UKRAINIAN CATHOLIC UNIVERSITY

BACHELOR THESIS

Rubik's Cube layout construction and validation algorithm

Author:
Bohdan BORKIVSKYI

Supervisor:
Oles DOBOSEVYCH

A thesis submitted in fulfillment of the requirements for the degree of Bachelor of Science

in the

Department of Computer Sciences
Faculty of Applied Sciences

Lviv 2020

Declaration of Authorship

I, Bohdan BORKIVSKYI, declare that this thesis titled, “Rubik’s Cube layout construction and validation algorithm” and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

Date:

UKRAINIAN CATHOLIC UNIVERSITY

Faculty of Applied Sciences

Bachelor of Science

Rubik's Cube layout construction and validation algorithm

by Bohdan BORKIVSKYI

Abstract

Many sports have some monitoring systems, that help judges to make wise decision in controversial situation, such as VAR in football. VAR allows judge to check situation from different views, so nothing is missed.

In this work we propose a prototype of such system for speedcubing, *i.e.* sport of solving different puzzles, such as Rubik's cube on time. This system can help to prevent mistakes on scrambling stage, when cube is mixed from solved state following the specific algorithm. The developed system is able to locate cube on scene and based on observable parts can check layout with a list of allowed layouts.

Acknowledgements

I would like to thank my supervisor Oles Dobosevych for his advice and help in dealing with troubles. Also, I would like to thank him, Yaroslava Lochman and Igor Krashenyi, for the valuable experience they shared on the Computer Vision course. I am grateful to my family and benefactors for the chance to study at UCU. And finally, I want to express my gratitude to friends, who supported me during my study.

Contents

List of Figures

List of Tables

List of Abbreviations

VAR	**V**ideo-**A**ssistant **R**eferee
FPN	**F**eature **P**yramid **N**etwork
DSC	**D**ice **S**imilarity **C**oefficient (Sørensen–Dice coefficient)
ANN	**A**rtificial **N**eural **N**etwork
CNN	**C**onvolutional **N**eural **N**etwork
IoU	**I**ntersection **o**ver **U**nion
LR	**L**earning **R**ate
SENet	**S**queeze-and-**E**xcitation **N**etowrk

List of Symbols

x	data sample
y	ground-truth label
$\hat{y}$	predicted label

Dedicated to the whole speedcubing community

Chapter 1

Introduction

Sport is about competition, and every participant is interested in this competition to be fair. People understand that human judgment cannot be accurate and started implementing different systems that will not miss relevant data and help the judge in making the right decision.

Speedcubing is not a very popular sport, but it already has a lot of fans, conducted competitions, and tournaments. Nevertheless, now everything is done by people, from scrambling puzzles to judging participant's attempts. If some mistake takes place at any stage of the competition, it makes solving attempt easier for someone, but also it is tough to detect this mistake. There was an accident in some competition when the puzzle was scrambled in the wrong way, and competitor set world record in solving Rubik's cube. Fortunately, the mistake was revealed, and the attempt was refused.

In order to prevent similar situations, we introduce a system that will follow the cube on the scrambling stage, checking if it is scrambled in a proper way. Using state-of-the-art deep learning techniques for semantic image segmentation [24], we detect cube on the image, then detect cube faces (i.e., the cube has six faces in total) and extract the color of each piece. After we formed the layout from extracted colors, we can compare it with the original layout.

Chapter 2

Related work

2.1 Rubik's cube

As Rubik's Cube was invented almost half-century ago, there were many attempts to understand its logic, Mathematics behind it. As a result, a lot of solving approaches [8, 23, 11] were invented. For its inventor - Erno Rubik, it took a month to solve it for the first time.

With raise of Information Technologies and Artificial Intelligence, researches now are not only able to solve the cube from a given state, but also receive information about this state directly from world. [25, 14]

2.2 ANN

Artificial neural networks [22] is a fundamental block of AI systems. Its creation was inspired by the human brain, so main units of ANN are called artificial neurons, and data flow inside ANN is similar to how brain neurons communicate with each other.

Neurons are arranged into layers; typical ANN structure consists of some amount of hidden layers, located between input and output layers (Fig. 2.1).

The ANN training process consists of two stages. The first one is forward propagation, and during this stage, initial information from sample x is propagated to hidden layers, and then output $\hat{y}$ and loss are calculated. The second stage is backpropagation [5]. At this stage, we calculate the gradient of our loss function for our weights and update weights to minimize loss.

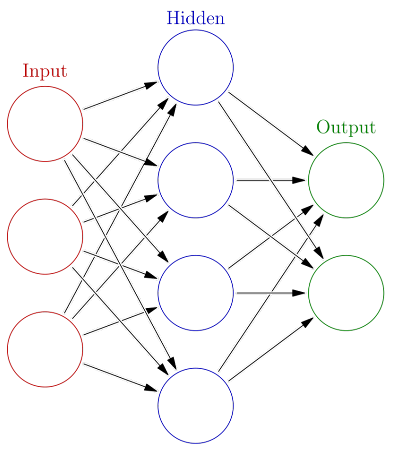


FIGURE 2.1: ANN structure [2]

2.3 CNN

Convolutional neural networks [9] presented a new type of layer that tries to solve some of the problems with layers connections in simple ANN.

ANN can handle some task that requires a small amount of data. Because neurons from different layers have a lot of connections with neurons on the next layers, adding new neurons to the input layer leads to a big increase in a number of connections on the following layers. It makes it not practical for input data with large size.

CNN introduces a new type of connection between layers. Now neuron on a convolutional layer (Fig. 2.2) is not connected with many neurons on the previous layer; it is connected with just a few of them. The second neuron is also connected with some portion of previous layer neurons but with a slightly different set of them. Benefits of such an approach are that every neuron knows information about a specific part of data, while in ANN due to high connectivity neuron has information from different parts of data. The other benefit is that number of parameters to be learned is extremely low in comparison to ANN. For each such layer, there is a common set of weights, called filter, which is applied to all neurons on the previous layer, and value is propagated to neurons, associated with that set of neurons.

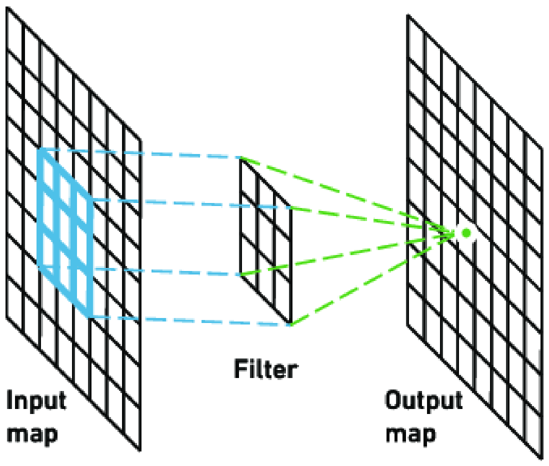


FIGURE 2.2: CNN layer structure [26]

2.4 Image segmentation

Nowadays, in the time of self-driving cars and other autonomous systems, object detection does not cause any surprise. These systems need better scene understanding than just a position of some object. We need to understand the boundaries of those objects and, in some cases, to distinguish between each of them. For example, in one case, we do not need to distinguish between buildings; it is one type of obstacle for an autonomous car. However, in medical research, it can be crucial to distinguish between individual cells.

