Assignment: Classification and Detection of ArUco Patterns in Different Conditions

Module Name: Advanced Machine Vision for Mapping and Localization

Student name: Syed Reyadh

Abstract

ArUco patterns are a type of marker-based visual fiducial system that have gained popularity in recent years due to their simplicity, robustness, and wide range of applications. These markers consist of a black and white square pattern, with a unique code encoded in the pattern. The code can be decoded using a detection algorithm, which allows for the identification and tracking of the marker in images. ArUco patterns have a wide range of applications, including augmented reality, robotics, and camera calibration.

Despite the wide range of applications, the detection and classification of ArUco patterns in images remains a challenging task, due to factors such as lighting conditions, camera noise, and occlusion.

Introduction

In this report, two methods for classifying and detecting ArUco patterns using convolutional neural networks (CNNs) in MATLAB was proposed. Specifically, the AlexNet and Yolo networks were used, which are both known for their ability to perform image classification and detection tasks with high accuracy. The dataset is generated using the ArUco library, which allows for the creation of a variety of markers with different properties such as size, marker ID, and error correction level.

Aim: Classify all the ArUco patterns in folder 2 & 3 and detect all the ArUco patterns from file 4 & 5.

Objective:

1. Determining the key factors that affect performance and present compelling evidence to back this up.
2. Evaluating the performance of the model in various conditions such as different lighting conditions, distances, angles, number of epochs.
3. Maximizing model validation accuracy in classification.
4. Minimizing model training time.
5. Creating a method for discovering and identifying ArUco markers in the given pictures.
6. Creating a robust and efficient algorithm for detecting ArUco markers under varying lighting and environmental conditions.

Method for Classification

For classification Alexnet architecture was used because of its ability to run on devices with limited computational resources. This makes it well-suited for use in embedded systems and mobile devices, where real-time performance is crucial. Additionally, the small size and low computational complexity of AlexNet make it well-suited for this project. Compared with Googlenet the training time with Alexnet was lot faster and more accurate. One of the most important features of AlexNet is the use of rectified linear unit (ReLU) activation functions, which helped to improve the training speed and reduce the risk of overfitting. The usage of dropout regularization, which randomly drops out neurons during training to prevent overfitting was also helpful.

The architecture of AlexNet consists of eight layers: five convolutional layers, two fully connected layers, and one output layer. The first layer of AlexNet is a convolutional layer with 96 filters, which is followed by max pooling and normalization layers. The next three convolutional layers have 256, 384, and 384 filters, respectively. The fifth convolutional layer is followed by a max pooling layer and two fully connected layers with 4096 neurons each. The final output layer has 1000 neurons, corresponding to the 1000 classes in the ImageNet dataset.

Diagram for Alexnet for classification is as follows:

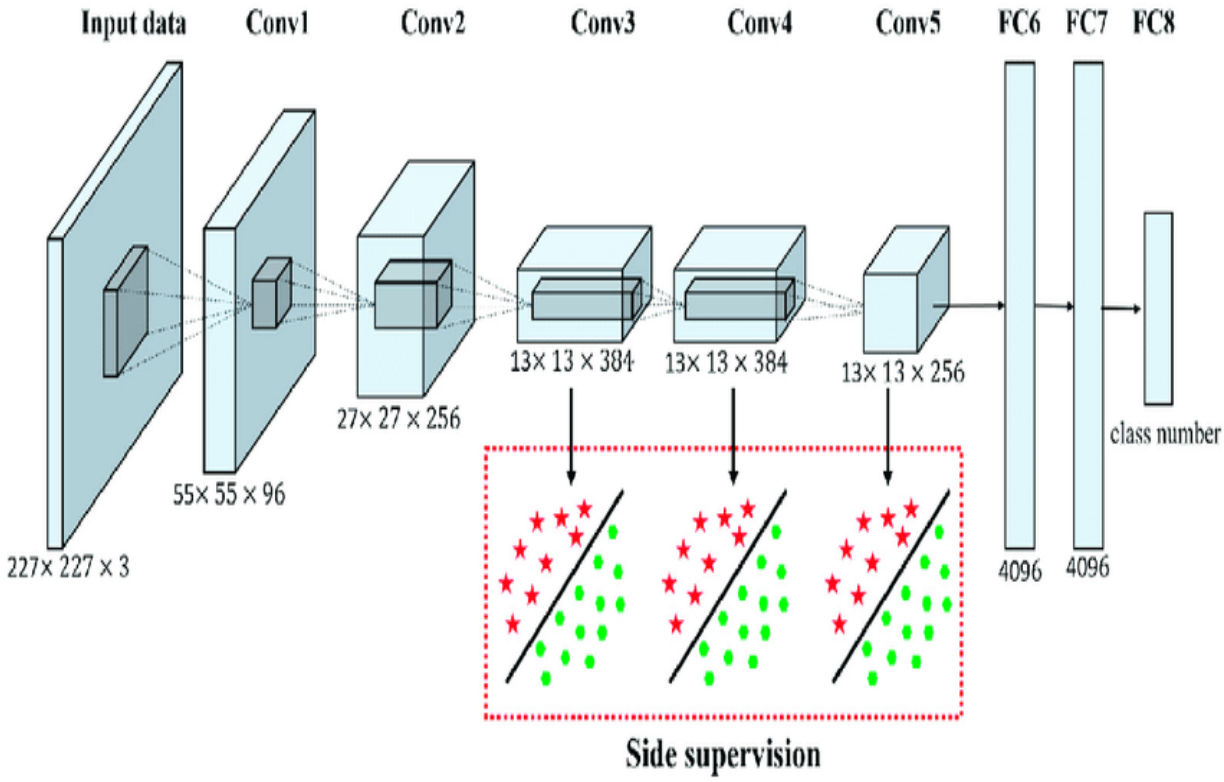

Fig 1: Alexnet classification diagram.

Overview of the process of implementing AlexNet for image classification,

1. Load the pre-trained AlexNet model.
2. Load the image wanted to classify.
3. Resize the image to match the input size of the model.
4. Preprocess the image (e.g., normalize the pixel values)
5. Pass the image through the model to get the class scores.
6. Find the class with the highest score and assign that as the predicted class.

Pseudocode for the classification method is:

- Start
- load pre-trained AlexNet model.
- img = load image
- img = resize (img, input_size)
- img = preprocess(img)
- scores = model(img)
- predicted_class = argmax(scores)
- End

A flowchart for the classification model is given below:

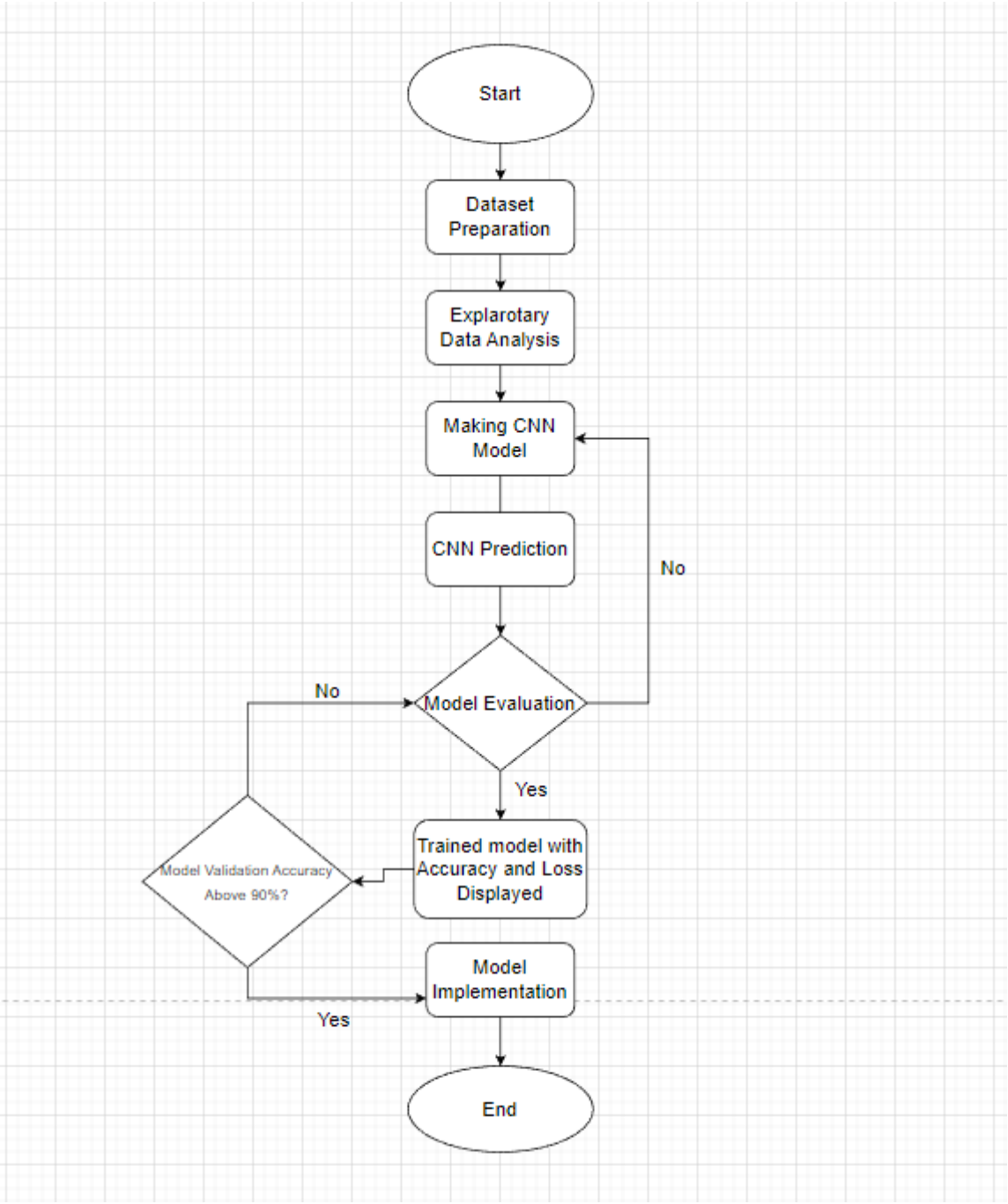

Fig 2: Flowchart

Result for Classification

On the result part couple of training options such as Maximum no. of epochs, Mini batch size Initial learning rate, shuffle, verbose etc. were considered. During 1st approach on the training option shuffle in every epoch was used and lower level of verbose was used for lower level of details in the model. This later caused extra delay in model training and for next approaches verbose and shuffle was omitted. During 1st approach only 6 epochs were used and the validation accuracy was 89.57%. The complete training time was 23 minute and 28 seconds.

