Міністерство освіти і науки, молоді та спорту України
Львівський національний університет імені Івана Франка
Факультет прикладної математики та інформатики
Кафедра обчислювальної математики

Курсова робота

на тему:

"Глибокі нейронні мережі для задач виявлення об'єктів"

Виконав:
студент II курсу групи ПМпМ-22с
напрямку підготовки (спеціальності)
113 – "Прикладна математика"
Бугрій Б.О.

Науковий керівник:
доц. Музичук Ю.А.

Львів - 2022

Contents

Introduction

In the last few decades humanity, thanks to smart systems of artificial intelligence, has succeeded in achieving impressive results in many areas of everyday life. Vivid examples of that are cars able to overcome difficult routes without human intervention, software that gives accurate diagnoses to patients based on medical test results and other detailed information, applications for speech replica and recognition, and others.

Machine Learning models made a significant impact in the areas of logistics, medicine, security, entertainment, and so on. The most common tasks performed by AI today are related to Computer Vision. It is the area of research focused on the ability of algorithms to perform computations and solve nontrivial problems based on visual data, like images or video streams. Since this problem appeared the first time, a lot of algorithms and approaches have been developed [1].

A significant part of these breakthroughs is due to the rapid development of Deep Neural Networks, which, compared to other machine learning algorithms, are able to show shocking results based on large data sets, sometimes even surpassing humans. But tasks like counting the number of people on the image, or recognizing a road sign are difficult and hide a number of challenges even for state-of-the-art neural networks [2].

In this work, we will get acquainted with common approaches to the problem of object detection and will build a model example to understand the main principles of the field and be able to perform deeper research in the future. We are going to experiment with the type of algorithms called Convolutional Neural Networks, which are able to show high performance in detecting complex objects and shapes. We are going to define performance metrics and evaluate our own model, to see how effective it is in solving the problem.

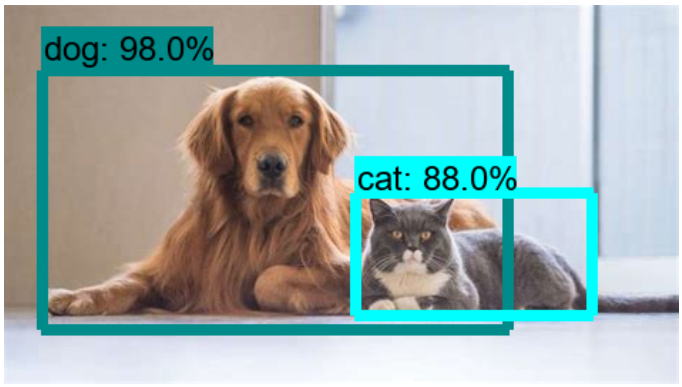
1 Description of the problem

1.1 Problem types

To understand what is object detection, let us consider the definitions of the following Computer Vision tasks:

- **Object classification**, or multiclass classification is a problem in which the image consists of one main object and, usually, a background. The object belongs to one of two or more predefined classes. The model, given the input image, should return a class of the object from the image.
- **Object localization** is the task that aims to locate the target object within the image or video. There might be multiple instances of the object or even different classes of objects on the same image. The model, given the input image, should return the location of objects on the image in the specified formats, like coordinates of the center of the object or somehow defined area which contains the targets.
- **Object detection** has a goal of not only finding objects from a chosen set of classes on the image but also figuring out what these objects are. The model, given the input image, should return the location and class label of every specific object from the subject area.

So, as follows from the above descriptions, an object detection problem is a combination of object classification and object localization. It has its own challenges, which have to be addressed. For example, on one image there might be multiple objects of different shapes and sizes, viewed from angles, etc. This problem is also costly in terms of computational resources it requires since usually we should use complex ML algorithms with a lot of parameters.



Picture 1. A typical output of an object detection algorithm [3]. The detected object is surrounded by the box with the probabilistic label assigned.