A common way of doing image segmentation is to use architecture with downsampling and upsampling parts, where the downsampling part is some CNN network, and its results are then upsampled.

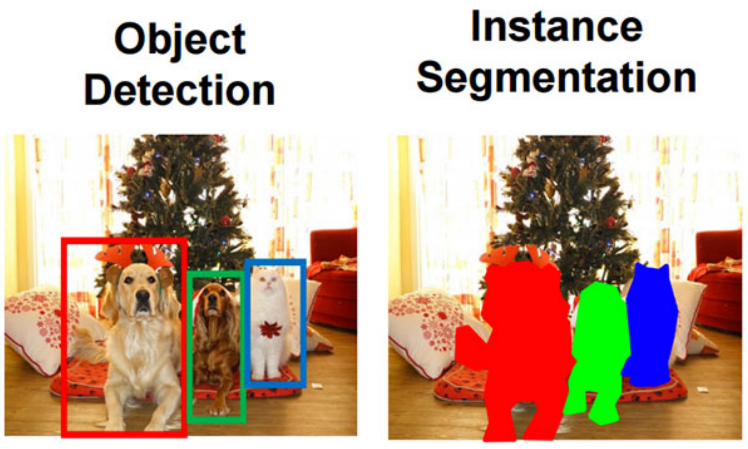


FIGURE 2.3: Object detection vs Image segmentation [32]

2.5 Existing approaches

There are different approaches to do Rubik's cube and faces detection; some of them are developed for detection from camera [10], and some others are robots-oriented [19].

In this [10] paper, researchers introduce a way of localizing the cube and detecting its faces using convolutional neural networks. The first stage is to detect the cube on the image - for this purpose, they trained CNN that predicts the center of the cube on the scene, and how large it is. Then they use K-Means [7] algorithm in combination with SLIC superpixel segmentation [34, 29] to identify cube pieces candidates. Then they used the same CNN architecture again for three models to predict cube faces centers.

This approach has shown good results in both cubes and their faces detection, as well as color extraction. However, it remains unclear how the system is going to behave if some part of the cube is covered by finger. Also, they assume that the cube has a basic cube layout; therefore, the system is not able to detect faces that do not conform to this layout.

Chapter 3

Proposed Method

The general algorithm to obtain the cube layout is to detect where the cube is located on the image, then locate cube faces and extract colors on each face.

3.1 Cube detection

The first stage is to detect the cube on the image. The simplest approach is to find sets of parallel lines, representing a cube. The problem occurs when the cube is in hand; not all lines can be completely visible. Also, there are approaches for finding the cube's corner points. These approaches are good, but they can fail if those points are covered with fingers or any other objects. Even if there is enough data to get corner points, some of the cube edges and faces cannot be visible because of some objects covering them, then the color of that object can be interpreted as the color of the cube piece. So along with the problem of cube detection, we need to know what is still the cube and what is something that covers it. Image segmentation is an approach that could solve both tasks.

3.1.1 Dataset

The first stage of image segmentation is data collection. We went through a variety of publicly available videos, where people solve Rubik's cube in official competitions and extracted frames from them. Because the system is designed to be used indoors, we were not looking for videos where people solve the cube outside.

In order to label the data, we have used Supervisely [33] web platform. It allows us to make precise object masks and export all masks together with a variety of configuration parameters. For each image, we manually created a mask with the Bitmap tool. Mask denotes only visible part of the cube; hence if any objects (e.g., fingers) cover some part of the cube, the mask will surround such objects.

To make even more data for model training we have used different augmentations [28, 6].

Augmentations can not only produce more data, but also can make big variety of data from small number of samples. Here we introduce these augmentations with corresponding probabilities (Table 3.1).

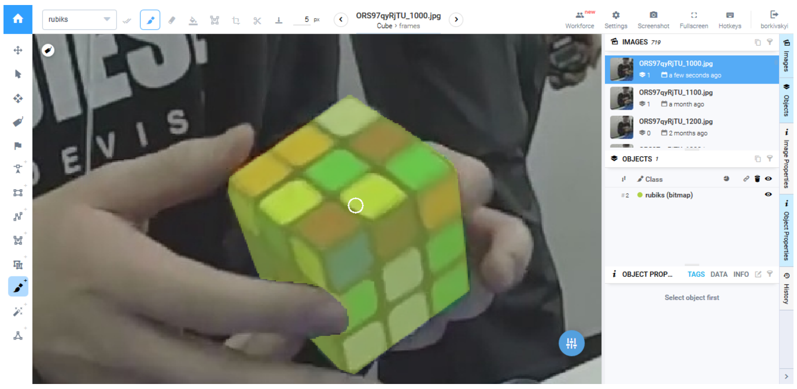


FIGURE 3.1: Example of mask labeled using Supervisely platform

Augmentation	Probability
HorizontalFlip	0.5
ShiftScaleRotate	1.0
CropNonEmptyMaskIfExists	0.9
RandomCrop	1.0
GaussianNoise	0.2
RandomBrightness	0.9
Blur	0.9
RandomContrast	0.9

TABLE 3.1: Augmentations

3.1.2 Models

During our research we conducted several experiments using Python [21] packages **pytorch** [27] and **segmentation-models-pytorch** [37] with two popular architectures in image segmentation [20, 24], namely UNet [30] and FPN[18].

UNet

UNet consists of two main parts - downsampling and upsampling pathways. The downsample part is responsible for extracting features from data. On each layer, it produces a feature map that is later combined with its correspondence from the upscaling part. In the second stage, valuable features are upscaled. However, some information is lost during the convolutional process, so we need to get extra information from that stage and concatenate current stage data.

FPN

At first glance, FPN is similar to UNet, but it has some major differences.

It also has feature extractor and upsampling parts. The main difference is that the final prediction is based not on the last layer but on the result of all upsampling layers that are concatenated altogether.

Also to solve problem with the information loss there are 1x1 convolutions used which are added with upsampling data.

3.1.3 Feature extractors

For better results, we trained our models with some pre-trained backbones. We conducted experiments with MobileNet [31] and SE-ResNeXt [12] backbones.

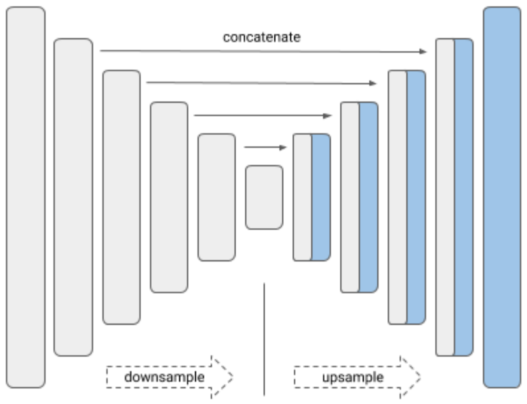


FIGURE 3.2: UNet architecture [36]

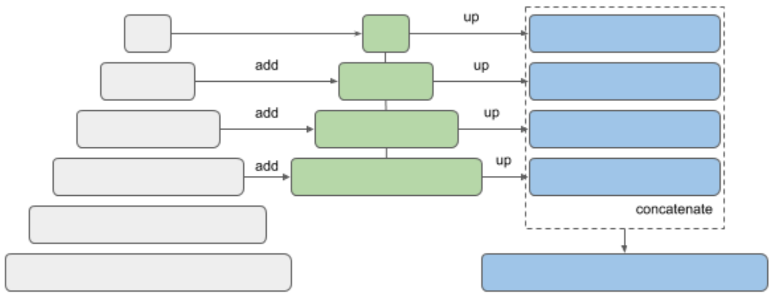


FIGURE 3.3: FPN architecture [36]

SE-ResNeXt50

SE-ResNeXt is an improvement of ResNeXt [35] with SE blocks [13] added to it. We decided to use improved ResNeXt version because it achieved great results compared to other models and won first place in the ILSVRC 2017 classification challenge.