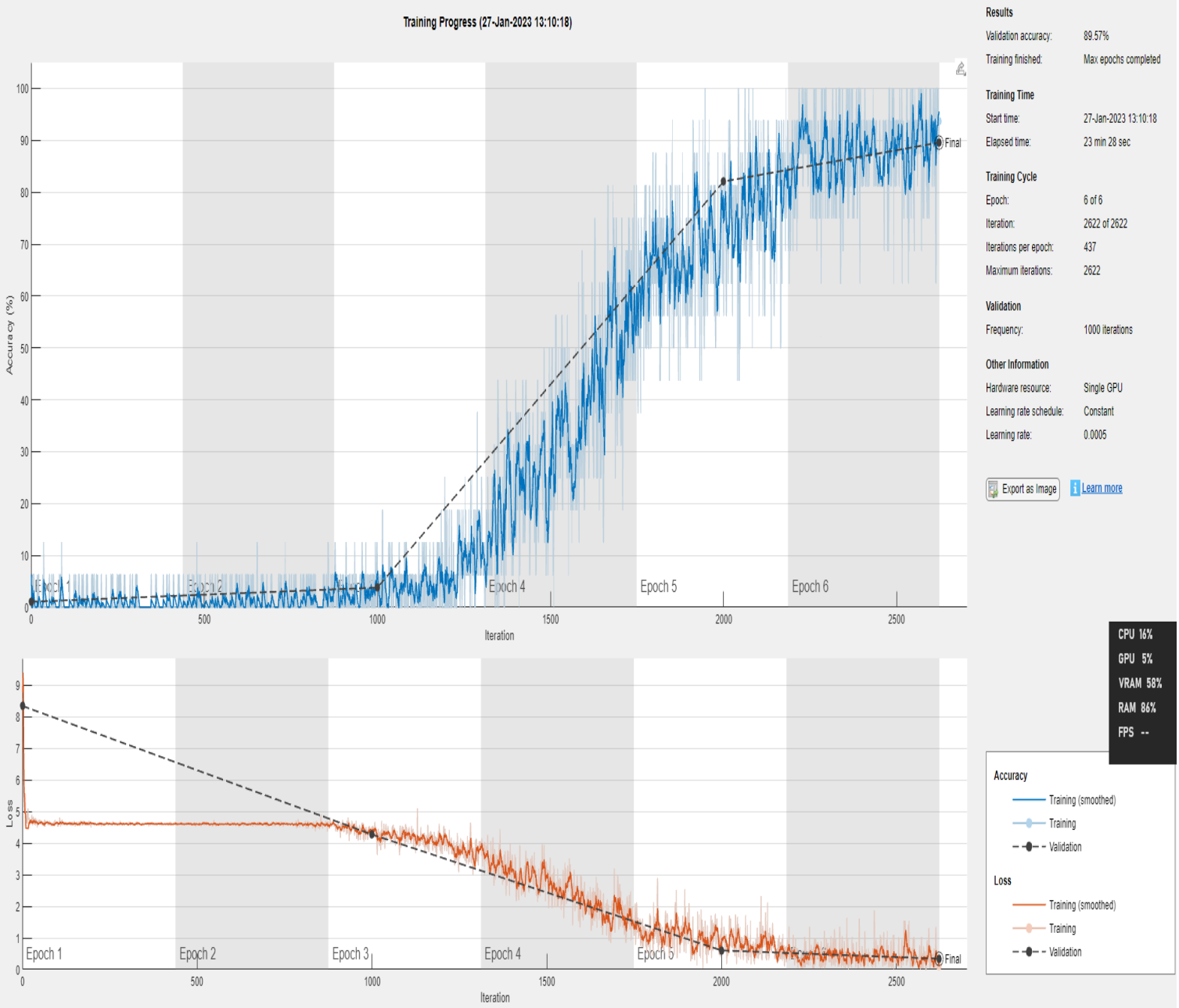

Fig 3: Trained model on 1st approach

The recognised ArUco patterns from 1st approach are shown below:

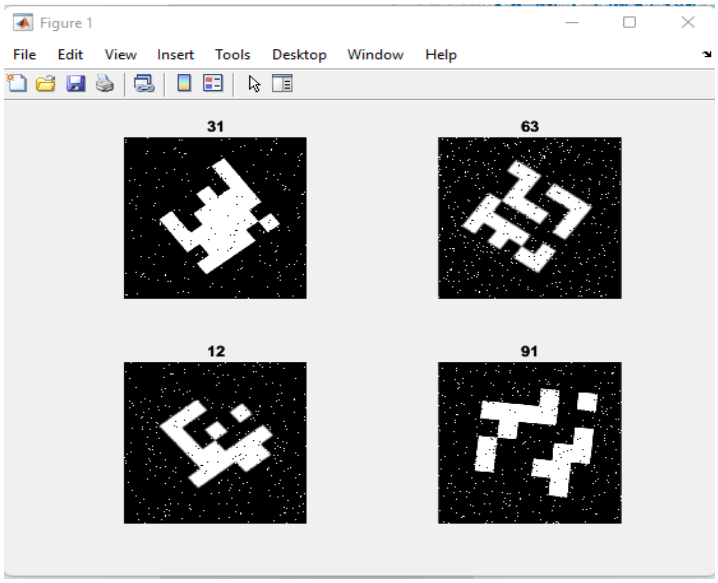

Fig 4: Classified ArUco pattern on 1st approach.

During 2nd approach verbose and shuffle were removed from the training options which helped to cut the training time. All this change caused the validation accuracy increasing from 89.57% to 95.10%. The new training complete time was 4 minutes and 51 seconds.

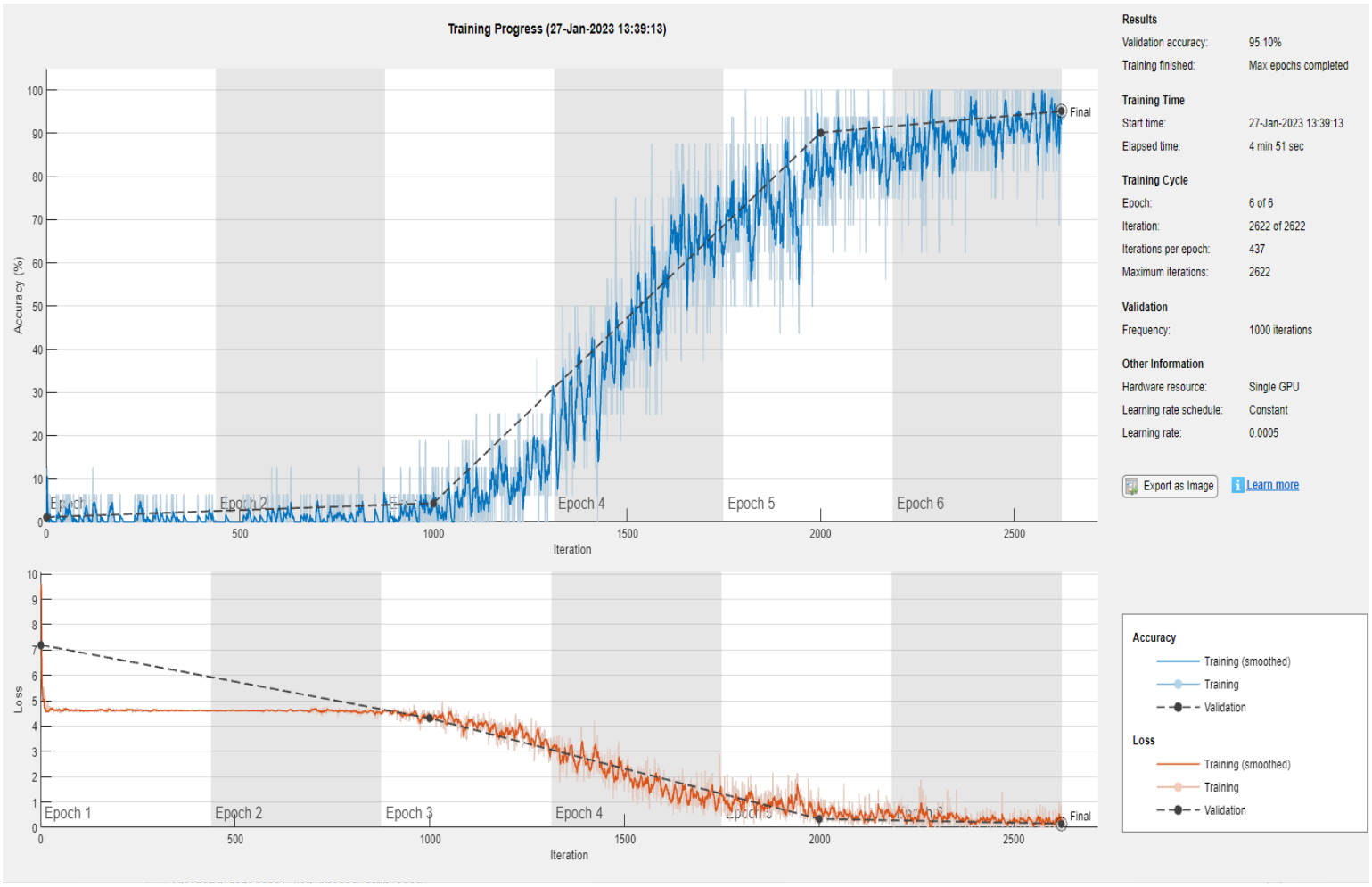

Fig 5: Trained model on 2nd approach

The recognised ArUco patterns from 2nd approach are shown below:

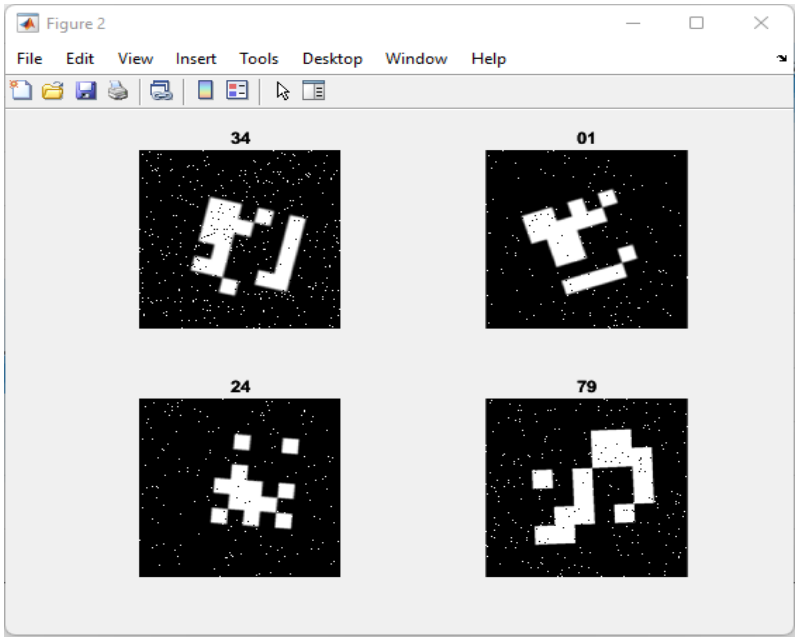

Fig 6: Classified ArUco pattern on 2nd approach.

During 3rd approach the number of Mini batch size was increased to 16 and the number of epochs were also increased from 6 to 10. Rest of the training options were kept same as 2nd approach. The result was mesmerising this time. Due to increasing epochs the complete training time was more than previous. But this time the validation accuracy was 99.77%.