1.2 Format of the data

To train the object detection model we need several types of inputs. First and most obviously we need an image of where the object is located. Each image should have a fixed height and width. Popular is the RGB image format, with three channels – red, green, and blue. Values of each channel are integers in the range from 0 to 255. To teach the model how to locate the object of the image first we need a way to specify the object's location. Each object has a different shape and size, therefore to solve this issue *bounding boxes* are used.

The bounding box is a rectangular area that fully contains the object. In this way we have a general idea about object location it the specific domain. For the sake of machine learning, square or rectangular bounding boxes are used. To define such a bounding box is enough to have the coordinates of two opposite corners, for example, the top-left and bottom-right corners or the center coordinates with box height and width.

The last piece of information we need for the model training and evaluation is the label of the class in each bounding box. After training of the machine learning model, the image itself is the input parameter, while the bounding boxes and labels – the outputs.

It is usually difficult to gather enough input data for the model since each instance should be manually labeled, which requires a lot of human effort. Therefore it is common to automatically generate additional images based on a few labeled examples to make this process cheaper.

1.3 Intersection over Union

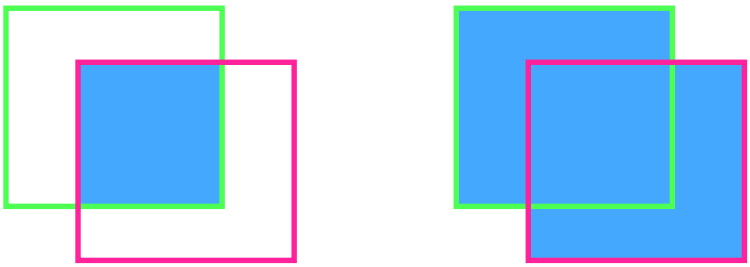
To be able to compare the performance of our models and tell how good they are at detecting objects on the images, we need to have a defined metric. We are trying to predict the bounding box which contains the object. But it is usually expected that the predicted and ground-truth boxes will not match. So usual metrics, like accuracy can't be applied, since it is designed for a completely different type of problems.

It is hard to determine, which prediction is better. It is logical to consider the area of the predicted bounding box which is covering the ground-truth region. But what if it just covers most of the image? Therefore another quantitative measure is used to compare true data with the results of the predictions, called *IoU*, *Intersection over Union*, which is computed by the following formula:

$$IoU = \frac{B_1 \cap B_2}{B_1 \cup B_2} \tag{1.1}$$

Basically, it is the Jaccard index of the two sets. It is obvious that values of this metric differ from 0 (no overlapping area at all) to 1 (exact match). So, the greater

the region of overlap, the greater the IOU. You can see the visual representation of the parts of this formula in the picture below.



$B_1 \cap B_2$ $B_1 \cup B_2$

Now we are able to calculate the average of these metrics over the batch of data or the whole training set to easily measure models' performance.