MobileNetv2

Model efficiency is important not only in terms of its accuracy but also in terms of speed (i.e., the time required to make inference). So we decided to use MobileNet backbone, which has ten times fewer parameters then SE-ResNeXt (2 million compared to 25 million in SE-ResNeXt)

3.1.4 Training process

During train, we used such training setup:

Optimizer

For optimization, we used Adam [15] optimizer, which is now commonly applied.

Loss

As our loss function we use Dice Loss [17, 24]
It is based on DSC, which is used to measure the similarity between two sets.
Formula of loss is as follows:

$$DL(y, \hat{y}) = 1 - \frac{2y\hat{y} + 1}{y + \hat{y} + 1}$$

Metrics

As a metric we have used IoU, also known as Jaccard Index [20, 24]. It is one of the most popular metrics used in image segmentation. It shows similarity between y and $\hat{y}$ by comparing their common part (intersection) with part covered by both of them (union):

$$\text{IoU} = J(y, \hat{y}) = \frac{y \cdot \hat{y}}{y + \hat{y}}$$

Learning rate

During training, we have used an adaptive learning rate. If the metric was not improved during some period of time, the learning rate is going to be reduced by 10.

Initial LR value is 1e-4, and it can reduce down to 1e-7.

Results are presented in Chapter 4

3.2 Cube faces detection

The next step is to detect cube faces. Here we make an assumption - we assume that cube is in the cuboid state, so no faces are turned relatively to others. Although it is a normal situation in speedcubing that after the attempt is finished, some of the cube faces can be not aligned, but when the attempt starts, the cube has to be a geometric cuboid.

During the segmentation process, we omit fingers that potentially can cover some parts of the cube; therefore, the produced mask shape will not be convex. Any cube projection is convex itself: if only one face is visible - the projection is square, which is convex if there are two faces visible, projection shape can be either pentagon or hexagon, and if there are three faces visible - mask shape can only be a hexagon.

For hexagon mask to be not convex there are two options:

- if mask vertex belongs to cube corner A (with single visible color) - it has to be closer to corner C (opposite corner on the same face) than to the diagonal BD (that goes through other two face corners). But this is geometrically not possible in projective geometry (Fig. 3.4);
- if mask vertex belongs to cube corner A (with two visible colors) - it has to be closer to cube center then line BC (between two adjacent corners). If this happens - opposite face will disappear, cube center will become new mask

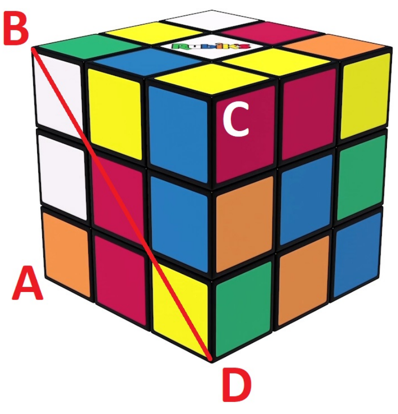


FIGURE 3.4: Case 1 [1]

vertex and adjacent hidden face will appear. Therefore such movements will lead us to new visible faces, but still to hexagon shape (Fig. 3.5).

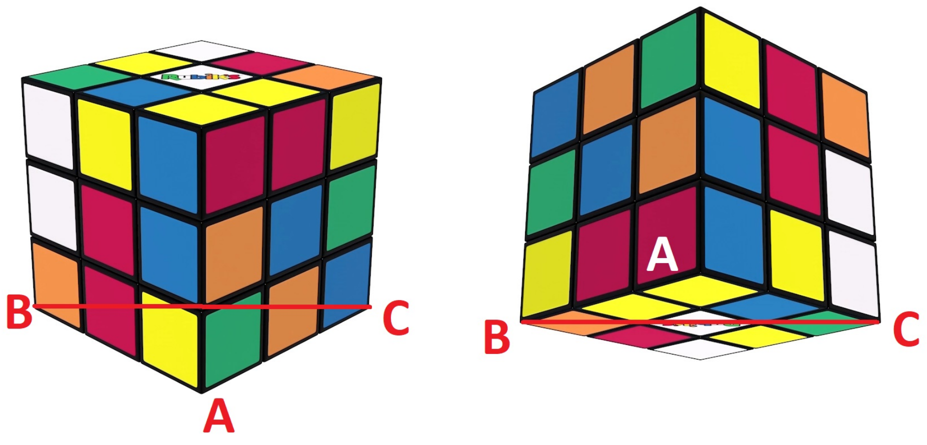


FIGURE 3.5: Case 2 [1]

Because two options are not possible - hexagon shape is always convex.

If the shape is pentagon - it means that there are two faces visible, and one of the corners is located exactly between two adjacent corners, so they form a single line. Because one of three parallel edges is located closer to the camera, it seems bigger than others, although all of them are of the same length. Consequently, the shape of such pentagon is convex. (Fig. 3.8)

In conclusion, every cube projection is convex. If our mask is not convex, the only reason for this is that some fingers cover the cube. In order to get a convex mask, representing the whole cube, we can calculate convex hull [16] of our mask (Fig. 3.6).

Convex hull gives us a set of convex mask borders, but because Rubik's cube has some rounded parts or there can be some noise on the image, the convex hull is split into small lines. So we gather those lines together, based on their angle on the

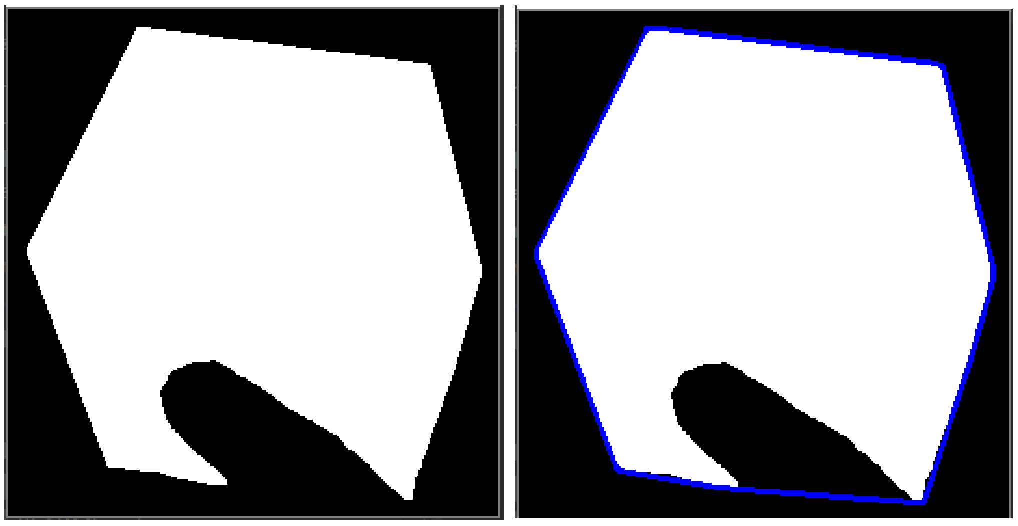


FIGURE 3.6: Convex hull

image. After this manipulation, we receive a set of lines that are mask edges indeed (Fig 3.7).