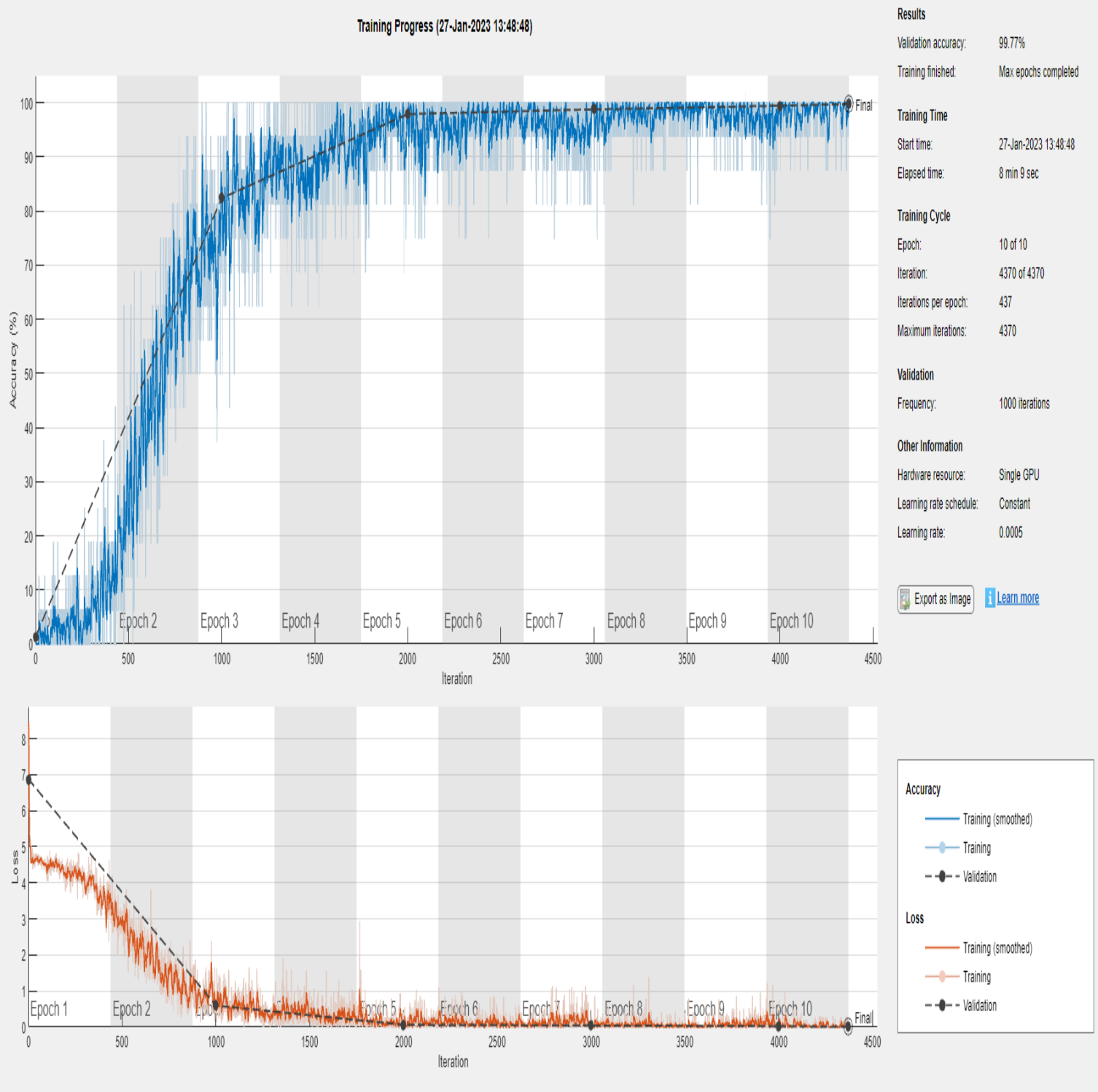

Fig 7: Trained model on 3rd approach

The recognised ArUco patterns from 3rd approach are shown below:

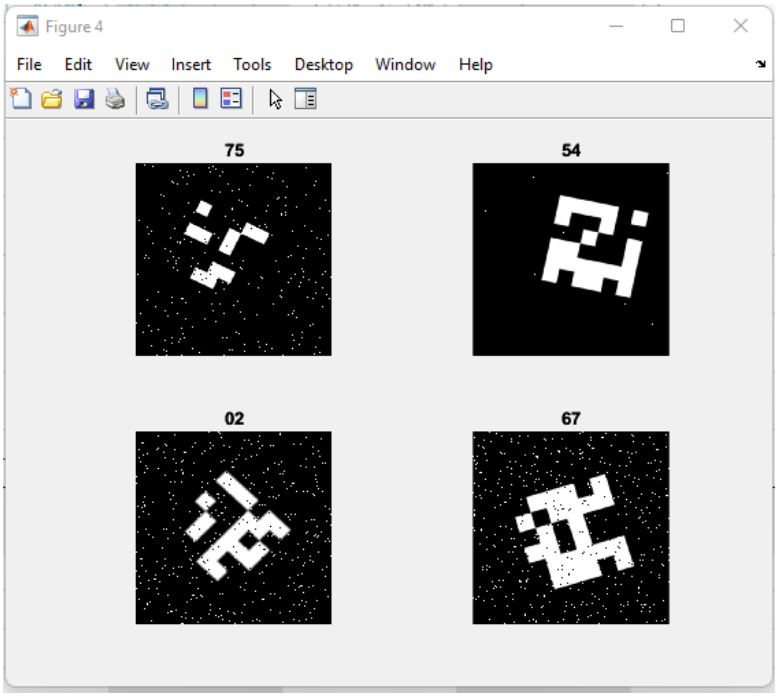

Fig 8: Classified ArUco pattern on 3rd approach.

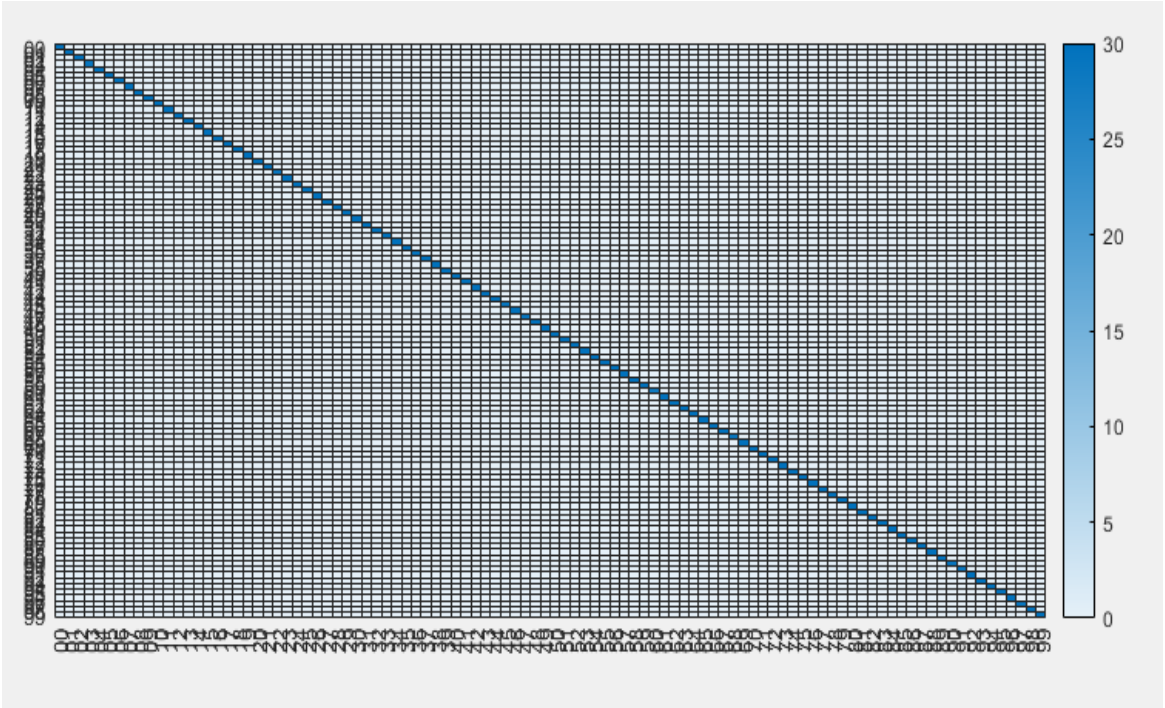

Fig 9: Confusion Matrix on 3rd approach.

First 3 approaches were experimented on the basic patterns given by UWE. Later some experiments were also done on the challenging patterns.

During 1st approach on the challenging patterns all the training options were kept same as 3rd approach of the basic patterns. The training complete time was 8:01 minutes and validation accuracy were 75.67%.

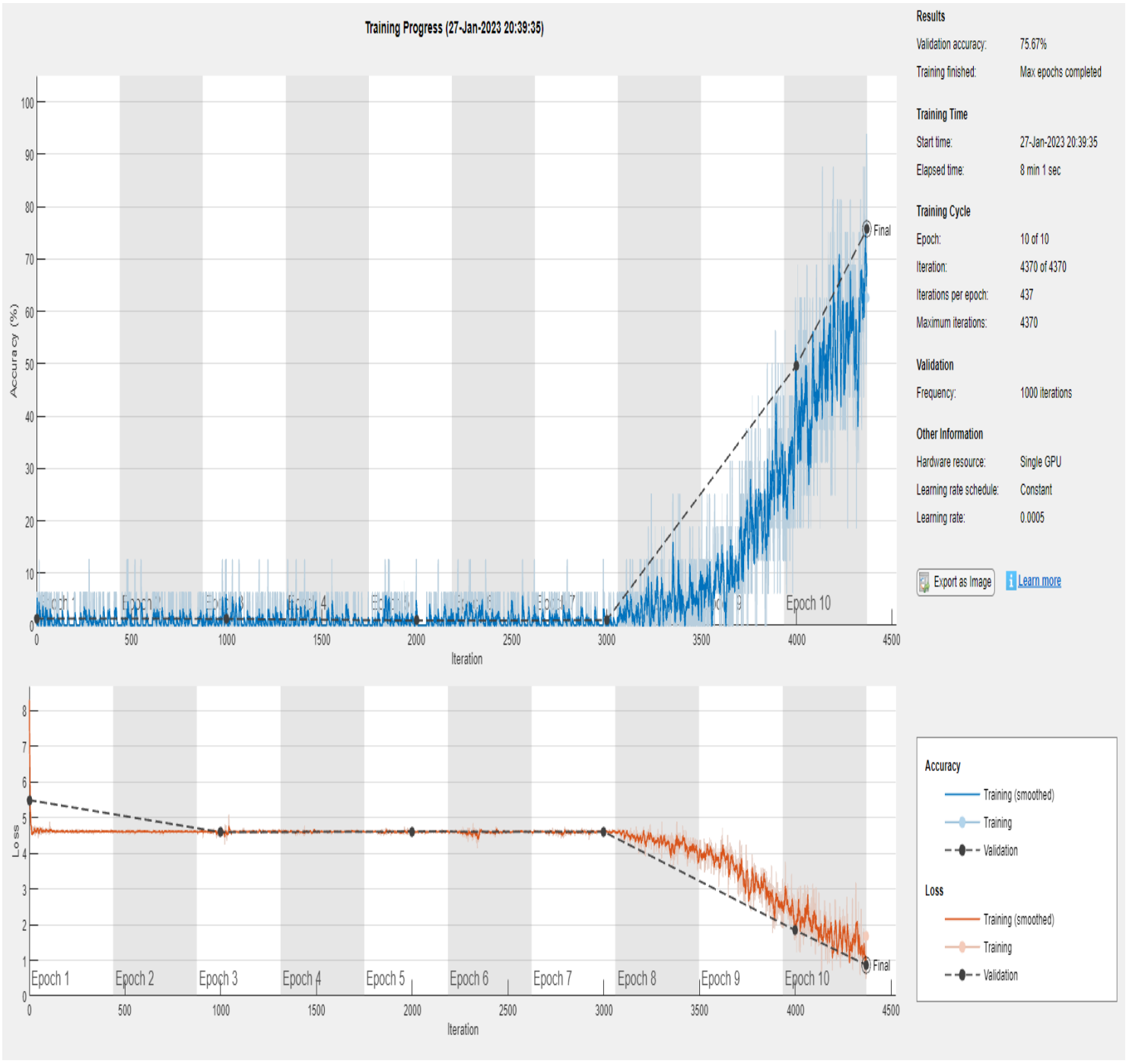

Fig 10: Trained model with challenging patterns on 1st approach

The recognised ArUco patterns from 1st approach are shown below:

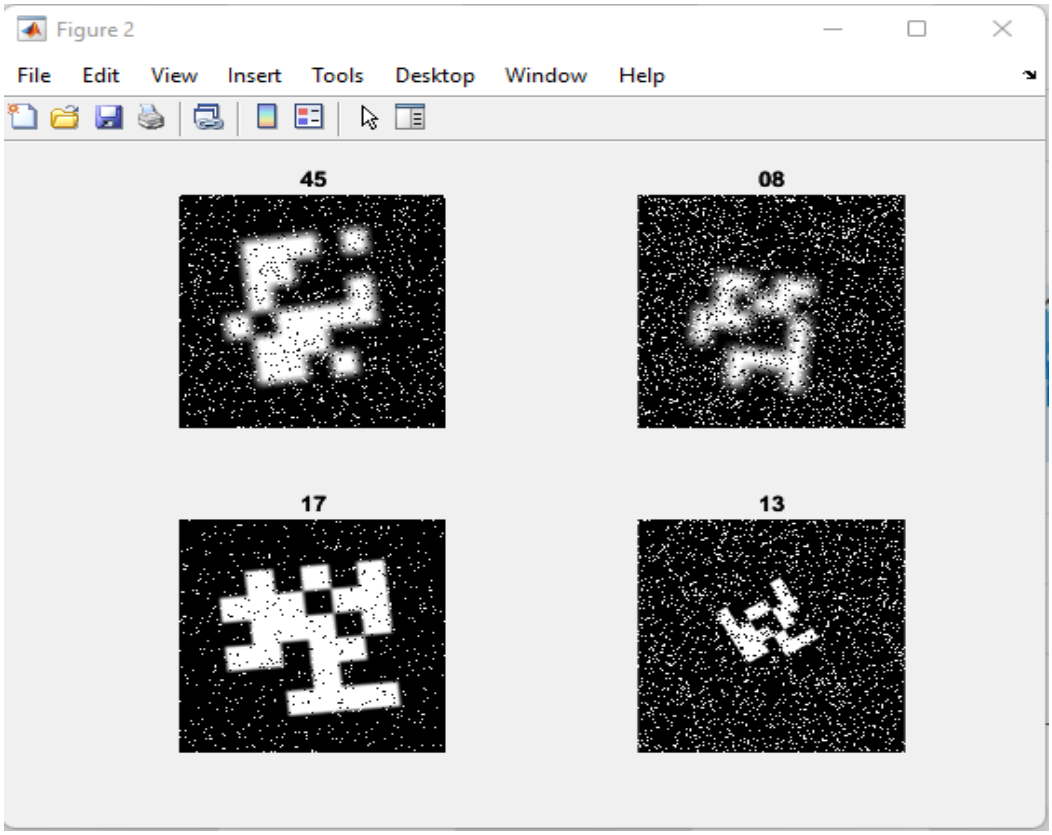

Fig 11: Classified challenging ArUco pattern on 1st approach.

After the 1st approach, on 2nd approach on the challenging patterns all the training options were kept same as 1st approach, but the mini batch size (number of images used per mini batch) was increased to 20. The training complete time was 7:25 minutes and validation accuracy were increased to 95.50%.

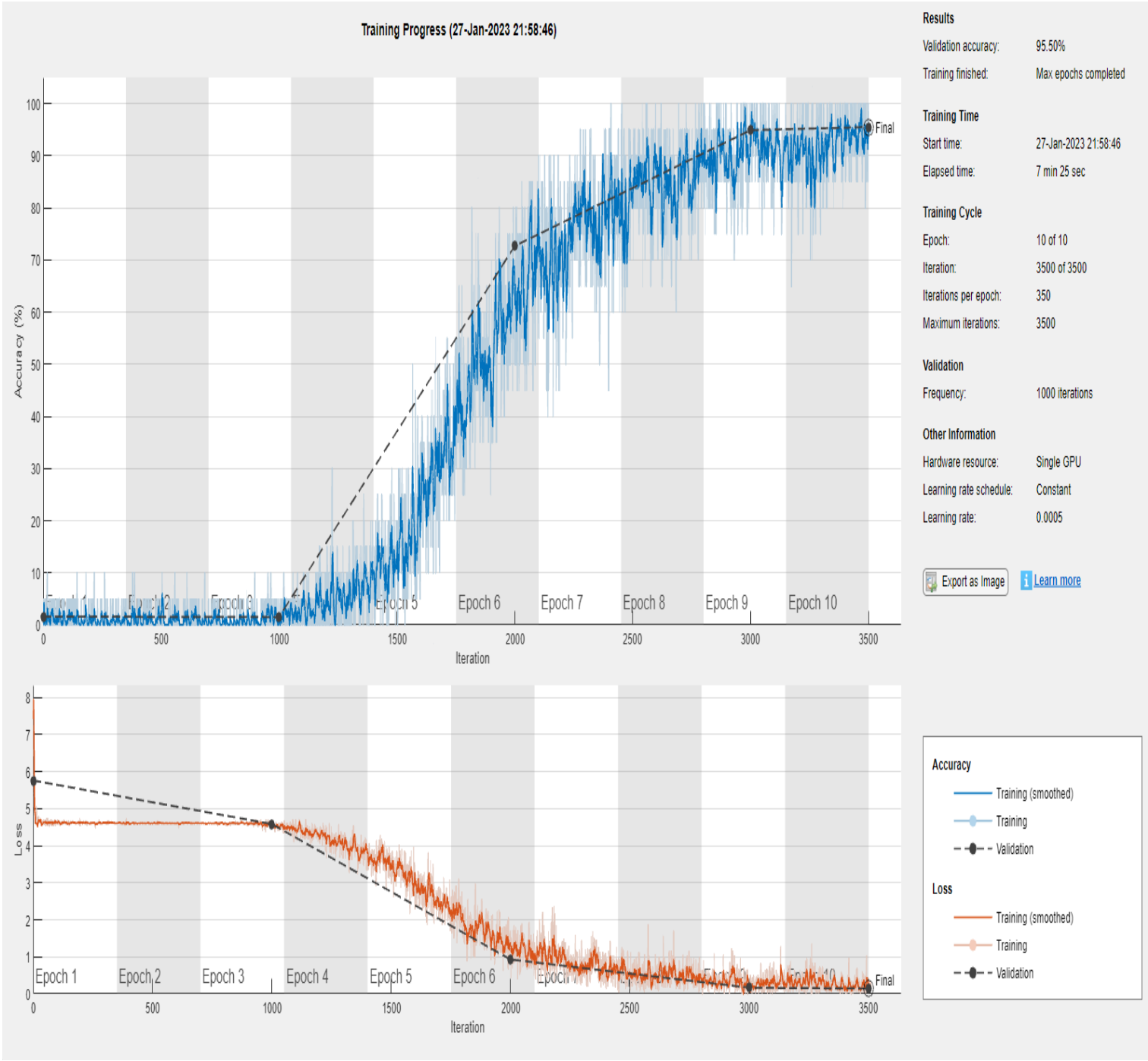

Fig 12: Trained model with challenging patterns on 2nd approach

The recognised ArUco patterns from 2nd approach are shown below:

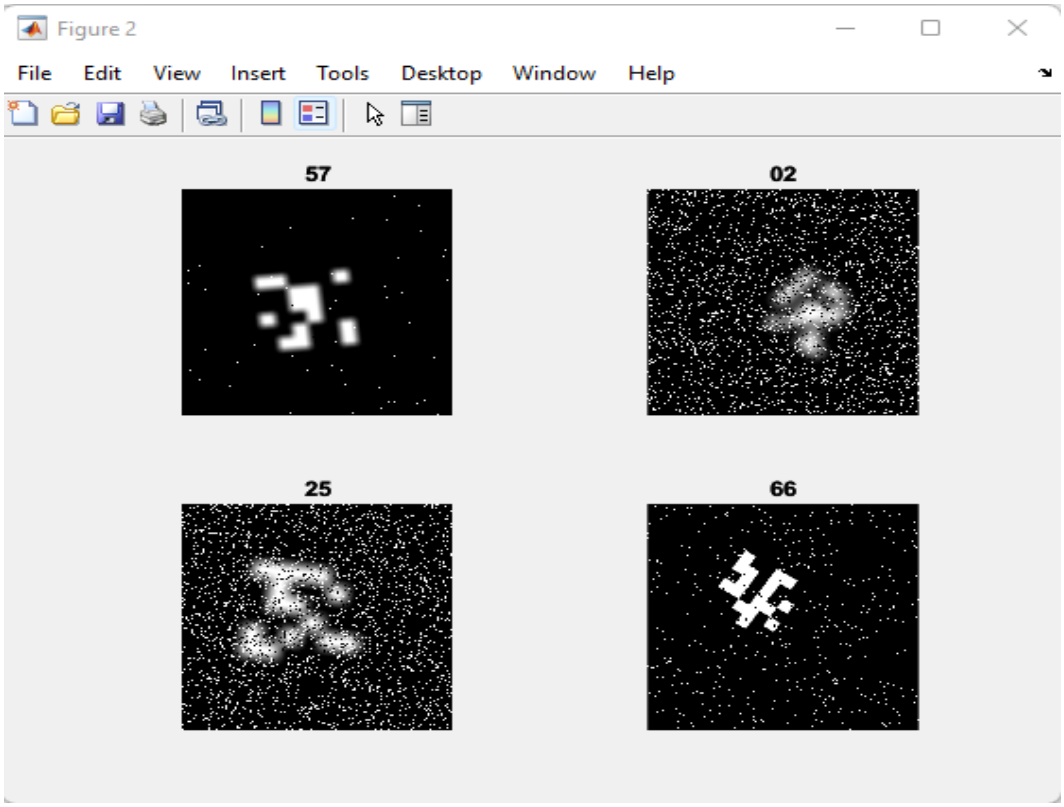

Fig 13: Classified challenging ArUco pattern on 2nd approach.