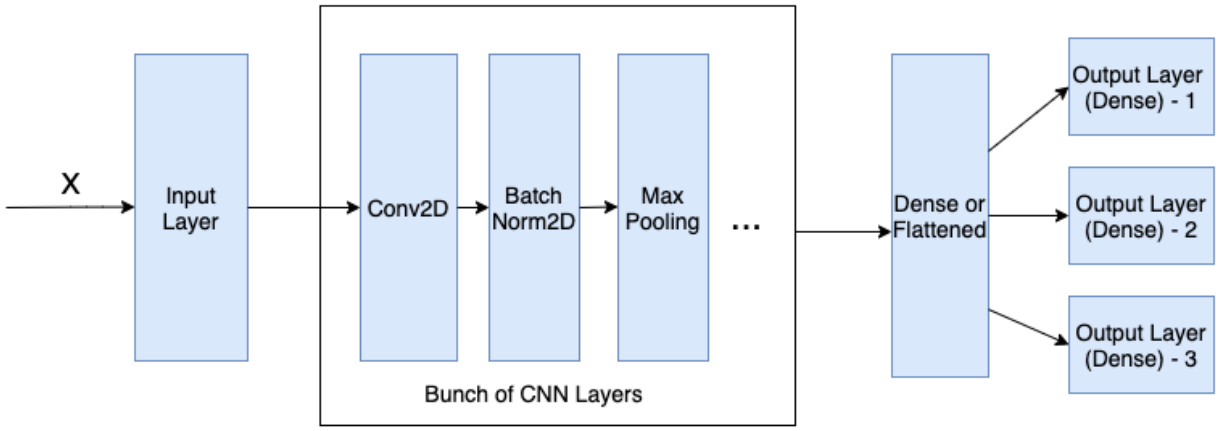
2 Convolutional neural networks

2.1 Multi-output neural network. Concept and structure

Convolutional Neural Networks are good at solving object detection problems. Usually, deep CNN consist not only of convolution layers. It is a complex structure of components with different purposes.

To detect the enemy vehicles we, basically, have to solve two problems with only one neural network. The network should detect the tank on the image and classify it. The key is that by using convolutional neural networks, we are able to build a feature extractor, which further we call the backbone network. It is a multilayer structure whose main purpose is to extract features from the image. It is embedded within the main network's architecture.

This network is used to encode the input into a certain feature representation. Then multiple layers of different purposes can be attached on top of the backbone network to perform different tasks, like binary classification and regression.

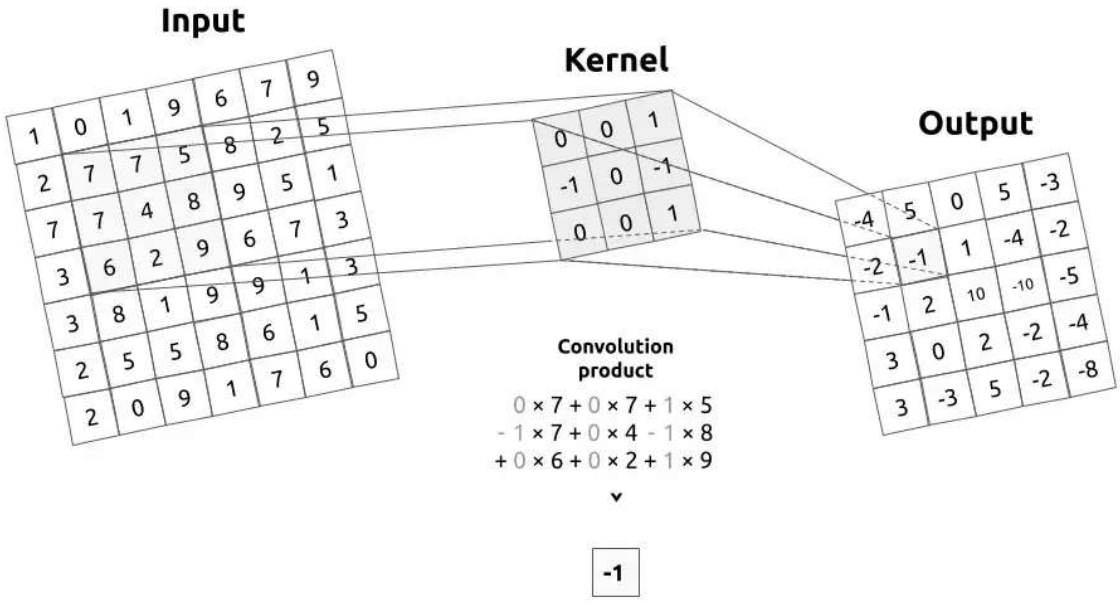


Picture 2. Example of the multi-output CNN architecture [7].

Process of forward and backward propagation for this architecture is similar to what we had for DNN in [4]. We will discuss what is the convolution layer and how it works in the next section.

