Good day to dear members of commission and to everyone attending. My name is Savelii Dosaev and today, I present a software project that I've been working on for the past several months called make a knight's move! Unfortunately, my 3 team members from other educational programme decided to slack and not to do anything, so today I will be presenting the project only from my perspective

The project is about helping people learn chess who prefer to use a real chessboard. To do that, one needs a phone application that can take a photo of a real chessboard, analyze it and propose advice. And it was I who had to turn an image to a digital board. Therefore, the object of the course work is Image recognition, or in particular finding the underlying chessboard grid followed by an appropriate change of perspective. The work also includes the creation of a convolutional neural network to detect types of chess pieces.

Now, the popularity of chess has grown significantly in the last two years. Two crucial reasons for that is immense popularization of chess by chess grandmasters that started to openly stream their chess games, as well as the extremely popular Netflix series called 'Queen's gambit'. These reasons created interest in communities to try out chess. But one should remember that chess is not a straightforward game. It takes quite some time to learn from mistakes, and to learn patterns and strategies. While it is easy to do online, with all analytical services at hand, people that use a real chessboard are at a disadvantage.

Therefore the goal for the team is to create a free android application that will take a picture of a real chessboard, process it and propose a tactic with explanation. There are four tasks that were split between team members to achieve the proposed goal: 2 guys for the application and 2 guys for the algorithm to turn a picture to position.

Since the project is closely intertwined with chess, there may be some unknown words that I will use. The most important definition here is FEN - which is a representation of a position in a compact way. You can see the example here: slashes divide different rows, numbers - blank tiles in a row, lower case letter - black piece, upper case - white piece.

There are three works that tried to solve the same task I have in my project. The first work made by these 3 Polish gentlemen is great. It is a breakthrough in the topic of chessboard recognition. The author claims the accuracy of chessboard capture to be 95% and speed of execution of 4.6s. Accuracy of chess piece detection is 95% and time is not mentioned. The second algorithm has a good accuracy of 90% for both tasks, but execution time is horrendous of up to several minutes. Third method utilizes the chess capture of the first method, and has chess piece detection with high accuracy of predicting chess pieces and low accuracy of predicting color. Its execution time is around 10 seconds.

From that, I could build the requirements for my algorithms to be either on par or even better. Time execution should be less than 10 seconds, and the accuracy for both algorithms should be at least 95%

Now let's get to the essence of the project. To transform a picture to position, 4 steps should be completed: chessboard capture, chess piece detection, chess rules and chess engine. I was responsible for the first two steps.

In the chessboard capture algorithm, It is needed to find a chess grid, which can be mainly achieved with one of two approaches: using geometry or using a neural network. Every decision made by a geometric approach can be explained, and there is no need for anything auxiliary to make it work. As for neural networks, they are significantly faster and much more robust, however, it acts like a black box. Since I'm lacking knowledge of non-trivial CNNs usage, and geometry provides almost guaranteed solution, I decided to go with it.

To successfully find a chess grid we need to do 5 major steps: find lattice points, find contours, find the orientation of a chessboard in the image, then warp it and crop it.
Lattice points can be found in two ways: using line detection or using gradient of an image. I went with line detection using Hough line detection algorithm first, since using lines makes more sense when one needs to find lattice points. Lattice points are supposed to be on the intersection of vertical and horizontal lines, and as you can see, the result is not that good. One line is just missing and some are tilted. Worst of all, it works for 5 seconds just to get lattice points. I then tried the second approach.

One can use a gradient of the image in the direction of x, direction of y, combine it and and prune it. The pruning was done through thresholding and using non-max suppression. Top left image is original, right to it is combined gradient, left bottom is pruned gradient. While there is still a lot of noise, one can be sure that most of lattice points lie there. It works lightning fast, which is a major upside.

Next step is to find contours in the image, which can be found easily with cv2 functions. The objects of interest are cells of the board. However, the method finds around 500 contours, which is a little too much. To prune contours, I check whether it is a square in terms of angles, check if it has four corners, and check if these corners lie near lattice points. After that, only 32 remain, and they all need to be checked.

I would create the identity chess tile that lies in the image plane and find the change-of-basis matrix. Using it, I can take a whole-screen identity grid, transform it with a matrix and check how many lattice points overlap with the grid. The grid with the most overlaps wins.

I would then get the corner points, warp the image with the inverse of change-of-basis matrix and crop everything unneeded. Here you can see the result of the chessboard capture algorithm. Every resulting photo is 600x600 pixels, which can be uniformly split in tiles.

One problem to solve is to get orientation right. Someone could take a photo from the side, which, if remained not fixed, will give out incorrect FEN. To fix it, it suffices to use binary filter, and since there are at least 32 unoccupied tiles, we are practically guaranteed that there will be a free black cell. Once we have the blackest cell, I check its index, and if it is incorrect for a black cell, I rotate the image 90 degrees.

After it is done, one can split the warped image in 64 tiles, get predictions for every tile from CNN and use it to create FEN.

CNN was trained using a dataset of 500 labeled chessboards. They were all preprocessed using my algorithm and then cut and assigned a class. I used InceptionResNetV2 for a base CNN, since it gave me the best results.  It was trained for 15 epochs to avoid overfitting, and I achieved 97% on test validation.

The result of execution of both algorithms could be seen here. The only thing that needs to be provided is path to the image. Here, CNN only made one mistake that can be easily fixed with chess rules.

Overall, both of my algorithms work in the range from 7 to 9 seconds. The chess capture algorithm works with incredible accuracy and is quite robust: even if a hand obstructs a quarter of a board, it will still converge correctly. CNN is not that robust since I used the same board to train it.

To upgrade the work, one can find an even better combination of parameters for the functions, or just switch to a neural network. In the case of CNN, diverse data will make it much better and finding a smaller base model can increase speed.

Here you can find all the referenced works, and that's it! Thanks for your attention, I'm ready to answer your questions.