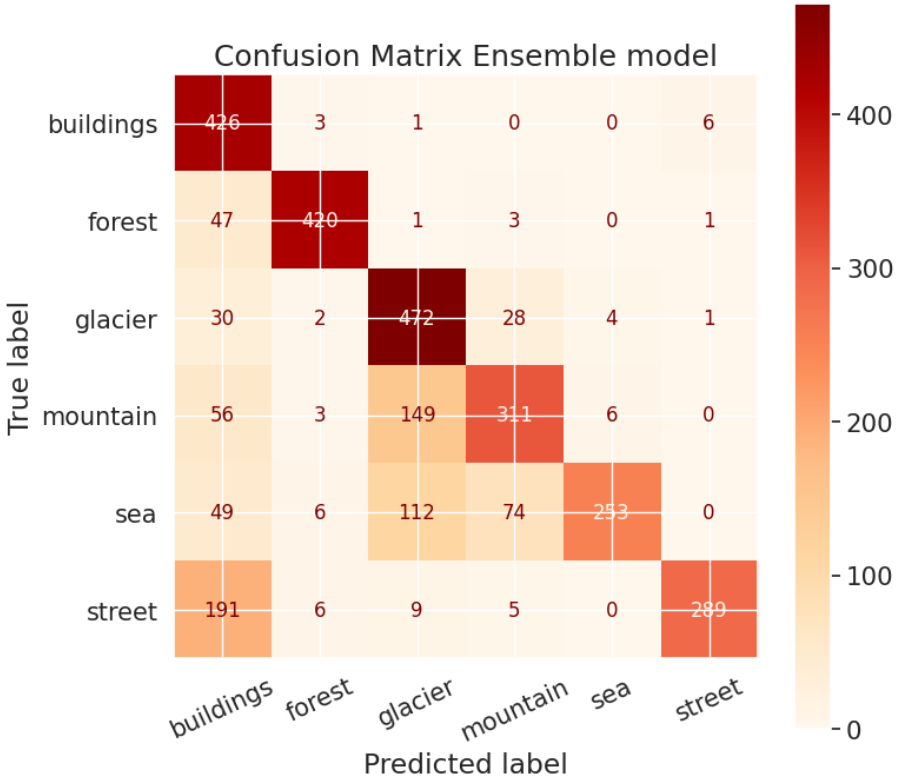
```
from tensorflow.keras.models import Model, load_model
from tensorflow.keras.layers import Input, Average
#look at the logs correct model names!!!
model_1=load_model(work_path + '/model1.h5')
model_1 = Model(inputs=model_1.inputs,
                outputs=model_1.outputs,
                name='model1')

model_2=load_model(work_path + '/model2.h5')
model_2 = Model(inputs=model_2.inputs,
                outputs=model_2.outputs,
                name='model2')

models = [model_1, model_2]
model_input = Input(shape=(150, 150, 3))
model_outputs = [model(model_input) for model in models]
ensemble_output = Average()(model_outputs)
ensemble_model = Model(inputs=model_input, outputs=ensemble_output,
                       name='ensemble')

ensemble_model.compile(optimizer=bestHP['Optimizer'],
                       loss='categorical_crossentropy', metrics=['accuracy'])
```

```
[25] history=ensemble_model.fit(train_x, train_y, epochs=50, batch_size=256, validation_data=(val_x, val_y), callbacks=[erl_stop, mod_chk
Epoch 1/50
52/52 [==============================] - 43s 431ms/step - loss: 0.2354 - accuracy: 0.9378 - val_loss: 0.4009 - val_accuracy: 0.8929
Epoch 2/50
52/52 [==============================] - 7s 135ms/step - loss: 0.1416 - accuracy: 0.9620 - val_loss: 0.4006 - val_accuracy: 0.9014 -
Epoch 3/50
52/52 [==============================] - 6s 122ms/step - loss: 0.1005 - accuracy: 0.9662 - val_loss: 0.4137 - val_accuracy: 0.9100 -
Epoch 4/50
52/52 [==============================] - 6s 123ms/step - loss: 0.0790 - accuracy: 0.9703 - val_loss: 0.4039 - val_accuracy: 0.9100 -
Epoch 5/50
52/52 [==============================] - 6s 124ms/step - loss: 0.0668 - accuracy: 0.9706 - val_loss: 0.5053 - val_accuracy: 0.8829 -
Epoch 6/50
52/52 [==============================] - 6s 123ms/step - loss: 0.0483 - accuracy: 0.9771 - val_loss: 0.4334 - val_accuracy: 0.9129 -
Epoch 7/50
52/52 [==============================] - 6s 125ms/step - loss: 0.0390 - accuracy: 0.9816 - val_loss: 0.4312 - val_accuracy: 0.9171 -
```



and here comes our metrics:

	precision	recall	f1-score	support
0	0.59	0.97	0.73	436
1	0.98	0.87	0.92	472
2	0.69	0.88	0.77	537
3	0.73	0.75	0.74	525
4	0.93	0.54	0.68	494
5	0.90	0.56	0.69	500
accuracy			0.76	2964
macro avg	0.80	0.76	0.76	2964
weighted avg	0.80	0.76	0.76	2964