2.2 Convolutional layer

The CNNs is able to successfully capture the spatial dependencies in an image through the application of relevant filters. A convolutional layer is the main building block of a CNN. It contains a set of filters (or kernels), parameters of which are tuned throughout the training process. The size of the filters is usually smaller than the actual image. We use convolution operation between each filter and the part of the image to create an activation map.



Picture 3. The example of convolution operation [6].

Then the activation function is used on the outputs of the convolution operation.

2.3 Dense layer

In our previous works [4], [5] we studied neural networks build completely of dense layers. The main issue of this architecture itself is that the data is fed in the format of 1-dimensional vectors. It means that the model does not pay attention to the spatial patterns of the data, and each pixel is treated independently. Therefore, networks constructed only of dense layers are having a hard time recognizing general patterns for the task of image detection, where pixel location is the most important feature. It is also very costly in terms of computations required since it has a huge number of trainable parameters.

But it still suits well the role of the last layers of the model, to combine all features of previous layers together and produce the results in the desired format.

2.4 Batch normalization

We already know that normalizing the inputs is a powerful tool to speed up the model training process and make predictions better. But we can use it not only for the input data. We can also normalize the output of each layer of the network.

This approach is effective and widely used [9]. It allows to significantly boost the model performance and save some time during both forward and backward paths. It makes the task of finding the right gradients easier, so we are able to find the optimal model faster.

To give some memory for the normalization process, we use a momentum-like algorithm [11] with parameters, distinct for each batch normalization layer. Batch

normalization applies a transformation that maintains the mean output close to 0 and the output standard deviation close to 1.

During training, the layer normalizes its output using the mean and standard deviation of the current batch of inputs. On the other hand, during the prediction process, the layer normalizes its output using a moving average of the mean and standard deviation of the batches it has seen during training.

2.5 Max pooling

The pooling layer is used to reduce the dimensions of the feature maps [8]. Thus, it reduces the number of parameters to learn and the amount of computation performed in the network.

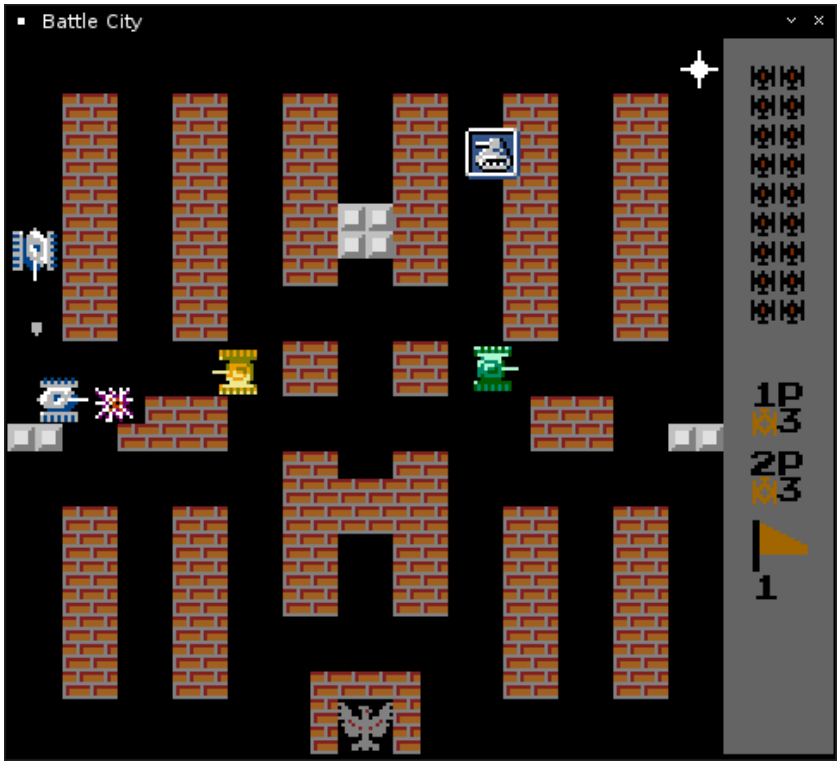
Max pooling downsamples the input along its height and width by taking the maximum value over an input window for each channel of the input. The layer summarises the features present in a region of the feature map generated by a convolution layer. So, further operations are performed on summarised features instead of precisely positioned features generated by the convolution layer.

