uc3m | Universidad Carlos III de Madrid

Bachelor in Computer Science and Engineering

2021-2022

Bachelor Thesis

"Chess piece location and identification using Machine Learning"

Pablo Tomás Campos Fernández

Supervisor

Antonio Berlanga de Jesús

Madrid, Spain, June 22th 2022

ABSTRACT

This project aims to identify and locate chess pieces from a chess board photograph to recreate the chess board position in digital format. To do so, several approaches will be tested to challenge previous limitations and requirements for this task to overcome them and achieve a simpler and more flexible solution. Overcoming said limitations are of great importance, as doing so will increase accessibility to chess recognition models, moving one step closer to end-user making automatic digital recreations of their over-the-board chess games.

Keywords: Deep learning, deep neural networks, deep convolutional neural networks, transfer learning, residual neural networks, confusion matrices and bird eyes view.

DEDICATION

This project would have never been possible if it were not for the incredible Computer Science Engineering professors at Universidad Carlos III de Madrid. A special thanks is due to my mentor, Antonio Berlanga. To my family and friends who have proved to be an endless source of support, including Irene Fluxá, Enrique Benvenutto, Fernando Bermúdez, Jorge Ríos, Javier Cruz, and all the precious people that I was so grateful to meet four years ago as a freshman. Thank you all for your ideas, criticism, and memorable moments.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

1.1. Contextualization

Covid-19 changed every aspect of our lives, chess being no different. As international tournaments transitioned from over-the-board games to an online format, the benefits of the latest format became apparent. Not only did the games become easier to stream and view, but also to analyze. As the games were already played in a digital format it was much easier to feed the positions to chess engines to analyze them. With the chess scene slowly returning to over-the-board games, could there be a way to preserve the best from both worlds?

This work aspires to build, as the title of this project may suggest, a chess piece classifier and locator so that a bird's eyes image of a chess position can be converted to its equivalent in digital format. Several methods used in the classification and location tasks will be discussed to find the most suitable approach.

As for the classification task, the results obtained by three different machine learning models will be compared as the training data increases. These models will be trained and evaluated with different chess sets in order to gauge their generalization power between different artistic styles in chess sets. The best model will then be used for the location task, where the chess board will be broken up into 64 squares, each one of them passing as input to the classification model.

1.2. Objectives

This project has three objectives. First, to evaluate the performance of classification models on a chess set not seen during training. This way, the model's capacity to identify the key features of a chess piece, no matter its art style, will be evaluated.

Second, to compare the results on performance of reducing the amount of labels of a multiclass model, more concretely identifying if it is necessary for the classification model to identify the color of the chess piece as well as the piece itself, or if the color can be inferred independently from the type of chess piece.

Lastly, to use the models from the second objective to effectively classify a chess piece in its correct position given a bird's eye view of a chess board fulled with pieces.

1.3. Code

The code related to this project is available in an online repository[1] published by the author under the Creative Commons Attribution 4.0 International license.

1.4. Document Structure

This document is composed of four main areas: introduction, development, evaluation, and conclusion. The introduction provides the reader with all the contextual information necessary to understand the project. This includes previous takes on the project, how this work intends to deviate from them to evaluate newer approaches, and the methodologies used in the process.

Second, the development section aims to illustrate the journey of the project, from creating the dataset to training the models. The functionality of the algorithms and models used to achieve the objectives proposed in section 1.2 will be explained in this section.

Following up with evaluation, this section focuses on comparing and evaluating the results of the different approaches tested to find the best approach for this task.

Lastly, a conclusion is most necessary to emphasize the results obtained, explaining how their benefits and drawbacks. Afterward, the section lays a road map for future work and improvement, focusing on the weaknesses of the project and highlighting accessibility.

1.5. Legal Framework

The legal framework for this project relates to software tools, intellectual property, and data privacy compliance.

1.5.1. Software

The following software tools and platforms were used during the development of this project:

- **Python**: A general-purpose high-level interpreted programming language. Created by Guido van Rossum, it is now maintained by the Python Software Foundation. Released under the Python Software Foundation License.
- **Scikit-learn**: A free software machine learning library created by David Cournapeau for the Python programming language. Released under the New BSD License.

[1] https://github.com/PaburoTC/Bachelor-Thesis

- **PyTorch**: An optimized tensor library for deep learning using GPUs. Developed and maintained by Facebook since 2016, it is licensed under a 3-Clause BSD license.
- **FastAI**: A high-level open-source library for developing neural networks. Created and maintained by the non-profit organization with the same. Released under the Apache License 2.0.
- **Google Colaboratory**: a platform hosted by Google which freely offers online Jupyter notebooks with access to hardware acceleration such as GPUs or TPUs.