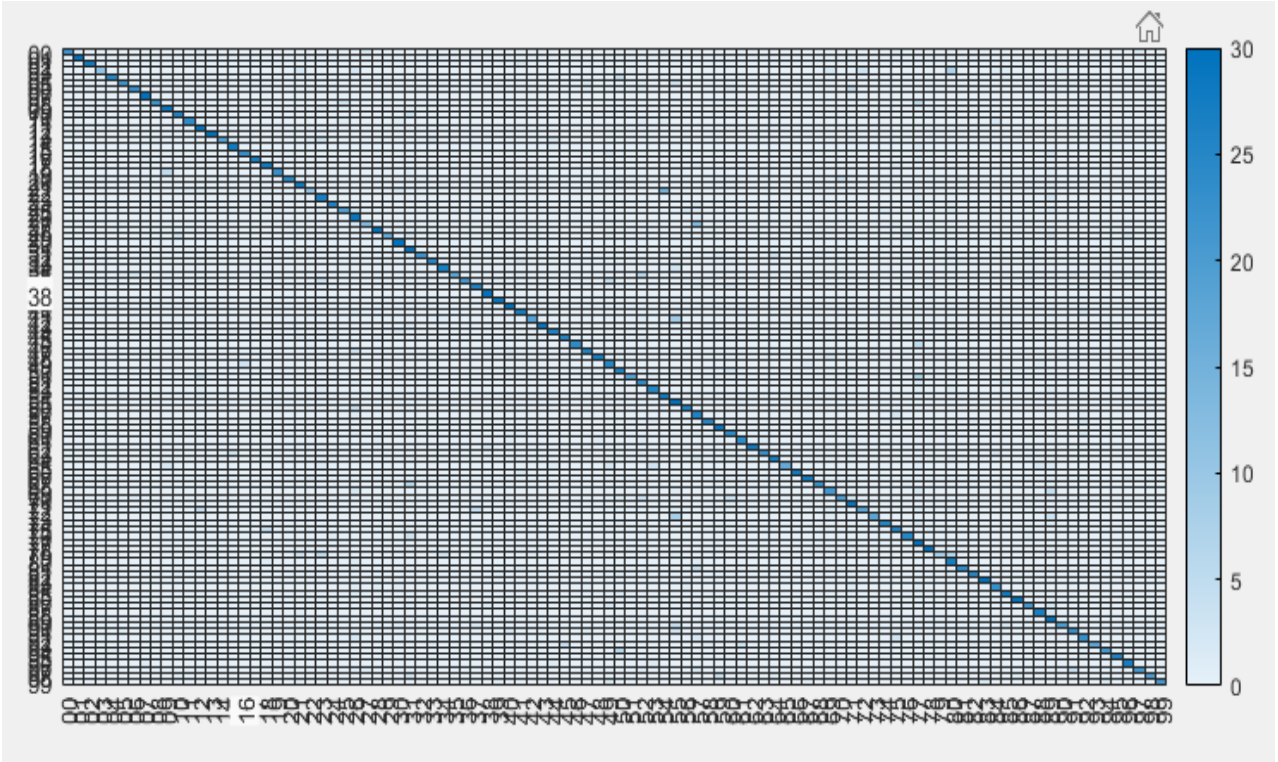

Fig 14: Confusion Matrix on 2nd approach.

Method for Detection

YOLO: You Only Look Once is referred to as YOLO informally. This method identifies and finds various things in images (in real-time). YOLO performs object detection as a regression problem and outputs the class probabilities of the discovered photos.

The reasons behind choosing this model are described below:

1. Real-time performance: YOLO is suited for applications that call for quick processing because it is built for real-time object detection.
2. Single network pass: Compared to techniques that require numerous passes, YOLO conducts detection in a single network pass.
3. End-to-end solution: YOLO is an end-to-end solution, which makes it simpler to create and use. It performs both object detection and categorization duties.
4. Large dataset compatibility: Large datasets can be used to train YOLO, which could enhance its ability to recognise ArUco patterns.
5. Robustness to scale and orientation: YOLO's detection system can handle many scales and orientations that ArUco patterns can have.

Overview of the process of implementing YOLO for image classification:

1. Load the pre-trained YOLO model.
2. Load the image wanted to detect.
3. Resize the image to match the input size of the model.
4. Preprocess the image (e.g., normalize the pixel values)
5. Pass the image through the model to get the validation scores.
6. Find the patterns with the lowest validation RMSE score.

A flowchart for the YOLO detection model is given below:

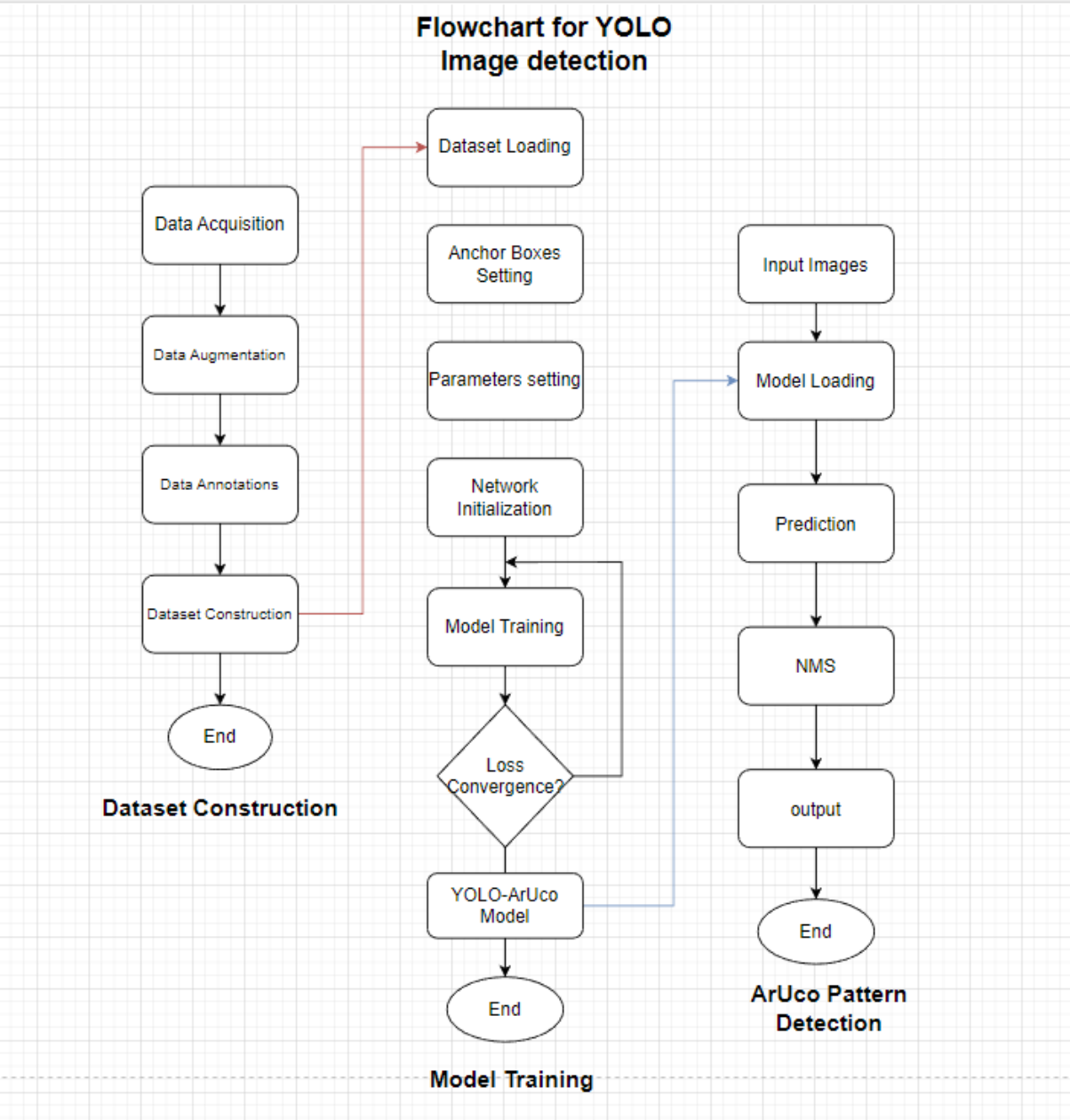

Fig 15: Flowchart

Result for Detection

The detection of the ArUco pattern from the image dataset works into two parts. First a matlab script was created to train the model in YOLO architecture and later another matlab script was created to display the detected images trained from the previous script. In this section results from both scripts were discussed.

During the first approach the model was trained with these training options assigned values in fig 16.

```
% set options
options = trainingOptions('adam',...
          'InitialLearnRate',0.002,...
          'Verbose',true,...
          'MiniBatchSize',16,...
          'MaxEpochs',100,...
          'Shuffle','every-epoch',...
          'VerboseFrequency',20,...
          'ValidationFrequency',10,...
          'CheckpointPath',tempdir,...
          'Plots','training-progress',...
          'ValidationData',validationData,...
          'ExecutionEnvironment','gpu');
```

Figure 16: Training option element values

The taken time to train the whole model was 9:38 minutes and the validation RMSE was 0.75789 (Fig 17). The main goal is to get the validation RMSE number as low as possible. In this approach the validation data was over lapping with root mean square loss and normal loss. The main goal is also to cut down the amount of over lapping.

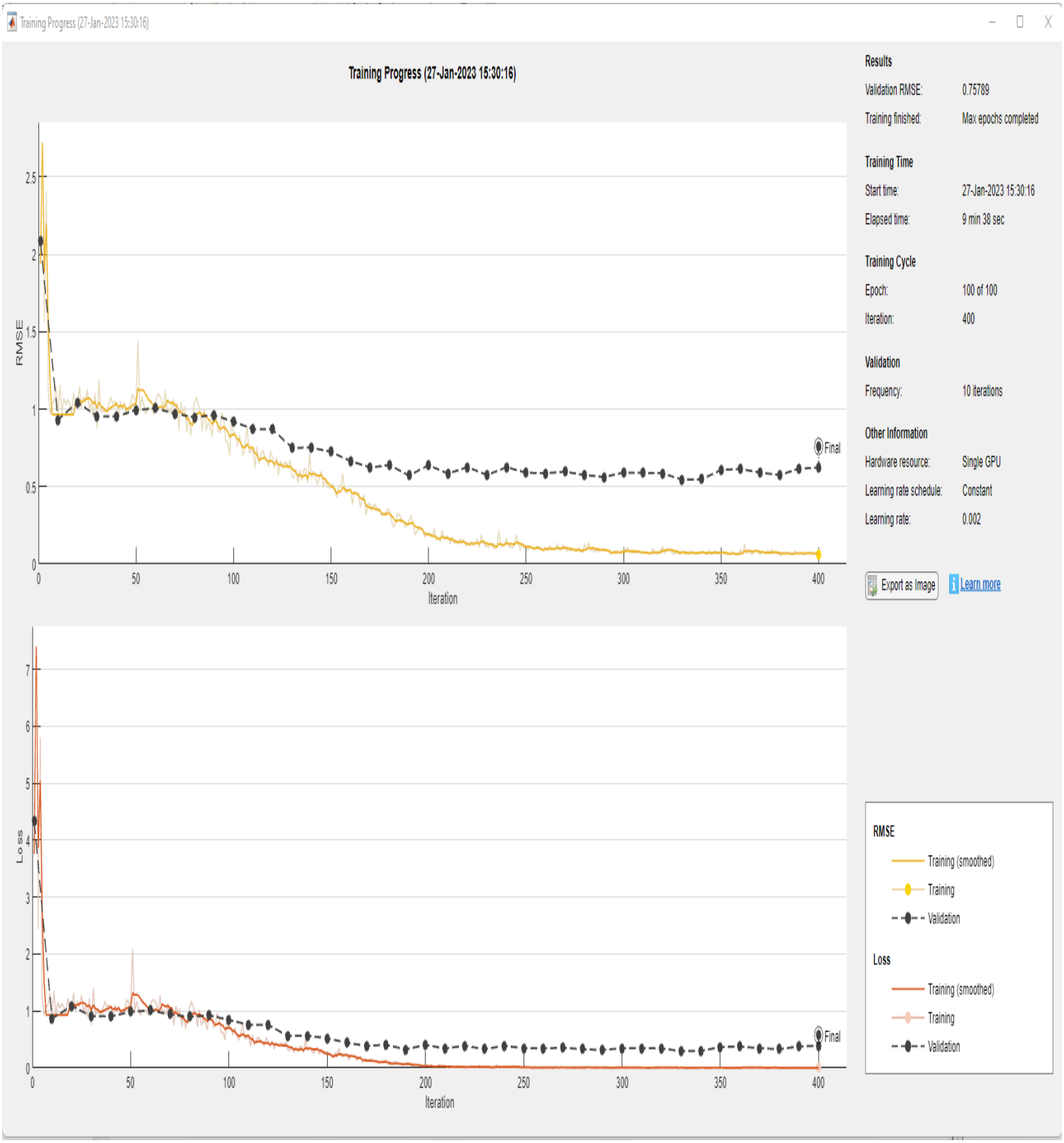

Fig 17: Training data from 1st approach.

