Assignment 2 – Computer Vision SS2025

Mohamed Aziz Boughalmi 202411510

Pijus Sadauskas 202412139

Jacob Heye Hilbrands 202411953

Task 2

In this project we tackled the ChessReD piece-counting problem in two complementary ways: (1) first as a continuous regression task, and then (2) as a 32-class classification task, using a single ImageNet-pretrained ResNet-50 backbone. For regression, we replaced the final fully connected layer with a three-layer MLP head (2048→512→256→1) using ReLU activations and 50 % dropout; for classification we used a two-layer head (2048→256→32) with label-smoothed cross-entropy.

To measure the effect of a brief "head-only warmup," every run followed a two-phase schedule over 30 epochs: for the first seven epochs we froze all convolutional layers and trained only the new head at an AdamW learning rate of 1×10^{-3}; for epochs 8–30 we unfroze the entire network and fine-tuned end-to-end at 1×10^{-4}. As a control, we also trained each head with **no freezing** (i.e. all layers trainable from epoch 1) using identical hyperparameters.

On the held-out test set we evaluated regression models with mean squared error (MSE) and mean absolute error (MAE), and classification models with accuracy, macro-averaged recall, and macro-averaged precision. The results for all four variants are summarized below:

	MSE	*MAE*	*Accuracy*	*Recall*	*Precision*
Regression (freeze)	4.208128	1.621030	0.209150	0.237907	0.245642
Regression (no freeze)	1.704736	0.985883	0.349673	0.347143	0.315798
Classification (freeze)	16.722222	2.728758	0.303922	0.140884	0.128019
Classification (no freeze)	5.712418	1.699346	0.320261	0.190499	0.161461

We see that in both regression and classification settings, **full end-to-end fine-tuning** (no-freeze) substantially outperforms the head-only warmup: it lowers MSE/MAE in regression and improves classification accuracy, with corresponding gains in recall and precision. These results suggest that, for ChessReD, initially freezing the backbone offers only limited benefit compared to unconstrained fine-tuning. Overall, our regression model with no freezing works best and is used in the attached script.py. We also tested other smaller models like ResNet-18, but they performed worse in every category than the Resnet-50 model, so we did not document their results.

Task 3.1 : Chess pieces detection

For this task, our final choice is to use a pretrained YOLOv8 model that we finetuned to the chessdata we have. The images have high resolution which makes the data preparation and the training very slow. So, we resize the images to 512x512, which is a reasonable resolution that still captures enough details for detection. The annotations in the dataset have already been divided into training, validation, and testing sets, so we use that partition. The YOLO model requires the data to be structured as such:

The file data.yaml contains the paths to the images and labels and the names of the categories. We train the model for 10 epochs with a batch size = 16. We noticed that by the last 2 epochs there was no improvement of the metrics, so we don't have to train for more epochs.

Result on the test set:

- mean average precision50: 0.98
- mean precision: 0.95
- mean recall: 0.94

Task 3.2: Digital twin

First approach: train a CNN on the images and their annotations to output an 8x8 matrix representing the board. This failed because the model started overfitting very early.

Next approach: train a CNN to extract the corners of the board which then we use in collaboration with the output of task 2.1 to create the digital twin. We didn't proceed with this approach because the CNN was not very precise: It learned to get very close to the corners but there are always some errors. We need perfect corner extraction to be able to use that to create the digital twin.

Final approach: Use the corners and the homography extracted in task 1 in addition to the pieces detection done by the YOLO model to create the digital twin.

To overcome the problem of overlapping pieces and pieces that seem to belong to many squares, we use only the center of the bottom third of the bounding box.

Of course, the results will be limited by the results of task 1. Orientation of the board is still a **problem:** for some images it's vertical, for others it's horizontal. That is because the order of the corners extracted in the first task is not consistent and depends on how the photo is taken.

Evaluation:

As we've established in the previous section, our pipeline struggles to get the right orientation of the chessboard. To evaluate the results, we used the annotations provided in the dataset and we computed the accuracy (num matches / num squares). There were two evaluations: a strict evaluation against the ground truth and a rotation invariant evaluation. In the orientation invariant evaluation, we compare the 4 rotated versions of the predicted chessboard against the ground truth, and we choose the orientation with the highest accuracy.

Strict evaluation: Overall accuracy of 61%.

Rotation invariant evaluation: Overall accuracy of 97%

In conclusion, if we find a robust method to get the right orientation of the board, the pipeline performs almost perfectly (97% accuracy).