This approach not only reduces the number of computational resources needed for training and making predictions. It also makes the model more robust to the noise and harmful perturbations of the input data.

3 The data set. Enemies detection.

As we mentioned before, getting data for ML model training might be challenging and expensive. So we have decided to build our model for the task, an environment for which can be easily reproduced to be able to generate enough training samples.

The computer game "Battle City"from the early 80s can be considered as a rough model of the battlefield viewed from above. At that time the game was very popular. The main goal of the gaming process – protect the base and destroy the enemies. You can see an example in the following picture.



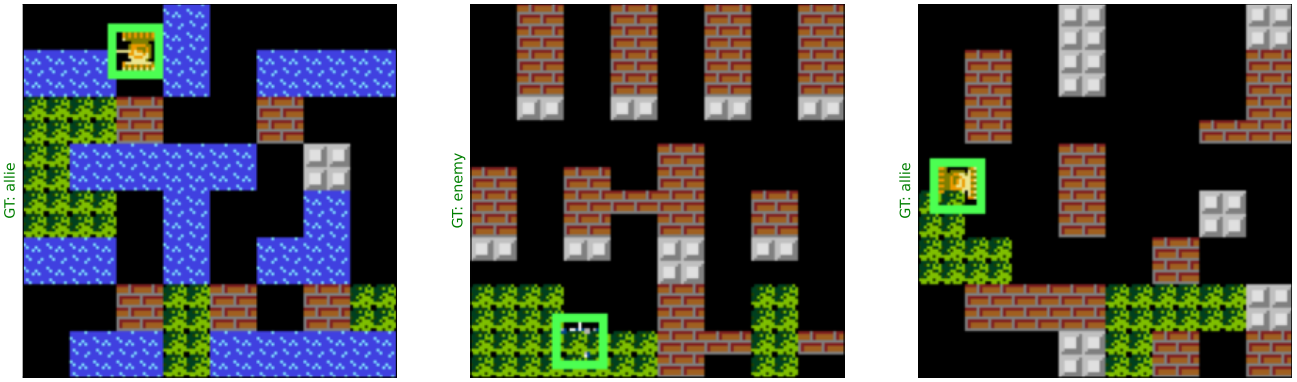
Picture 4. Gameplay example from the "Battle City".

The textures and the Python implementation of the game itself are available to the public [10], and we will use them to generate data for our machine learning model. The game has 35 unique levels each with its own landscape, composed of bricks, water, bushes, and ice. There also are several types of armored vehicles, which we split into two classes, allies and enemies, as shown in the example below.

Picture 5. Allies (first two) and enemies.

Using all that we have built a utility to randomly place the tank on the given part of the map and return the image itself in the RGB format with two labels.

The first label is the class of the tank placed on the image, allie or enemy. The second label – the coordinates of the top-left corner of the bounding box. We do not need the coordinates of the bottom-right corner since all the objects have the same size and ground-truth bounding boxes are squares. Here are a couple of examples:



Picture 6. Examples of images with ground-truth bounding boxes.

The ground-truth bounding boxes are shown in green. You can also see the label below each image. That's a rough representation of the data we use for the training of the neural network.

4 Model for enemies detection

4.1 Architecture

Now, having a clear understanding of the problem we are trying to solve, the algorithms specifics and approaches, we are able to start hyperparameters' tuning for the current problem. We need a model deep enough to represent the complex dependencies of the data. We have chosen the number of layers and amount of filters empirically, based on the range of experiments.