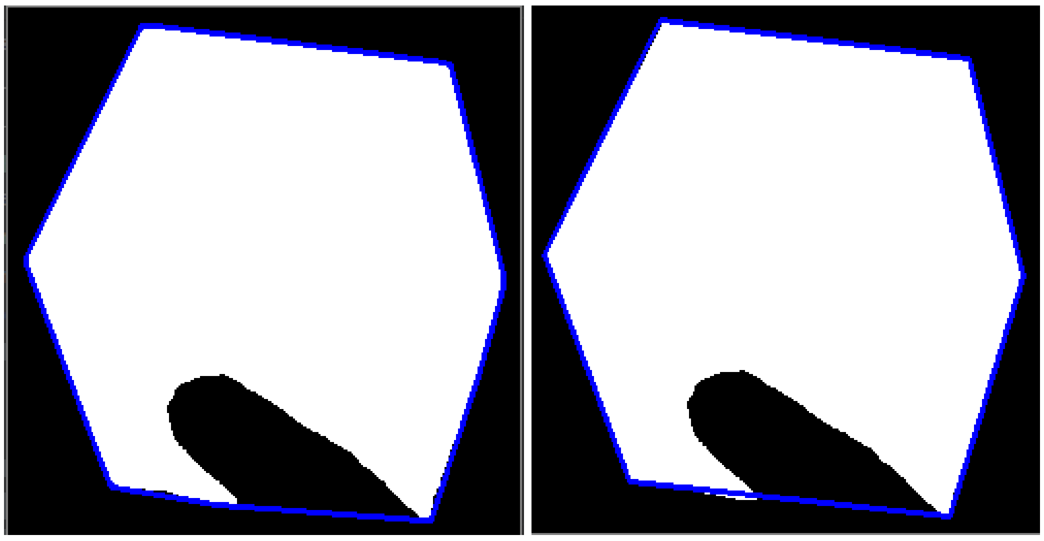


FIGURE 3.7: Convex hull reduced to six edges

After we obtained the mask, that is a projection of visible faces we can extract faces. Depending on the cube position, a number of visible faces can vary.

When the mask is square, then it represents only one face. If the mask has pentagon or hexagon shape, in order to find faces, we should find the last visible corner which is inside our mask. To do that, we can use calculations based on the perspective. Cube edges are parallel in the real world, but on the image, they are not. In fact, they do intersect, and the intersection point is called a vanishing point. We can use it to find the last corner. This corner belongs to edges, which are inside our mask, so to find their intersection, we firstly should find those edges. For each such edge, there are two visible edges that are parallel to it, and which are edges of cube mask. If the mask has five corners, then there is only one such edge, and if the mask has six corners, then there are three edges.

In a case with a pentagon mask (Fig. 3.8), when there are two faces, we already know three points of each face. But to extract face from the image, we need four of them. Firstly, we have to find intersection point A between parallel cube edges (parallel in the real world) because at this point they also intersect with the edge we want to find. The first point of this edge is point B that does not belong to any of the initial edges. The second point C lies on edge opposite to the first point and is the intersection of this edge with a line going through points A and B. As we know coordinates of points for the edge and for the line going through points A and B, we can easily calculate the intersection point C. So now we know all four points for each face and can extract them from the image.

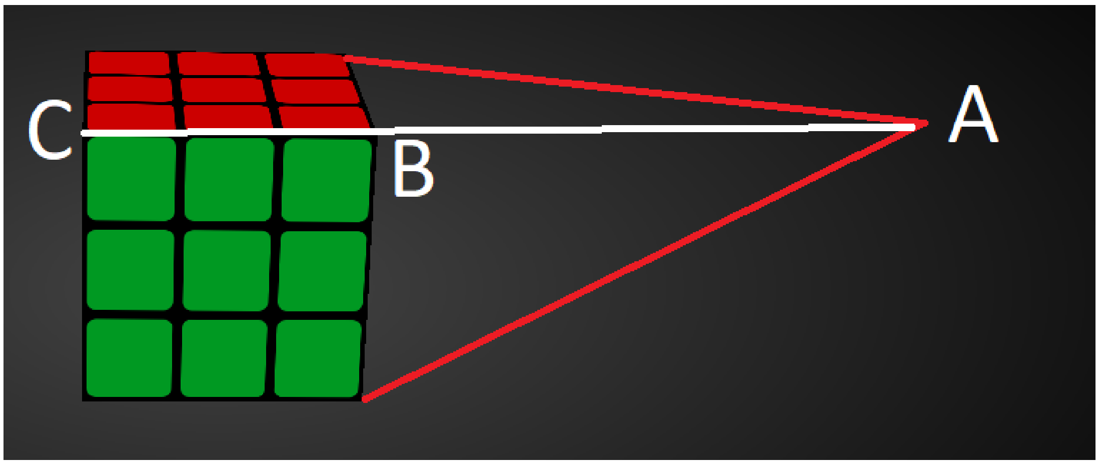


FIGURE 3.8: Intersection point C in case of two visible faces

If the mask is a hexagon(Fig. 3.9), we also know three points for each of the three faces. The last point belongs to a corner that is visible but is inside the mask (corner O). It is also an intersection point of three edges that are visible but are not mask edges. Here we apply the same approach as with pentagon mask with a small difference - edges intersect with each other, but not with mask border edges. In an ideal situation, all three edges will intersect in one point, but because the mask cannot be absolutely precise, there can be some deviation. So we calculate intersection for every pair out of three edges, receiving three points in total. Then we calculate the mean point for these points, and it is nearly the corner of the cube and the fourth point for every face. Now we can extract all three faces from the image.

3.3 Cube faces extraction

In order to get an image of the individual face, we can apply a transformation to face area to convert it from quadrangle to square.

There are different types of geometries, with different transformations allowed in each.

As was stated before, parallel edges of the cube, in fact, intersect on the image, so there is no parallelism, and the only transformation we can use is a projective transformation.

Therefore, calculating the transformation matrix from source to destination positions and then applying it to the cube image will result in a square image of one face. We have to calculate three different matrices to get the image of each face.