This project and all the tools used for it are suitable for non-commercial research.

1.5.2. Intellectual Property

The intellectual property of the following project belongs to the author, as it has been developed without the intervention and resources of any public or private entity. Nevertheless, it is strongly encouraged to replicate and modify the work done in this project solely for research or non-commercial applications with the appropriate recognition of the author.

1.5.3. Data Privacy

As it shall be shown in section 2.1, the dataset was crafted from scratch by the author. The dataset has been published both in the platform Zenodo [2] [1] and in an online repository [3] by the author under the Creative Commons Attribution 4.0 International license.

1.6. Socio-Economic Environment

The techniques displayed in this project can be applied to the real world across multiple fields, ranging from the world of chess to our daily life.

Starting with the latter, computer vision algorithms have a big presence in our daily life: from traffic sensors to medical applications, such as the identification of cell tumors [2]. It is therefore of vital importance to explore all the techniques that can improve the field of computer vision independently of the task at hand such as transfer learning, newer model architectures like residual neural networks, or better training paradigms with the use of early stopping.

Continuing with the world of chess, this project can be put to use in many different scenarios. Starting with the intended use, this project aims to help spread chess to as many

[2] https://zenodo.org
[3] https://github.com/PaburoTC/Bachelor-Thesis

people as possible. By recreating a chess position in digital format from a real-life game, it now becomes much easier to analyze the position with a computer, share the game or even continue it with your friends or against a computer. But the uses do not stop there. Being able to correctly identify a chess position is a necessary step for building robots that can play chess physically, and the technologies presented in this project could act as an interface for such robots in the real world. It is to be noted that this project could trivially be adapted to other famous board games such as go or checkers why simply changing the dataset.

1.7. State of the Art

1.7.1. Previous Work

Historically the task of locating and identifying chess pieces has been broken up into two minor tasks: board recognition and piece recognition. Board recognition alludes to the identification of the chess board in the input image together with spatial information, such as its orientation and angle with respect to the focal point of the image. On the other hand, piece recognition refers to the identification and location of chess pieces on the board. As of today board recognition is a frequent prerequisite to piece recognition since knowing the coordinates of the board is of immense help at locating chess pieces.

Board recognition

The most fruitful ways of recognizing a chess board within an image as of today are considered corner detection and line detection. Corner detection, as it name gives away, focuses on identifying the corners of the chess board. With the corners identified, the Hough transform can be applied to identify the straight lines of the board. An example of this can be seen in figure 1.1.

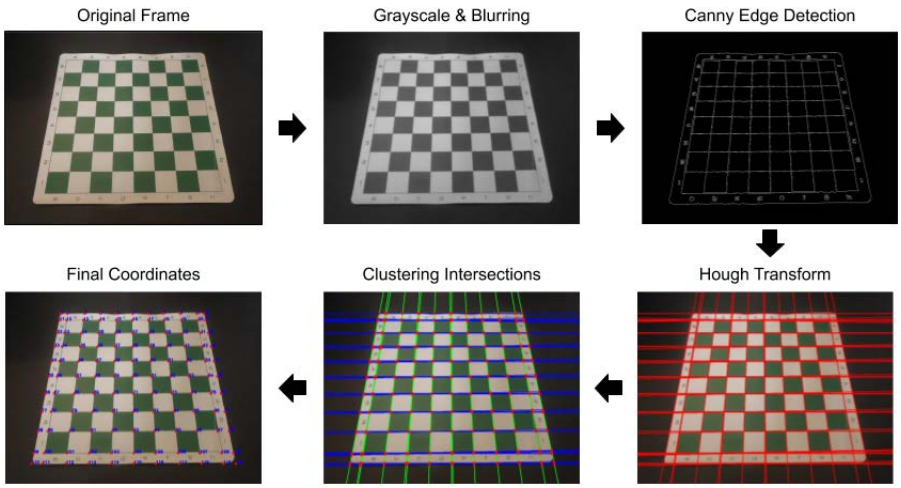

Fig. 1.1. Chess board recognition using corner detection and the Hough Transform. Taken from [3].

It is to be taken into consideration that corner detection can suffer from background noise and most importantly, occlusion of the corners by chess pieces. This can be solved by performing chess board recognition with no pieces on the board from a bird's eyes view camera.

However, using the domain knowledge that there are in total 18 straight lines, half of which are orthogonal to the remaining half, edge detection approaches become much more successful than corner detection. An example of this technique can be observed in figure 1.2.