The backbone network has five convolutional layers, each one of which is followed by the batch normalization and max pooling layers. We use a fixed kernel size equal to 3, but the number of filters is gradually increased with depth, from 4 to 64. Max pulling reduces output dimensions in half each time. The last two layers of this feature extractor are flatten and dense layers respectively. All the layers mentioned before use relu [12] as an activation function. A detailed description of the architecture can be found in Appendix 1.

After that, we attach two independent dense layers, that are going to produce the final predictions. The one for the classification task uses softmax [4] as an activation since we would like to know the probabilities of the image belonging to a certain class. Another one uses no activation function at all. The output dimension of this layer equals to two, so it returns the coordinates of the top-left corner of the bounding box (all the bounding boxes for our task have fixed size).

To get an idea about the model's complexity, in the following table we bring on the number of model parameters.

Total	92,028
Trainable	91,780
Non-trainable	248

Table 1. The number of the model parameters.

4.2 Training process

During the training process, first, we have to decide which loss function to use for the performance evaluation. For class outputs, we choose the categorical cross-entropy function, which is common for classification tasks. On the other hand, for the coordinates' output, we use simple mean square error loss.

During the backward path, we use the ADAM optimizer [11] instead of the stochastic gradient descent method. Its results are generally better than other optimization algorithms, have faster computation time, and require fewer parameters for tuning. That is thanks to the ability of this algorithm to scale the gradients and take bigger steps during training.

Even while using an ADAM optimizer, the learning rate is also an important parameter. If we select the value too big, the model can be fast on the first steps,

but won't be able to achieve an optimal solution. For the learning rate too small, the situation is the opposite. Therefore we use not fixed but adaptive learning rate. The ADAM starts with value $\alpha = 0.01$ and scales down every 30 epochs with a scale factor equal to 0.2. It allows us to take bigger steps at the beginning and be more precise closer to the end of the training.

Since we are able to generate an almost endless stream of input data, we do not have to create test or train sets. We can use new images at each step of the training process. Here one more hyperparameter to adjust is the batch size. It is the input images at each step of the optimization process. The smaller the batch size – the easier the computations are and therefore we are able to perform each step faster. In this way, the model will already make some progress after processing a small amount of data. But this also means that gradients may change significantly during the process. To avoid this we feed the model 64 images at a time.

During 100 epochs, having 100 steps per epoch and batch size equal to 64, we can easily evaluate that during training our model is able to see up to 640 000 images. But it is not always required to process this amount of data. We implemented an early stopping of the training process in case the neural network stops making any progress during the last 10 epochs.

It is worth mentioning that all the values described above were selected as a result of numerous experiments and model tuning.