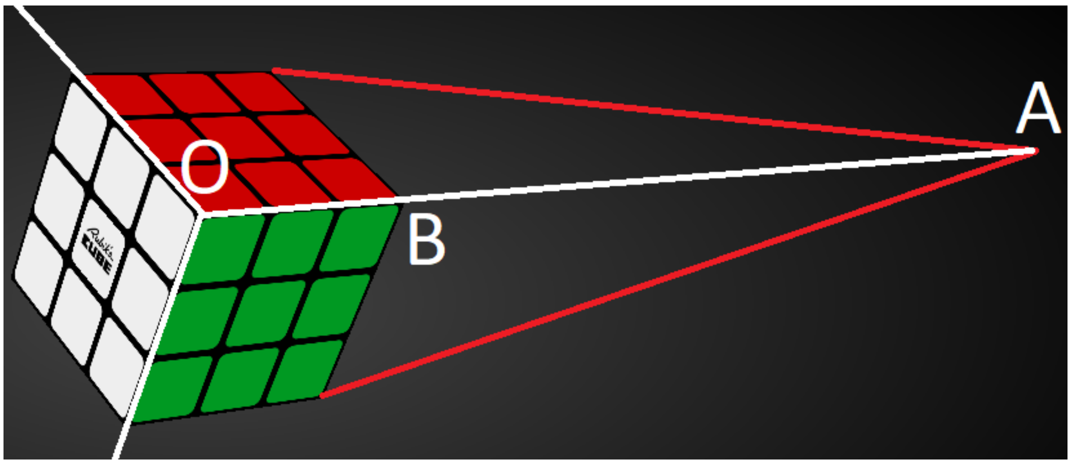
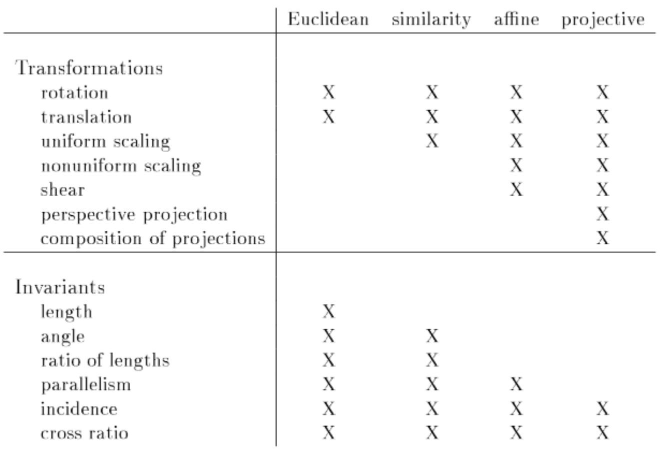


FIGURE 3.9: Intersection point O in case of three visible faces



	Euclidean	similarity	affine	projective
Transformations				
rotation	X	X	X	X
translation	X	X	X	X
uniform scaling		X	X	X
nonuniform scaling			X	X
shear			X	X
perspective projection				X
composition of projections				X
Invariants				
length	X			
angle	X	X		
ratio of lengths	X	X		
parallelism	X	X	X	
incidence	X	X	X	X
cross ratio	X	X	X	X

FIGURE 3.10: Types of geometries [4]

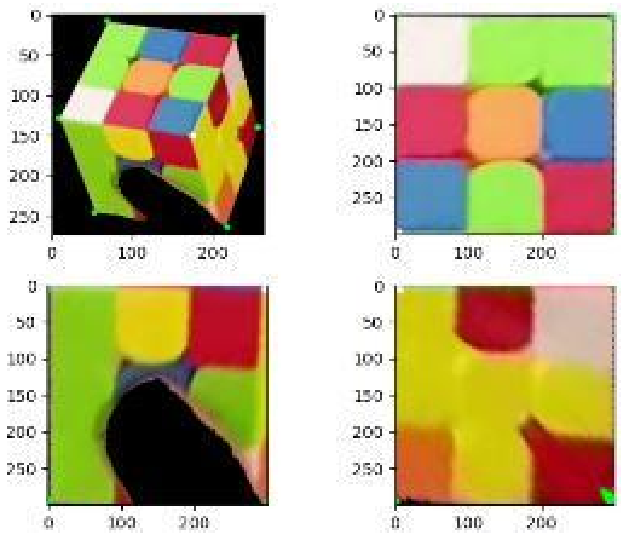


FIGURE 3.11: Faces extraction

3.4 Color extraction

After we have a single face image, we can extract colors from it. As our image is square and we exactly know how pieces are arranged on the face, we can just split the face into nine squares.

Nowadays stickerless cubes (an example is on Fig. 3.11) become more popular, and the whole face piece is of a single color, but still, there are a lot of stickered cubes (an example is on Fig. 3.9), where the cube background is of one color, and the stickers are of a different color. But there is a chance that we can take background into account, or there can be some noise in case of not precise face extraction. So to detect piece color, we decided not to look at its borders with some configurable margin.

After we have calculated mean values of color on the piece, we compare it with predefined color values, which includes not only pure colors (e.g. (0, 255, 0) for green) but also some mixed variations. Because each color is set of three values in the RGB scale, we can perceive it as a vector in three-dimensional space. In order to choose the closest color to one of the defined, we can calculate the difference between color vector and predefined vectors. The vector with the smallest difference corresponds to the color of the piece.

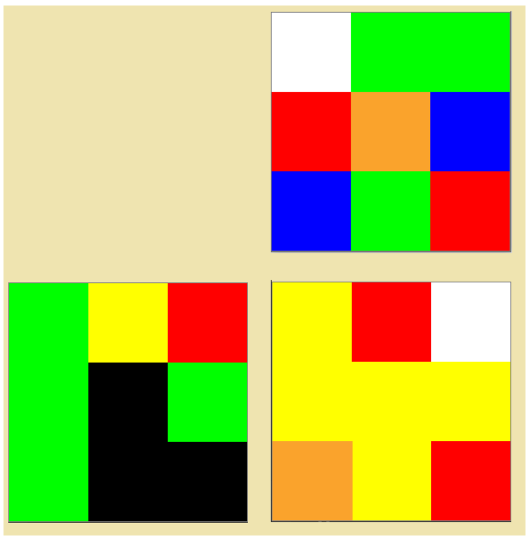


FIGURE 3.12: Color extractions result

3.5 Layout validation

In order to compare obtained colors with the real layout, we designed a program that simulates Rubik's cube movements and can produce a layout for a selected scramble.

Based on cube centers' colors, we adjust it to the default cube position and check if the cube layout fits the scramble.

In figures 3.11 and 3.12 there are some cube pieces that are not visible due to fingers covering them, so corresponding pieces are predicted to be black. At this stage, we do not take these pieces into account, so they are omitted during validating layout.

After validation succeeded, we mark cube on the frame with the corresponding color (i.e., green if the cube is scrambled correctly and red otherwise). In Fig. 3.13 we first validated the cube layout with its corresponding scramble and then with some other scramble. As we can see, for appropriate scramble the cube was marked with a green border, and for the wrong scramble with a red border.