All the detected ArUco patterns from the first approach among these 8 pictures are shown below:

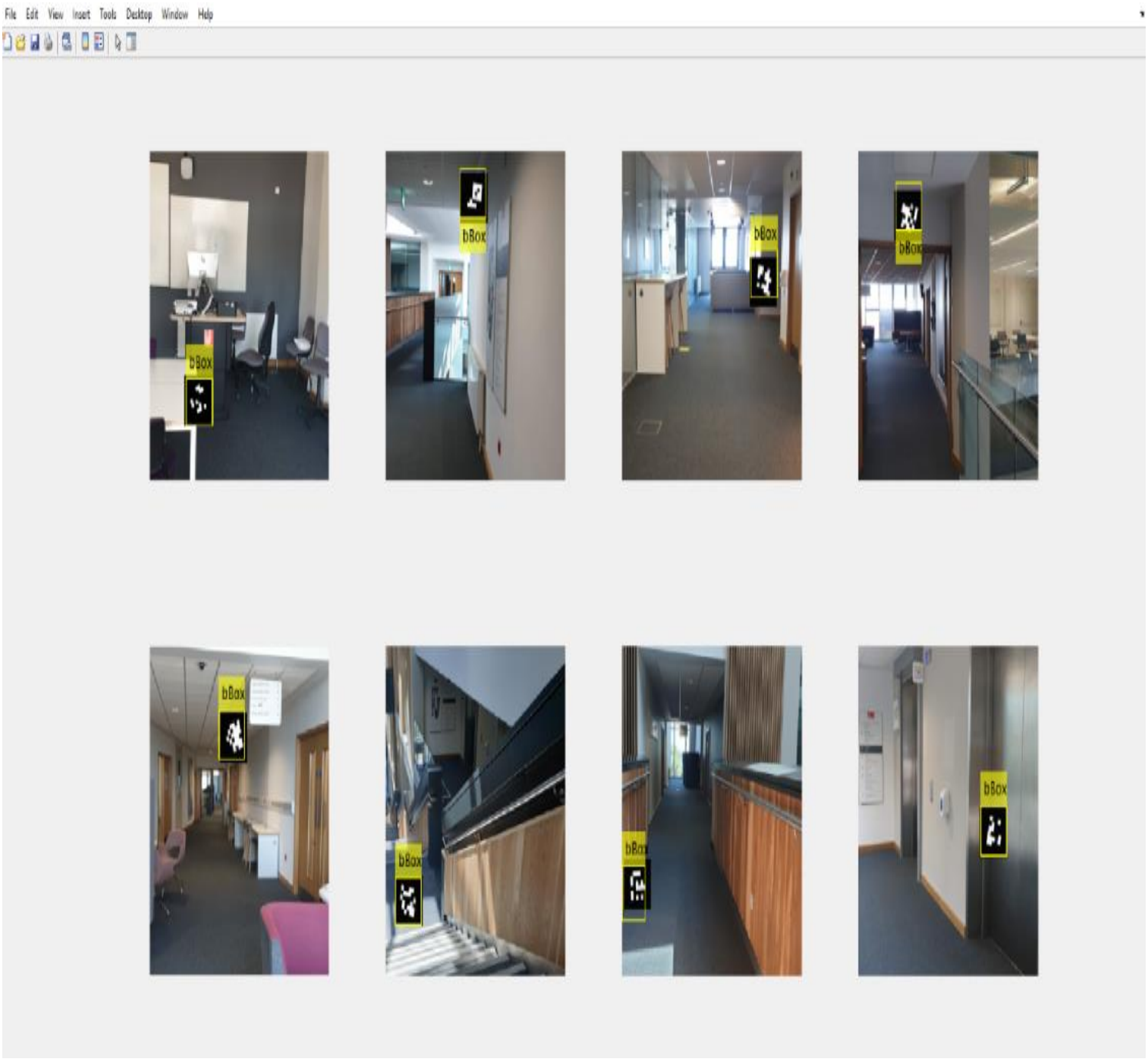

Fig 18: Detected images from the image dataset on 1st approach.

During the second approach the model was trained with these training options assigned values in fig 19. Mini batch size number was changed in this approach.

```
% set options
options = trainingOptions('adam',...
          'InitialLearnRate',0.002,...
          'Verbose',true,...
          'MiniBatchSize',20,...
          'MaxEpochs',100,...
          'Shuffle','every-epoch',...
          'VerboseFrequency',20,...
          'ValidationFrequency',10,...
          'CheckpointPath',tempdir,...
          'Plots','training-progress',...
          'ValidationData',validationData,...
          'ExecutionEnvironment','gpu');
```

Figure 19: Training option element values

The taken time to train the whole model was 4:43 minutes and the validation RMSE was 0.38033 (Fig 20). The training time had decreased and the RMSE value was lower in this approach.

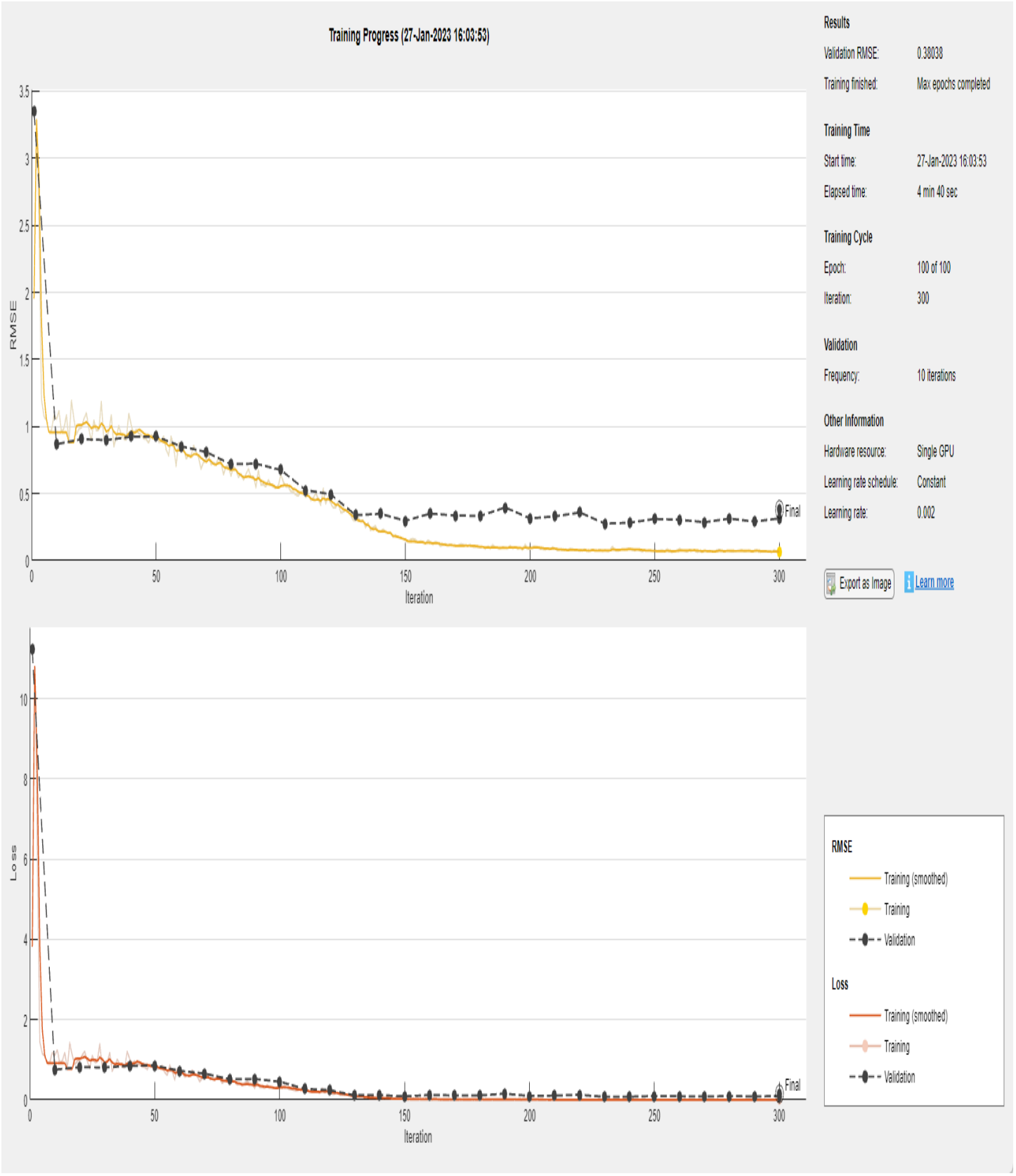

Fig 20: Training data from 2nd approach.