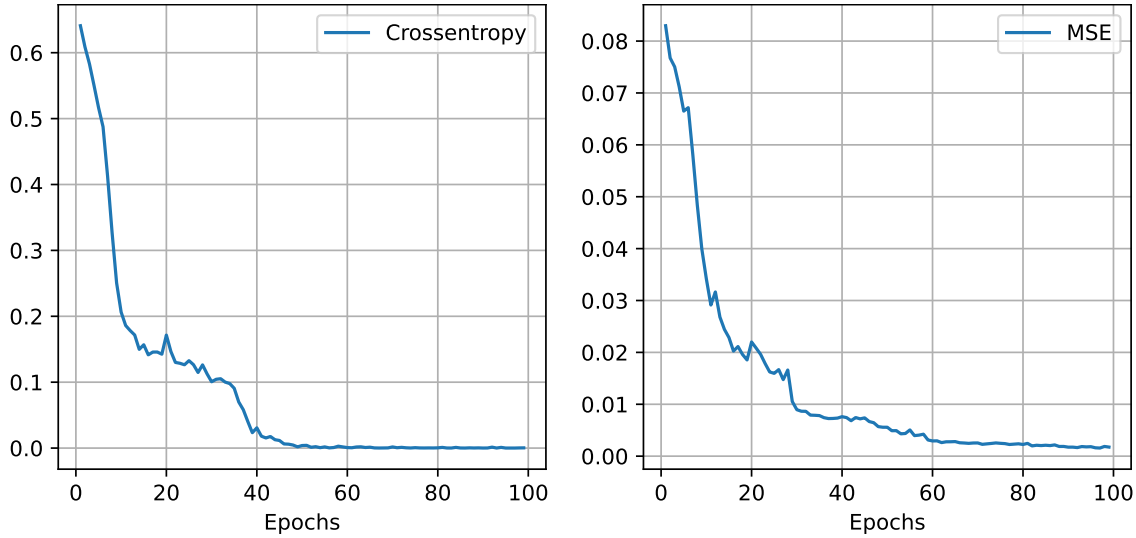
4.3 Evaluation

In this section, we are going to evaluate the model from the previous chapter. First, let us take a look at a couple of predictions' examples.

Picture 7. This picture shows the model output examples. The green square is the ground-truth object location, while the purple one – predicted bounding box.

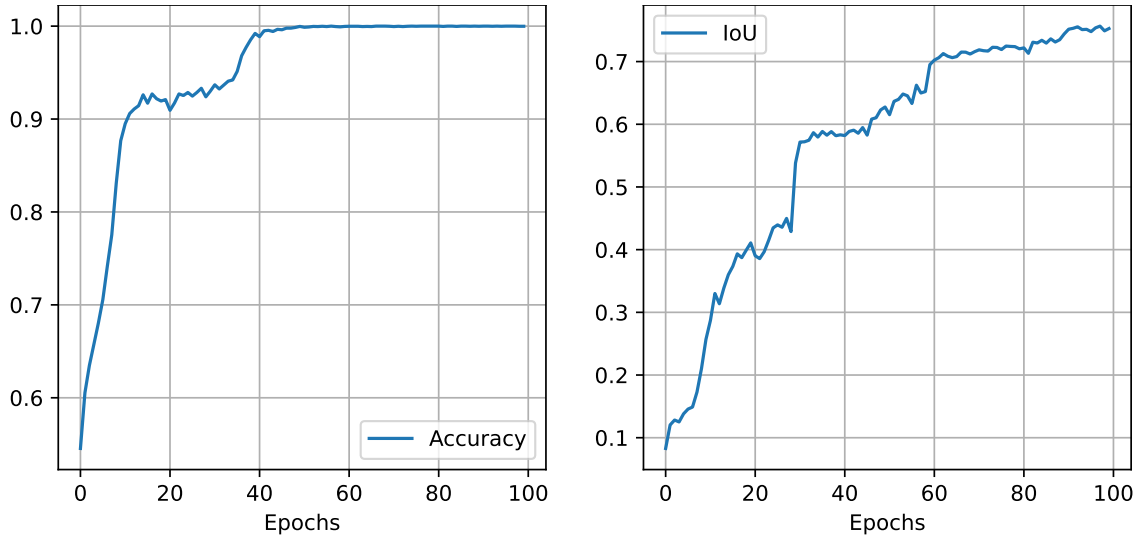
As we can see, the model is able to perform the task pretty well. The only thing it struggles with are tanks, masked in bushes, but even though the network is able to find an approximate location.

Now let us address the historical data about model training to see how the values of the loss functions change during the training process:



Picture 8. Evaluation of outputs of both output layers during training.

The next plot displays the accuracy of our model in assigning allie/enemy labels (left graph) and average intersection over union metric for predicted bounding boxes.



Picture 9. Model performance during training.

We achieved impressive results in terms of classification accuracy – 99%. The value of IoU is also high – 75%, and it means that our model suits data well and is the reasonable and effective solution to the defined problem.

5 Conclusions

Object detection is a far more difficult and complex task compared to image classification. In this work, we got acquainted with the problem of object detection and related issues. We considered using Convolutional Neural Networks for finding its solution. We also described which data is needed to build the machine learning model and which metrics are used to evaluate the model's performance.