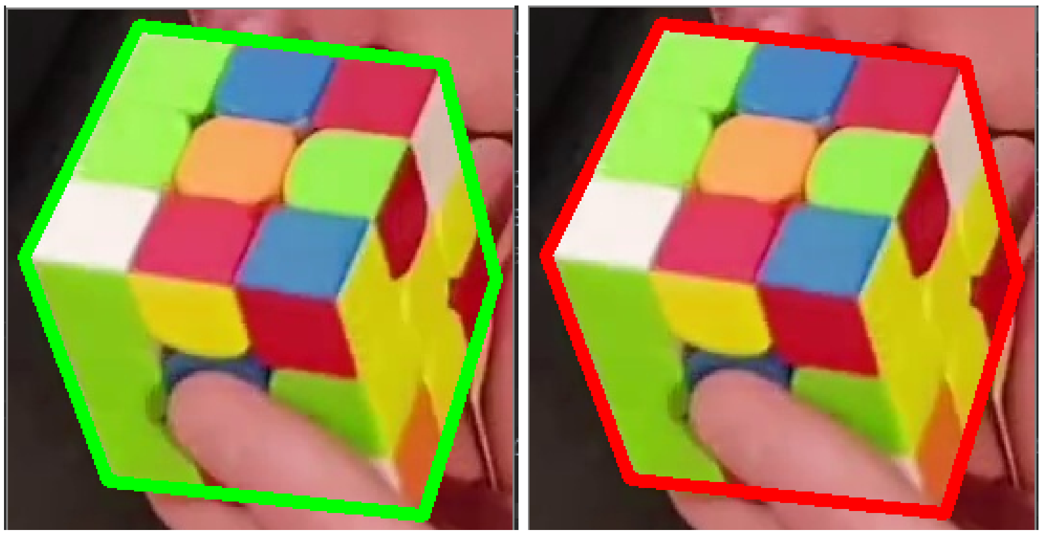


FIGURE 3.13: Color extractions result

Chapter 4

Experiments results

For gathering training data, as well as predicted labels, we used Wandb [3], which allowed us to instantly follow the training process on a remote computer, combine different metrics on one plot for better comparison and to follow improvements in image segmentation based on the predefined image.

Every model was trained for 100 epochs, and results are presented in Table 4.1

Although the best loss value was reached by FPN with a MobileNetv2 backbone, the best IoU score was reached by UNet with SE-ResNeXt50 backbone, which loss was two times bigger than the FPN result.

Both backbones achieved similar results during train with different architectures, so henceforth we introduce results for UNet with SE-ResNeXt50 and FPN with MobileNetv2 backbone, as they achieved better results for corresponding architectures.

As we can see from results (Fig. 4.1), loss decreased very quickly in the experiment with the FPN model, while the loss in the UNet experiment was decreasing slowly during all training.

During the FPN experiment, the learning rate was reduced a few times until it reached its minimal value, while during UNet experiment, the learning rate was reduced only once at the end of the training process. Although UNet reached the highest IoU value, it has the potential to improve this result with the longer training process.

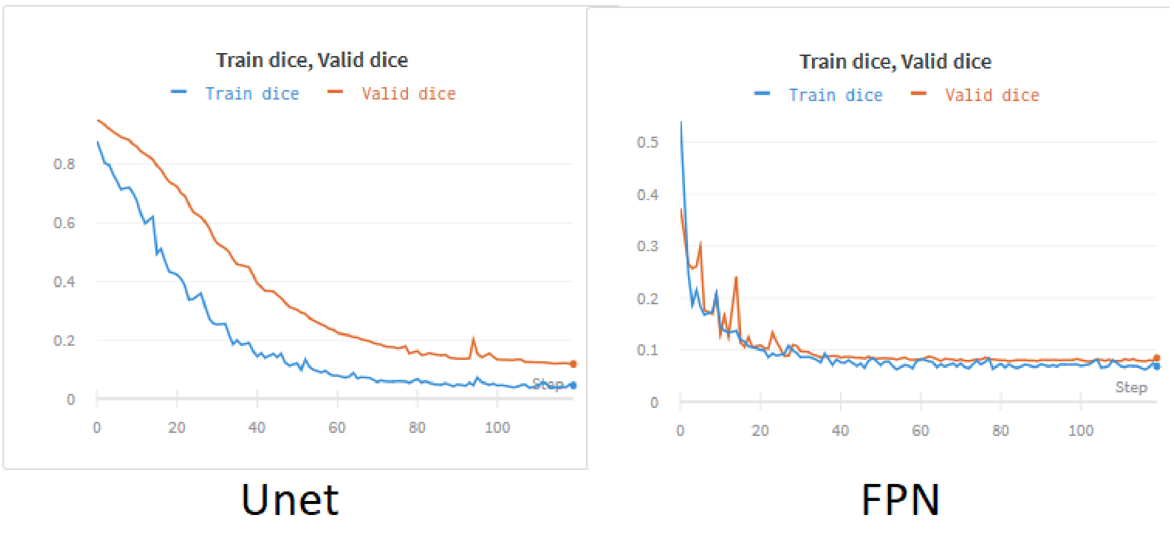


FIGURE 4.1: Dice loss change during training

Here we introduce improvements of model understanding during the training process (Fig 4.4). On the first figure, the model went through 6 epochs, it is not confident whether the colorful mat is a cube or not, but it knows that hands and

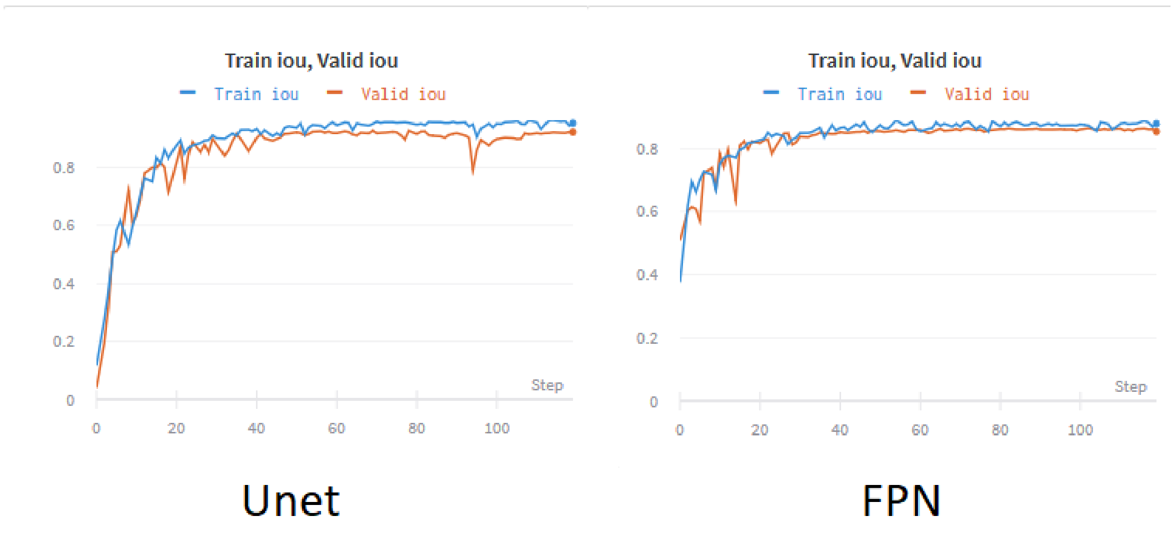


FIGURE 4.2: IoU change during training

Model	Backbone	Validation Loss	Validation IoU
Unet	SE-ResNeXt50	0.1181	**0.9273**
Unet	MobileNet-v2	0.1237	0.9182
FPN	SE-ResNeXt50	0.0777	0.8634
FPN	MobileNet-v2	**0.0565**	0.8978