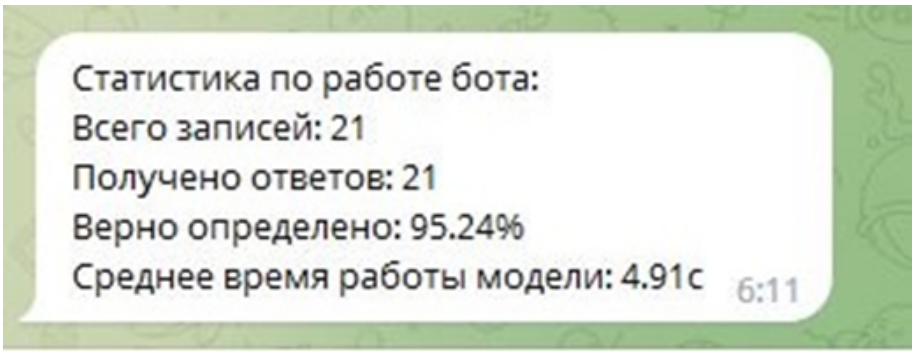
3.6. Server

3.6.1. Choice of technical platform

As the most affordable and sufficient option, a VPS server with the following characteristics was chosen:

- CPU 1 x 3 GHz
- RAM 1 Gb
- NVMe 15 Gb
- Ethernet 100 Mbit/s

As tests have shown, this configuration (with additional tuning and optimization of the model) is sufficient for the successful operation of all necessary services. The average image recognition time is 4-5 seconds.



3.6.2. Software setup

Our project is based on the image of the popular operating system Ubuntu 22.04.

To ensure high reliability and sufficiency of resources, an additional swap was configured in the 1 GB.

Also, additional applications and libraries were installed for the service to work:

- Python 3.10.6
- python3-pip
- systemd
- tensorflow
- opencv-python-headless
- python-telegram-bot
- db-sqlite3

3.6.3. Main control flow

The main.py contains:

- set of necessary imports and parameters

recognizer and ***c_sql*** – model and db-control libs

datetime and ***os*** – technical libs

telegram – main telegram-bot lib

- ***main*** control thread - there is the database initialization and setup telegram command handlers.
- a number of command procedure

§ ***start*** – start introduce dialog with welcome message

§ ***conv_handler*** – the main process that receives the photo for classification

§ ***stat*** – technical command, sending bot statistic

§ ***logs*** - technical command, sending log file

§ ***clrstat*** - technical command, clearing bot statistic

3.6.4. c_sql - db-control lib

contains the following functions:

- **init_db** - In accordance with the principle of idempotence, this function creates a database from scratch and sets up the necessary tables for the project.
- **write_result** - This function records the parameters of the user communication process and the image classification process to collect further analytical metrics.
- **get_stat** - This function scans the database and returns statistical information on system operation.
- **clr_stat** - This function backs up the data on the statistics of the application and resets the current table of indicators

3.6.5. recognizer – model lib

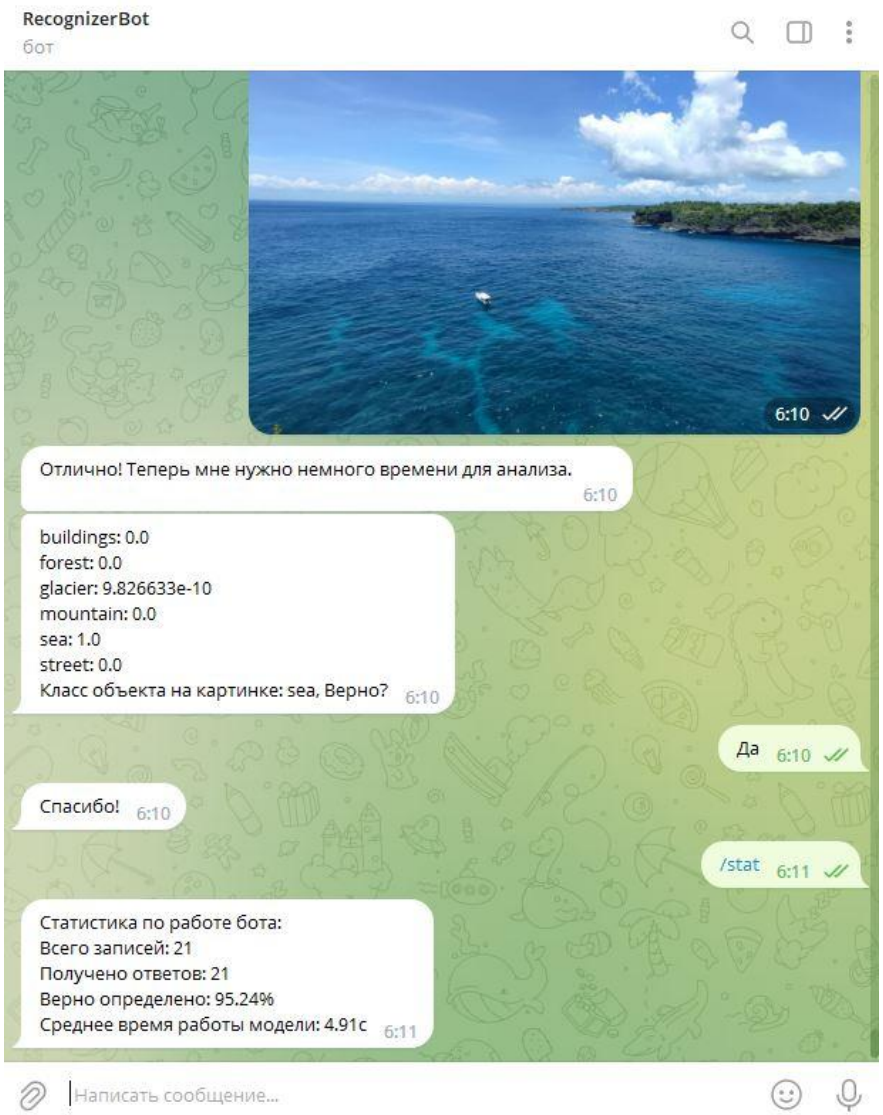
contains the following functions:

- **start_recognition**- control function responsible for the operation of the image processing and classification algorithm
- **get_img** - performs image preprocessing using the cv2 library
- **recognize_img** - applies the model to the resulting image, collects and processes the results

4. Conclusion

This project was very challenging and interesting. We have gained understanding and experience of using NN for image classification. We also gained experience in development in Python, analysis of initial data for NN, new knowledge in the field of high-level design (UML, IDEF0, Swim Lane), experience in teamwork and project management.

The main problem in this project was to determine the hyperparameters of the classifier models, choose importance features, and optimization strategies. The applied methods and algorithms for constructing CNN show that the goal was achieved, and they say that it is necessary to select and use various models and methods for their implementation to solve classification problems.



5. Appendices

5.1. Knowledgebase

5.2. GitHub

https://github.com/WNick-main/Recognizer/

5.3. Google Colab

https://colab.research.google.com/drive/1eWAliXqz70VBQ2NAlZHc8KBsruVawH70#scrollTo=Q-P6BNElgCFm

6. References

1. Dataset, https://www.kaggle.com/datasets/puneet6060/intel-image-classification
2. Convolutional Neural Networks, Explained, https://towardsdatascience.com/convolutional-neural-networks-explained-9cc5188c4939
3. Convolutional neural network, https://en.wikipedia.org/wiki/Convolutional_neural_network
4. Bayesian optimization, https://en.wikipedia.org/wiki/Bayesian_optimization#:~:text=Bayesian%20optimization%20is%20a%20sequential,expensive%2Dto%2Devaluate%20functions
5. Bayesian optimization, https://towardsdatascience.com/bayesian-optimization-concept-explained-in-layman-terms-1d2bcdeaf12f
6. Ensemble Models: What Are They and When Should You Use Them? https://builtin.com/machine-learning/ensemble-model
7. Digital twin, https://future2day.ru/texnologiya-cifrovyx-dvojnikov/
8. Digital Twin in Industry, Tao F, Zhang H, Liu A, State-of-the-Art. IEEE Trans Industry Inform. 2019; 15(4): 2405–2415.,https://ieeexplore.ieee.org/document/8477101
9. IDEF0, https://en.wikipedia.org/wiki/IDEF0
10. IDEF0, https://www.cfin.ru/vernikov/idef/idef0.shtml
11. Swimlane, https://ru.wikipedia.org/wiki/Swim_lane
12. Swimlane, https://www.lucidchart.com/pages/tutorial/swimlane-diagram
13. Swimlane, https://www.mindmanager.com/en/features/swim-lane-diagram
14. Confusion matrix, https://www.kaggle.com/code/valentynsichkar/confusion-matrix-for-image-classification/notebook
15. Classification Report in Machine Learning,

https://thecleverprogrammer.com/2021/07/07/classification-report-in-machine-learning/#:~:text=A%20classification%20report%20is%20a,of%20your%20trained%20classification%20model.

16. Accuracy, Precision, Recall or F1?, https://towardsdatascience.com/accuracy-precision-recall-or-f1-331fb37c5cb9

17. Confusion Matrix, Accuracy, Precision, Recall, F1 Score https://medium.com/analytics-vidhya/confusion-matrix-accuracy-precision-recall-f1-score-ade299cf63cd#:~:text=F1%20Score%20becomes%201%20only,0.857%20%2B%200.75)%20%3D%200.799.