On the specific example, we were able to verify that CNN can easily solve such kinds of problems. Even though the model struggles a bit with detecting the exact tanks' location in the bushes, it is able to find at least approximate locations and classify armored vehicles as allies or enemies extremely accurately.

To boost the performance and improve other properties of our neural network we used a variety of technics. During training, the network parameters were optimized with the ADAM algorithm to scale the gradients. The adaptive learning rate was used to move faster in the beginning, but to do smaller steps later. A batch normalization layer was used to scale the intermediate outputs. Max pooling reduced the number of inputs and trainable parameters in deeper layers. In the end, we built a fully functional neural network for enemy recognition.

In general, this area of research is an interesting and actual. New algorithms and approaches can be developed to solve more complex, nontrivial Computer Vision tasks. Aslo it is worth checking the existing approaches to make this models robust against adversarial examples [13] and perform common attacks on the models of such type.

Appendix

Appendix 1. The complete architecture of the backbone network.

l	Name	Dimensions	Parameters
0	InputLayer	$128 \times 128 \times 3$	0
1	Conv2D	$126 \times 126 \times 4$	112
2	BatchNormalization	$126 \times 126 \times 4$	16
3	MaxPooling2D	$63 \times 63 \times 4$	0
4	Conv2D	$61 \times 61 \times 8$	296
5	BatchNormalization	$61 \times 61 \times 8$	32
6	MaxPooling2D	$30 \times 30 \times 8$	0
7	Conv2D	$28 \times 28 \times 16$	1168
8	BatchNormalization	$28 \times 28 \times 16$	64
9	MaxPooling2D	$14 \times 14 \times 16$	0
10	Conv2D	$12 \times 12 \times 32$	4640
11	BatchNormalization	$12 \times 12 \times 32$	128
12	MaxPooling2D	$6 \times 6 \times 32$	0
13	Conv2D	$6 \times 6 \times 64$	18496
14	BatchNormalization	$6 \times 6 \times 64$	128
15	MaxPooling2D	$2 \times 2 \times 64$	0
16	Flatten	256	0
17	Dense	256	65792

Appendix 2. The environment description. We are using the Anaconda environment with Python 3.9.7; For model training, the Keras framework is used with Tensorflow v2.10.0; The computations are performed on GPU Nvidia GeForce MX450.

Література

[1] Z. Zou, Z. Shi, Y. Guo, J. Ye / Object Detection in 20 Years: A Survey. : arXiv preprint arXiv:1905.05055 (2019)

[2] *Xiaoke Shen* / A survey of Object Classification and Detection based on 2D/3D data : arXiv preprint arXiv:1905.12683 (2022)

[3] *Great Learning Team* / Real-Time Object Detection Using TensorFlow : mygreatlearning.com

[4] *Богдан Бугрій* / Атаки на глибокі нейронні мережі : Львів (2020)

[5] *Богдан Бугрій* / Розробка алгоритмів захисту від атак на глибокі нейронні мережі : Львів (2021)

[6] *Axel Thevenot* / A visual and mathematical explanation of the 2D convolution layer and its arguments : towardsdatascience.com

[7] *Kaushal Shah* / Building Multi Output Cnn With Keras : kaushal28.github.io

[8] *Jason Brownlee* / A Gentle Introduction to Pooling Layers for Convolutional Neural Networks : machinelearningmastery.com

[9] *Martin Riva* / Batch Normalization in Convolutional Neural Networks : baeldung.com

[10] *Raitis G.* / Battle City tanks repository : github.com

[11] *Sebastian Ruder* / An overview of gradient descent optimization algorithms : arXiv preprint arXiv:1609.04747 (2017)

[12] *Johannes Lederer* / Activation Functions in Artificial Neural Networks: A Systematic Overview : arXiv preprint arXiv:2101.09957 (2021)

[13] *Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, Rob Fergus* / Intriguing properties of neural networks : arXiv preprint arXiv:1312.6199 (2014)