TABLE 4.1: Training results

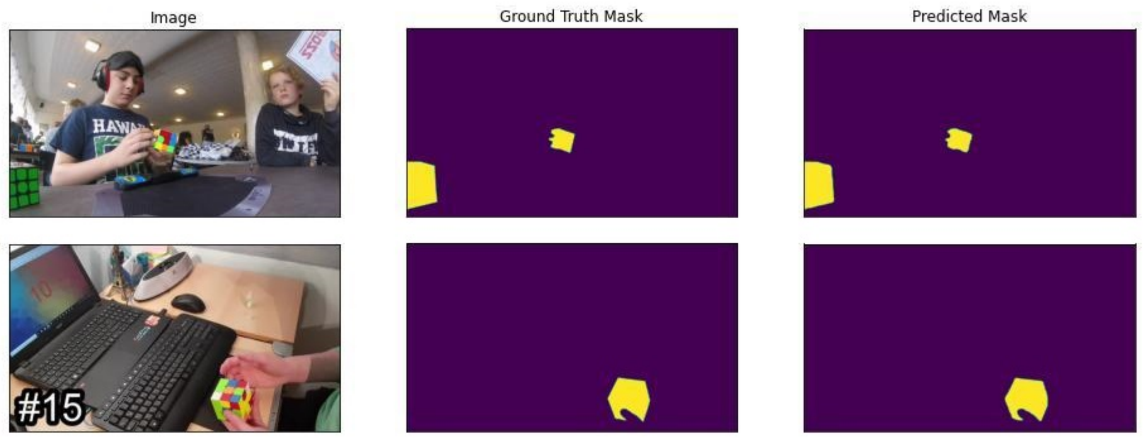
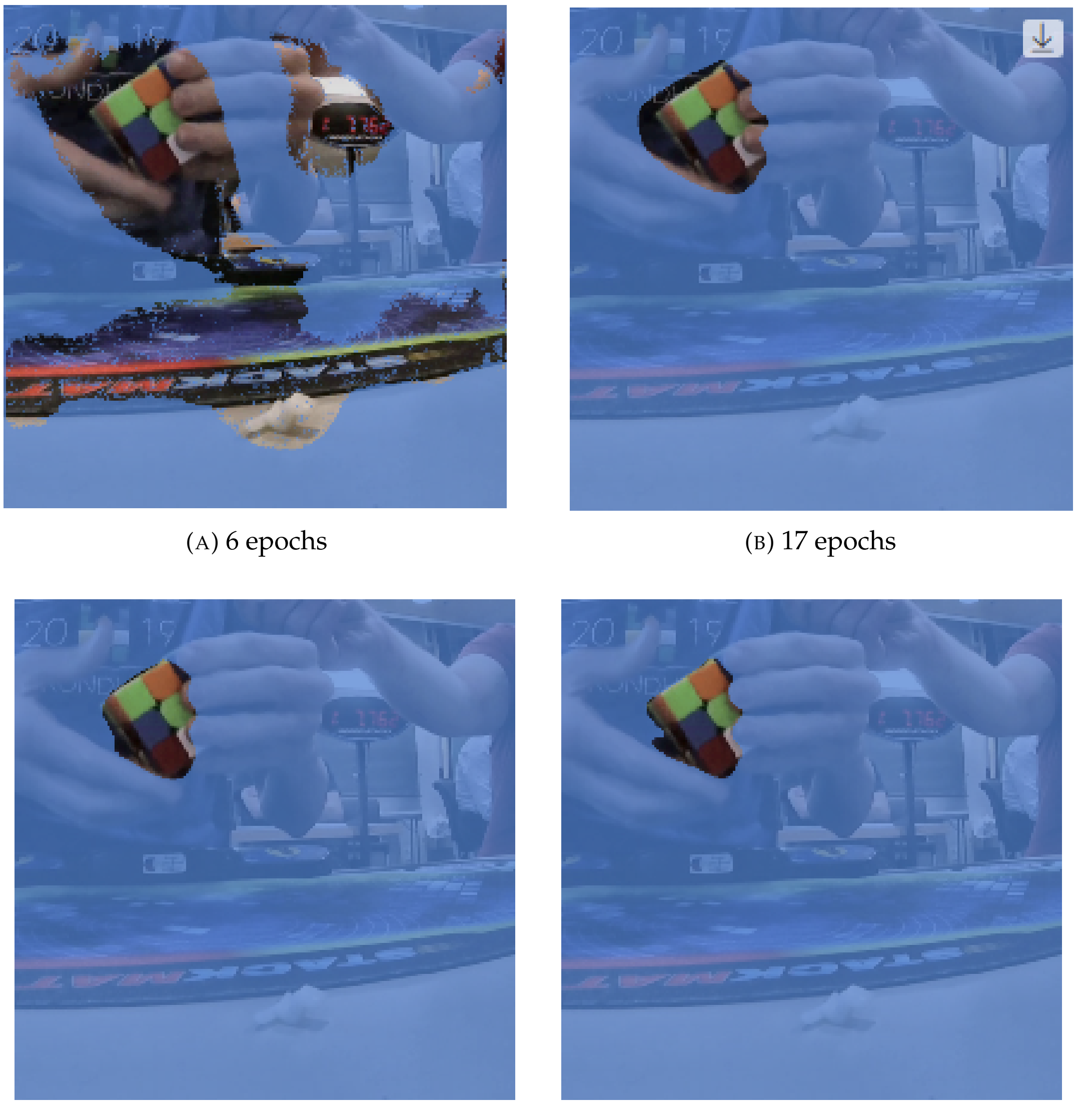


FIGURE 4.3: Visual comparison of model predictions with ground-truth labels

monotone color surfaces are not a cube for sure. On the second figure (17 epochs of training), it is pretty confident where the cube is on the image, but it still has doubts where the cube area ends. On the third figure (25 epochs), it marks cube quite precisely but still has some doubts about fingers. After 50 epochs, it is confident in segmenting cube and fingers but has some problems with dim faces.



(A) 6 epochs

(B) 17 epochs

(C) 25 epochs

(D) 52 epochs

FIGURE 4.4: Segmentation model predictions during training

Chapter 5

Conclusions

5.1 Dataset

We have collected a dataset of cube masks, where objects that cover the cube are omitted. This dataset can be used to train segmentation models. It can also be used for other purposes. For example, if one wants to create a cube detection system, just to know the cube location, they can calculate the middle point for the cube mask and train an utterly different model.

5.2 Approach

We propose a new approach for cube detection using recent advancements in image segmentation. As a result, we know not only cube location on the image, but also receive its mask.

Using cube mask and perspective rules, we compute all points necessary to extract cube faces. From each face, we extract pieces' colors and form an actual cube layout. And then, we compare it with a layout created from the given scramble.

Parts of the developed system can be used standalone. For example, the model can be used to predict labels for some other tasks that do not require following faces extraction, or the color extraction part can be used with well-prepared images of cube faces.

Chapter 6

Future work

6.1 Model performance

Although the model is performing quite well (best validation IoU equals 92%), but it may struggle to perform in dim environments. Also, on all images in the dataset, there are representatives with white skin color solving the cube, so we do not know how the model will perform with representatives of different skin colors. In order to solve these problems, we have to collect more data, representing different situations, such as brightness in the room, cube type, and people solving the cube.