All the detected ArUco patterns from the second approach among these 8 pictures are shown below:

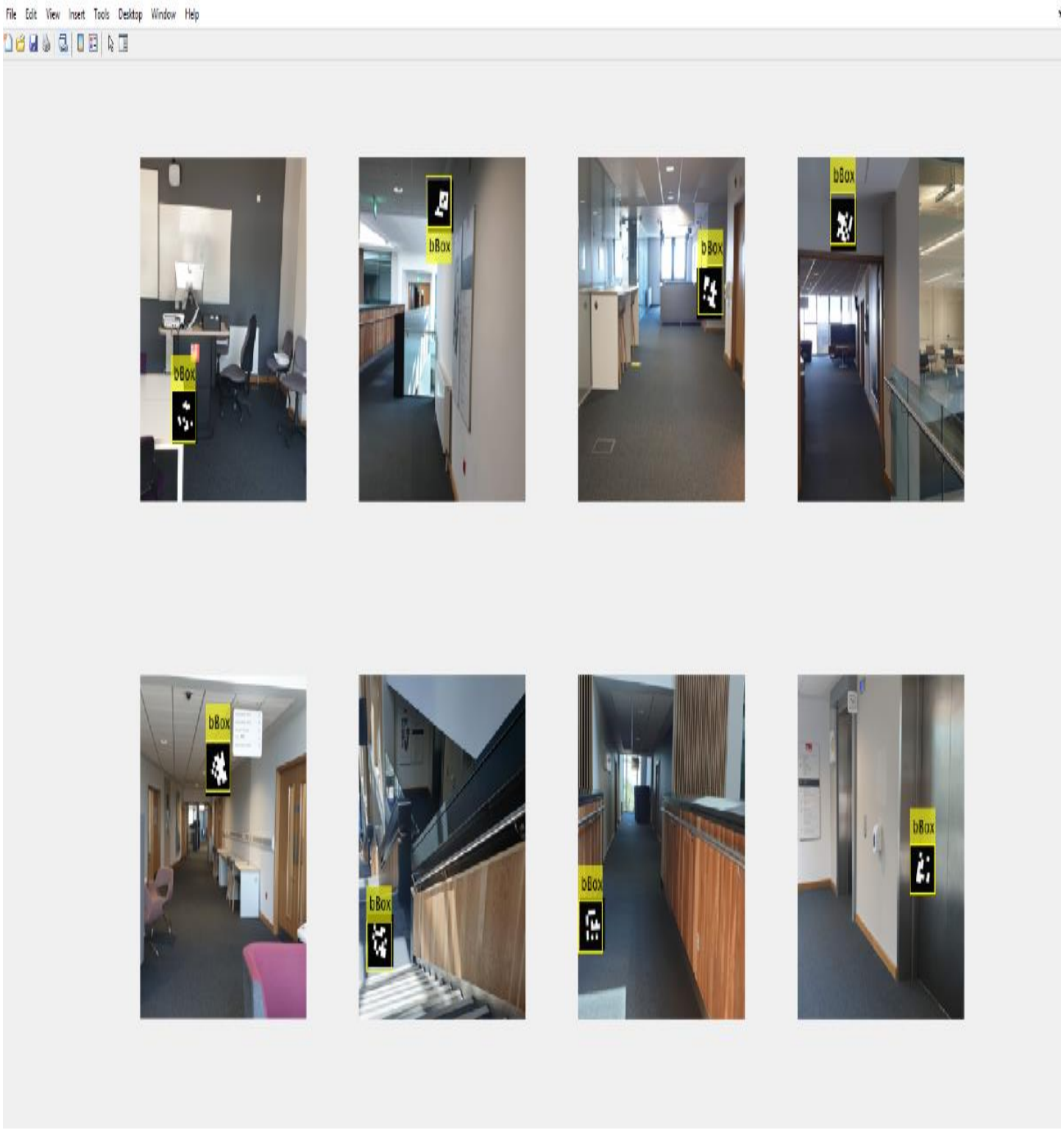

Fig 21: Detected images from the image dataset on 2nd approach.

During the third approach the model was trained with these training options assigned values in fig 22. This time the max number of epochs were changed.

```
% set options
options = trainingOptions('adam',...
          'InitialLearnRate',0.002,...%%
          'Verbose',true,...
          'MiniBatchSize',20,...%%
          'MaxEpochs',120,...%%
          'Shuffle','every-epoch',...
          'VerboseFrequency',20,...
          'ValidationFrequency',10,...%%
          'CheckpointPath',tempdir,...
          'Plots','training-progress',...
          'ValidationData',validationData,...
          'ExecutionEnvironment','gpu');
```

Figure 22: Training option element values

The taken time to train the whole model was 5:53 minutes and the validation RMSE was 0.38243 (Fig 23). The training time had increased and the RMSE value was bit higher in this approach.

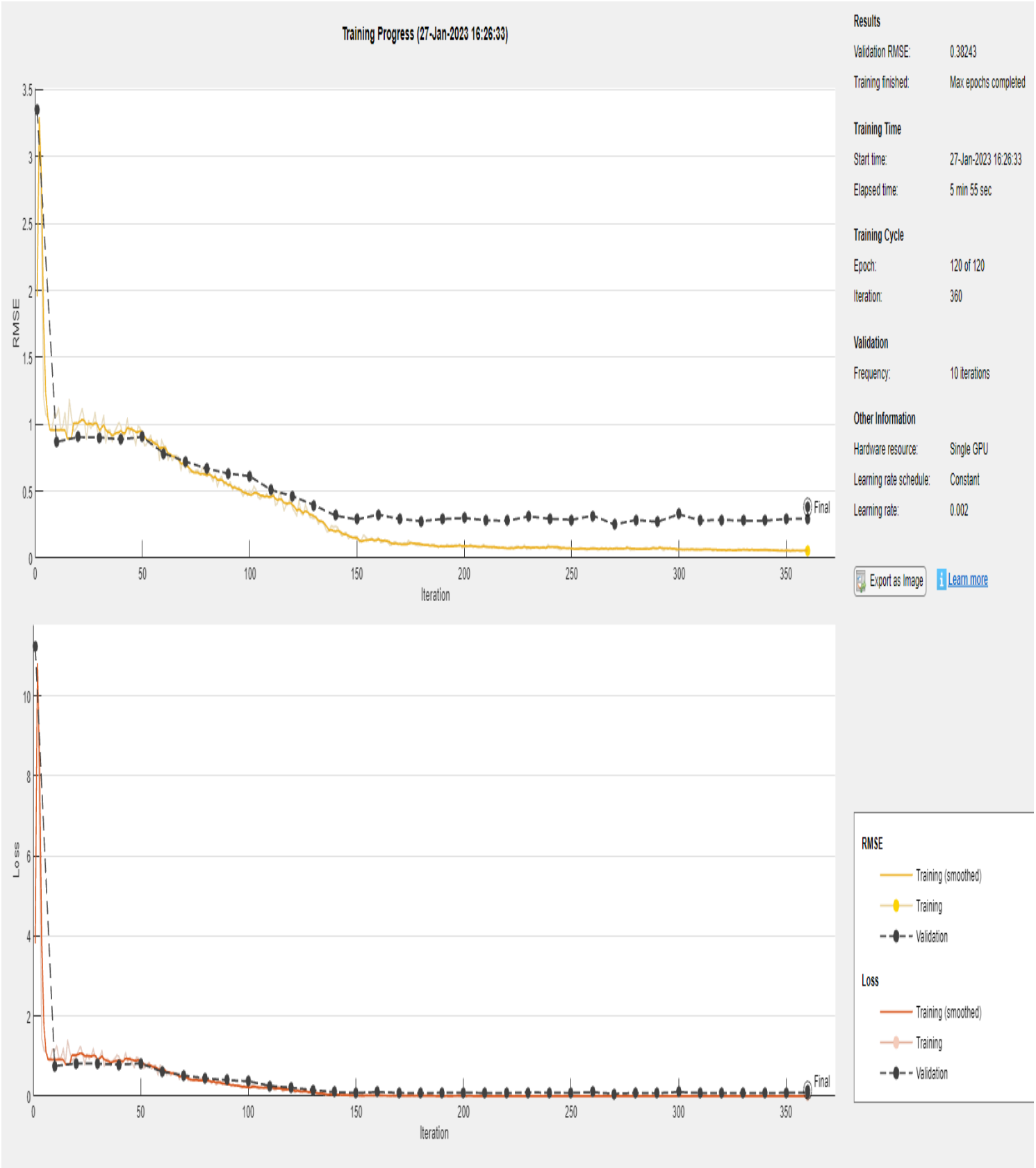

Fig 23: Training data from 3rd approach.

All the detected ArUco patterns from the third approach among these 8 pictures are shown below:

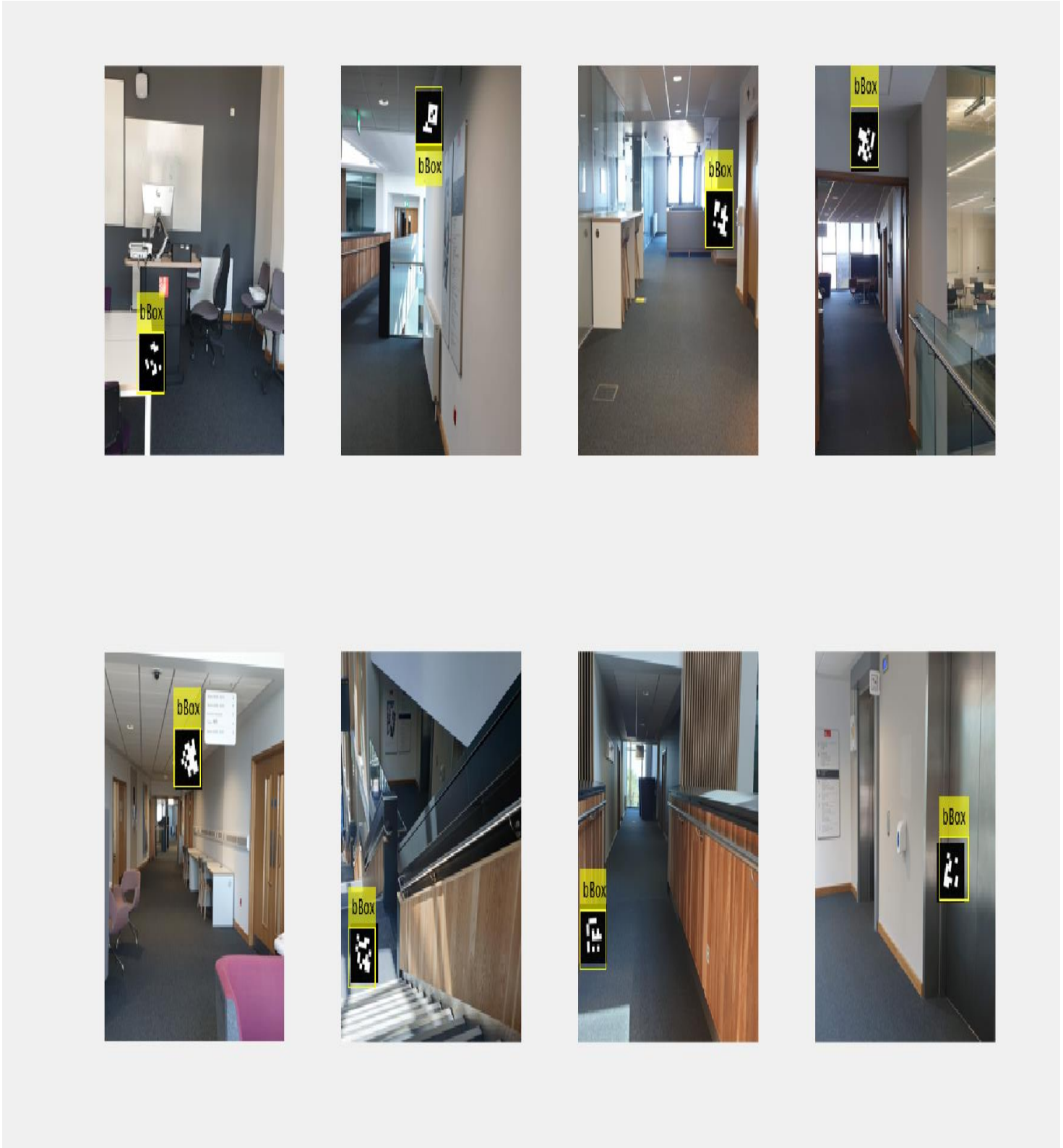

Fig 24: Detected images from the image dataset on 3rd approach.