As seen from Results (Chapter 4), even with a good metric score, it has the potential to perform even better. During experiments, we were training models for 100 epochs, but if there was no limit, the model could achieve better results.

6.2 Layout validation

Current algorithm checks if the cube is scrambled correctly based on a default Rubik's cube layout. However, in competitions, it is not required to have an identical layout so that it can vary (e.g., two colors are swapped, or instead of one color purple is used). In order to properly handle such cases, layout verification can be modified, not to check exact color locations, but relative location. For instance, if one of the colors is changed, then we have to check that all occurrences of that color on cube coincide with occurrences of any color in true layout.

Now the algorithm checks similarity with predefined colors. However, when a new version of layout validation is implemented, there will be no need to define a cube piece color. Then we just have to create a map of identical colors, and it should match a map of any color on the true layout.

Bibliography

[1] *3D rubik s cube model - TurboSquid 1196124*. URL: https://static.turbosquid.com/Preview/001196/124/44/3D-rubik-s-cube-model_Z.jpg.

[2] *Artificial neural network - Wikipedia*. URL: https://upload.wikimedia.org/wikipedia/commons/thumb/4/46/Colored_neural_network.svg/800px-Colored_neural_network.svg.png.

[3] Lukas Biewald. *Experiment Tracking with Weights and Biases*. Software available from wandb.com. 2020. URL: https://www.wandb.com/.

[4] Stan Birchfield. *An Introduction to Projective Geometry (for computer vision)*. 1998. URL: http://robotics.stanford.edu/~birch/projective/.

[5] Laurent Boué. *Deep learning for pedestrians: backpropagation in CNNs*. 2018. arXiv: 1811.11987 [cs.LG].

[6] Alexander Buslaev et al. "Albumentations: Fast and Flexible Image Augmentations". In: *Information* 11.2 (2020), p. 125. ISSN: 2078-2489. DOI: 10.3390/info11020125. URL: http://dx.doi.org/10.3390/info11020125.

[7] Marco Capó, Aritz Pérez, and Jose A. Lozano. *An efficient K -means clustering algorithm for massive data*. 2018. arXiv: 1801.02949 [stat.ML].

[8] Erik D. Demaine et al. *Algorithms for Solving Rubik's Cubes*. 2011. arXiv: 1106.5736 [cs.DS].

[9] Jiuxiang Gu et al. *Recent Advances in Convolutional Neural Networks*. 2015. arXiv: 1512.07108 [cs.CV].

[10] Jay Hack and Kevin Shutzberg. *Rubiks Cube Localization, Face Detection, and Interactive Solving*. 2015. URL: http://cs231n.stanford.edu/reports/2015/pdfs/jaykevin_final.pdf.

[11] Michal Hordecki. *ZZ speedcubing system*. 2008. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.190.6040&rep=rep1&type=pdf.

[12] Jie Hu et al. *Squeeze-and-Excitation Networks*. 2017. arXiv: 1709.01507 [cs.CV].

[13] Tongwen Huang, Zhiqi Zhang, and Junlin Zhang. "FiBiNET". In: *Proceedings of the 13th ACM Conference on Recommender Systems* (2019). DOI: 10.1145/3298689.3347043. URL: http://dx.doi.org/10.1145/3298689.3347043.

[14] Włodzimierz Kasprzak, Wojciech Szynkiewicz, and Łukasz Czajka. "Rubik's cube reconstruction from single view for Service robots". In: *Machine Graphics & Vision International Journal* 16 (May 2007), pp. 451–459.

[15] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2014. arXiv: 1412.6980 [cs.LG].

[16] Lingfeng Li et al. *Convex hull algorithms based on some variational models*. 2019. arXiv: 1908.03323 [cs.CV].

[17] Xiaoya Li et al. *Dice Loss for Data-imbalanced NLP Tasks*. 2019. arXiv: 1911.02855 [cs.CL].

[18] Tsung-Yi Lin et al. *Feature Pyramid Networks for Object Detection*. 2016. arXiv: 1612.03144 [cs.CV].

[19] Shenglan Liu et al. *Color Recognition for Rubik's Cube Robot*. 2019. arXiv: 1901.03470 [cs.CV].

[20] Xiaolong Liu, Zhidong Deng, and Yuhan Yang. "Recent progress in semantic image segmentation". In: *Artificial Intelligence Review* 52.2 (2018), 1089–1106. ISSN: 1573-7462. DOI: 10.1007/s10462-018-9641-3. URL: http://dx.doi.org/10.1007/s10462-018-9641-3.

[21] M. Lutz and an O'Reilly Media Company Safari. *Learning Python, 5th Edition*. O'Reilly Media, Incorporated, 2013. URL: https://books.google.com.ua/books?id=LWM6zQEACAAJ.

[22] Zhongkui Ma. *The Function Representation of Artificial Neural Network*. 2019. arXiv: 1908.10493 [cs.LG].

[23] Stephen McAleer et al. *Solving the Rubik's Cube Without Human Knowledge*. 2018. arXiv: 1805.07470 [cs.AI].

[24] Shervin Minaee et al. *Image Segmentation Using Deep Learning: A Survey*. 2020. arXiv: 2001.05566 [cs.CV].

[25] OpenAI et al. *Solving Rubik's Cube with a Robot Hand*. 2019. arXiv: 1910.07113 [cs.LG].

[26] *Outline of the convolutional layer*. URL: https://www.researchgate.net/figure/Outline-of-the-convolutional-layer_fig1_323792694.

[27] Adam Paszke et al. *PyTorch: An Imperative Style, High-Performance Deep Learning Library*. 2019. arXiv: 1912.01703 [cs.LG].

[28] Luis Perez and Jason Wang. *The Effectiveness of Data Augmentation in Image Classification using Deep Learning*. 2017. arXiv: 1712.04621 [cs.CV].

[29] Carl Yuheng Ren, Victor Adrian Prisacariu, and Ian D Reid. *gSLICr: SLIC superpixels at over 250Hz*. 2015. arXiv: 1509.04232 [cs.CV].

[30] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. *U-Net: Convolutional Networks for Biomedical Image Segmentation*. 2015. arXiv: 1505.04597 [cs.CV].

[31] Mark Sandler et al. *MobileNetV2: Inverted Residuals and Linear Bottlenecks*. 2018. arXiv: 1801.04381 [cs.CV].

[32] *Step-by-Step Tutorial on Image Segmentation Techniques in Python*. 2019. URL: https://www.analyticsvidhya.com/blog/2019/04/introduction-image-segmentation-techniques-python/.

[33] *Supervisely - Web platform for computer vision. Annotation, training and deploy*. URL: https://supervise.ly.

[34] Thomas Verelst, Matthew Blaschko, and Maxim Berman. *Generating superpixels using deep image representations*. 2019. arXiv: 1903.04586 [cs.CV].

[35] Saining Xie et al. *Aggregated Residual Transformations for Deep Neural Networks*. 2016. arXiv: 1611.05431 [cs.CV].

[36] Pavel Yakubovskiy. *Segmentation Models*. https://github.com/qubvel/segmentation_models. 2019.

[37] Pavel Yakubovskiy. *Segmentation Models Pytorch*. https://github.com/qubvel/segmentation_models.pytorch. 2020.