During the fourth approach the model was trained with these training options assigned values in fig 25. The max number of epochs, mini batch size and validation frequency were changed.

```
% set options
options = trainingOptions('adam',...
          'InitialLearnRate',0.002,...%%
          'Verbose',true,...
          'MiniBatchSize',8,...%%
          'MaxEpochs',50,...%%
          'Shuffle','every-epoch',...
          'VerboseFrequency',20,...
          'ValidationFrequency',50,...%%
          'CheckpointPath',tempdir,...
          'Plots','training-progress',...
          'ValidationData',validationData,...
          'ExecutionEnvironment','gpu');
```

Figure 25: Training option element values

The taken time to train the whole model was 2:35 minutes and the validation RMSE was 0.11547 (Fig 26). The training time had decreased and the RMSE value was the lowest among all the experiments.

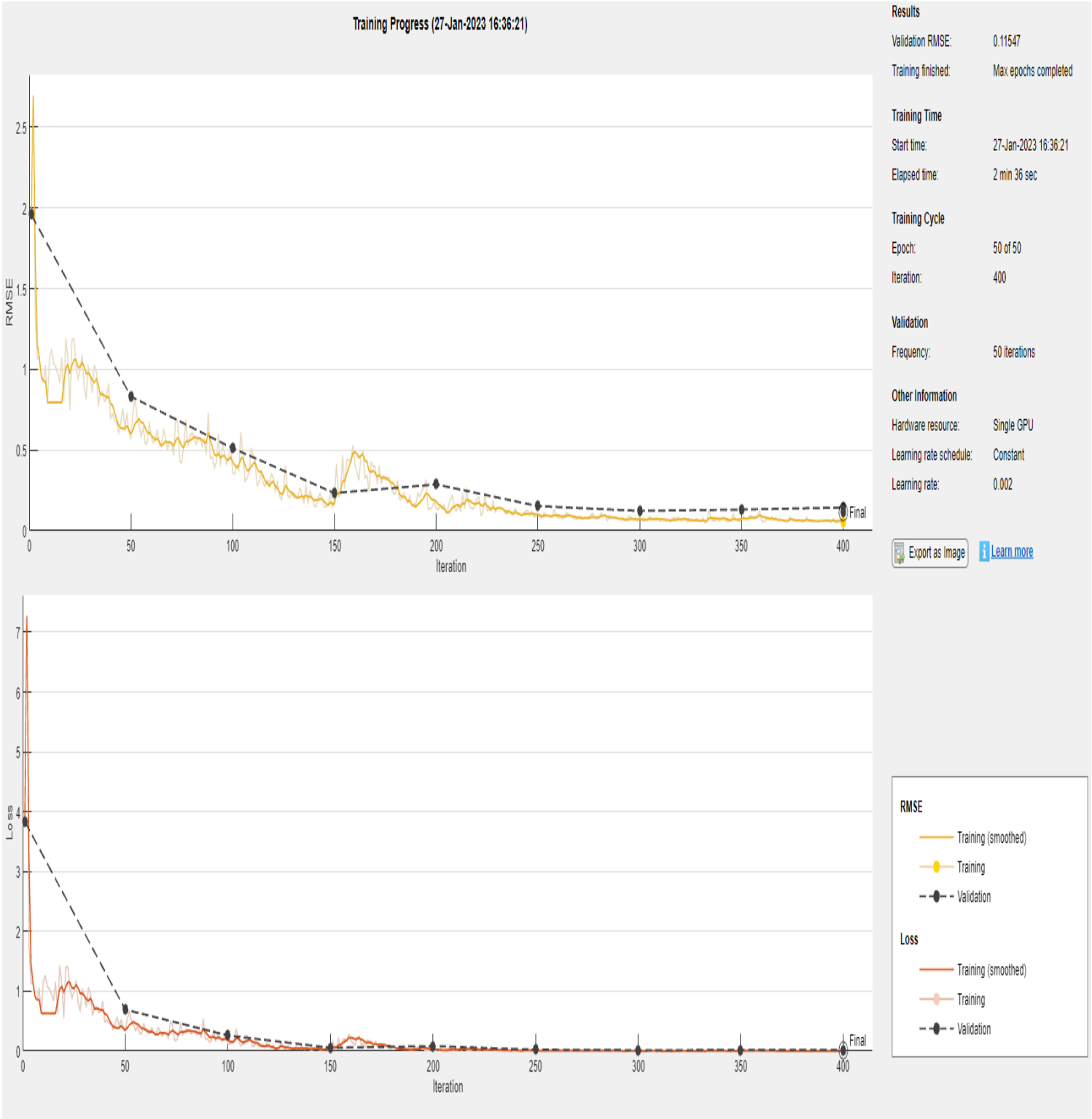

Fig 26: Training data from 4th approach.

Due to lower RMSE rate the detected boxes were fitted more accurately than the previous approaches. All the detected ArUco patterns from the third approach among these 8 pictures are shown below:

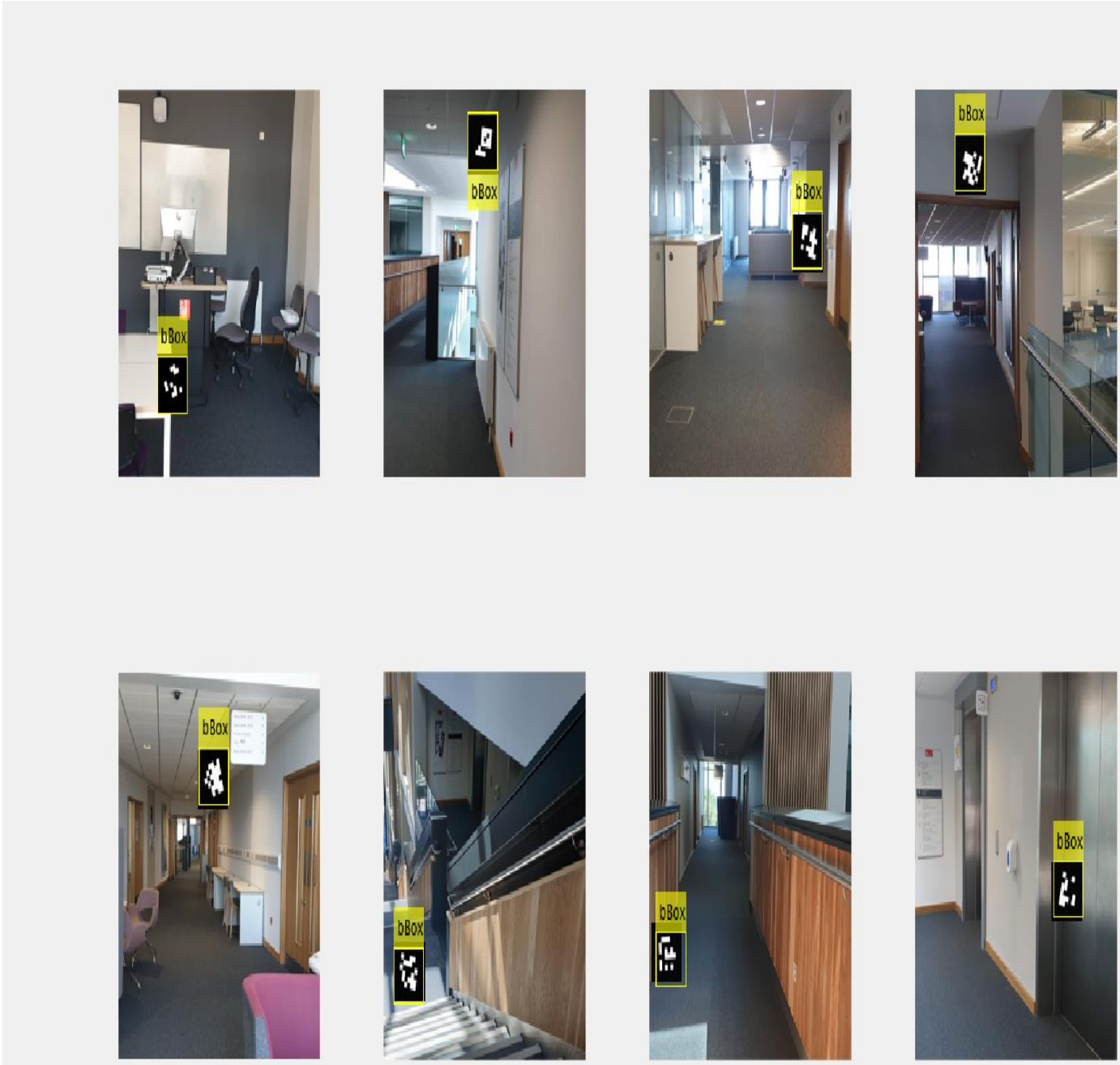

Fig 27: Detected images from the image dataset on 4th approach.

Conclusion

Since the computer utilised for this assignment was slow for training the network, the categorization and detection of ArUco tags were effective and that all the findings were of a respectable level. The project used a combination of computer vision and machine learning techniques to classify and detect ArUco patterns. Important findings include the capacity to accurately identify and classify ArUco patterns in different conditions and detecting them from training image sets. Both Basic and Challenging distorted images were classified. On 2500 photos, the Basic model correctly classified 89.57%, 95.10%, and 99.77% of the images. As for the Challenging distorted images accuracy was 75.67% and 95.50% correct classification which was also tested on 2500 images.

Some key areas for improvement or future research in this project include:

- Enhancing the ArUco pattern identification algorithm's adaptability to changes in lighting, camera angle, and marker orientation.
- Improving classification accuracy by incorporating increasingly sophisticated machine learning methods, such deep learning.
- Examining the application of numerous ArUco patterns in a single image or video frame to enhance the system's overall accuracy and robustness.

Reference

Organisation: GreatLearning.
Title of Website: AlexNet: The First CNN to win Image Net.
Available from: https://www.mygreatlearning.com/blog/alexnet-the-first-cnn-to-win-image-net/
[Accessed 21/10/2022]

Organisation: MathWorks.
Title of Website: alexnet.
Available from: https://uk.mathworks.com/help/deeplearning/ref/alexnet.html
[Accessed 27/10/2022]

Organisation: MathWorks.
Title of Website: Transfer Learning Using Alexnet
Available from: https://uk.mathworks.com/help/deeplearning/ug/transfer-learning-using-alexnet.html
[Accessed 28/10/2022]

Organisation: Deploy Containers Close to Your Users.
Title of Website: Introduction to YOLO Algorithm for Object Detection.
Available from: https://www.section.io/engineering-education/introduction-to-yolo-algorithm-forobjectdetection/#:~:text=What%20is%20YOLO%3F,probabilities%20of%20the%20detected%20images.
[Accessed 20/11/2022]

Organisation: MathWorks.
Title of Website: Object Detection.
Available from:
https://in.mathworks.com/help/vision/objectdetection.html?s_tid=CRUX_lftnav
[Accessed 12/12/2022]

Organisation: MathWorks.
Title of Website: Object Detection Using YOLO v2 Deep Learning.
Available from: https://uk.mathworks.com/help/vision/ug/object-detection-using-yolov2-deep-learning.html
[Accessed 15/12/2022]

Organisation: MathWorks.
Title of Website: Deep Learning Object Detector.
Available from:https://uk.mathworks.com/help/vision/ref/deeplearningobjectdetector.html
[Accessed 20/12/2022]

Organisation: MathWorks.
Title of Website: trainACFObjectDetector.
Available from: https://uk.mathworks.com/help/vision/ref/trainacfobjectdetector.html
[Accessed 22/12/2022]