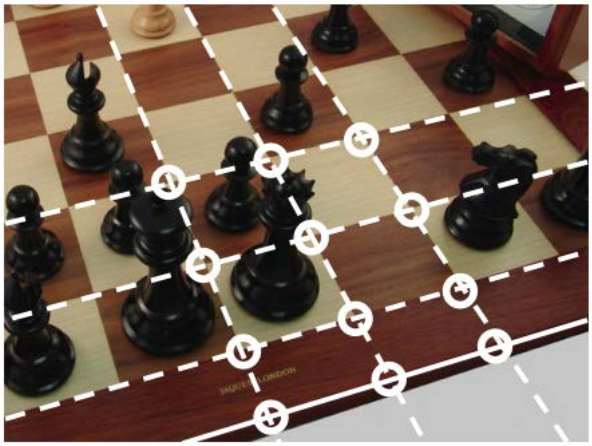

Fig. 1.2. Example of line detection in board recognition. Taken from [4].

Piece recognition

Inside the realm of piece recognition, two different approaches can be identified, which differ on whether or not piece recognition starts at the initial position of a standard chess game.

Game tracking applications have been used extensively for broadcasting chess tournaments online. Knowing the initial position, it can be photographed by a camera positioned over the chess board, obtaining a bird's eye view of the chess board. This initial picture can then be compared to future photographs of the board to determine if there has been a move, purely on the average color difference between the two pictures [5].

On the other hand, approaches that do not rely on having a picture per move until the current state of the board uses color segmentation. Color segmentation relies heavily upon being able to identify the four colors of a chess board: black pieces, white pieces, black squares, and white squares. In normal chess boards, this task is difficult since both black and white pieces share the same color with their respective squares, frustrating this approach. To overcome this difficulty, unreasonable constraints such as using green and red colored chess boards are placed, as seen in figure 1.3.

Fig. 1.3. Example of piece recognition using color segmentation on a red and green colored board. Taken from [6].

A more reasonable approach was proposed by Jialin Ding [7]. With the help of machine learning models learned to identify each piece, he applied a sliding window approach. This means that, with a rectified image of a chess board, he could decompose the chess board into its 64 squares and make a piece prediction for each square of the board. An exampled of a rectified board can be seen in figure 1.4.

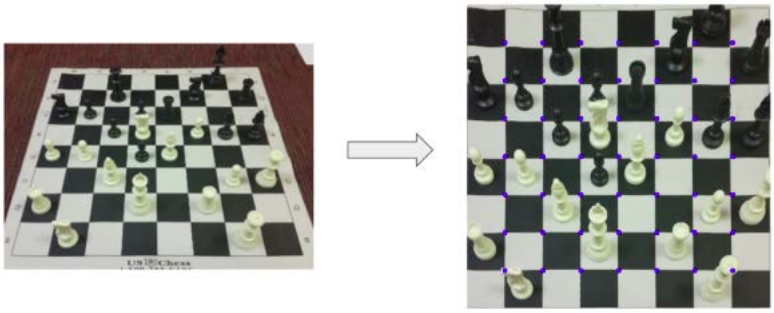

Fig. 1.4. Example of a rectified board from the original image. Taken from [7].

1.7.2. Investigation elements of the project

This section hopes to illustrate how this work differs from the traditional approaches made in section 1.7.1. For starters, this work completely skips the board recognition step. Its

only purpose is to enable piece location, which could be overcome by taking pictures of the board from its center and at a certain height. This way, the picture can be cropped until only the board remains, from where the piece recognition could continue in a similar manner as the sliding window proposed by Jialing Ding [7]. Proving this approach successful will in turn eliminate the restriction of having red and green colored chess boards [6].

The works cited in section 1.7.1 used the same set of chess pieces for training and testing their works. This in turn limits their approaches to only function with the same set of chess pieces used during training, heavily limiting their predictive power. This work will evaluate the result of machine learning classification algorithms trained with several chess sets to gauge their performance as the number of chess sets used during training increases. Moreover, they will be evaluated with an independent chess set not seen during training. Outside FIDE organized tournaments, chess sets vary in shape and form as they have a high degree of artistic freedom in their design. By training with several different chess sets and evaluating against and independent set, the models capacity of abstracting the key features of each chess piece, no matter its art style, will be evaluated. This is the first objective of this work.

Further elaborating on last paragraph, not all chess pieces are necessarily black and white. As happens with their shape and sizes, the color of a chess piece can also be altered due to artistic reasons. Therefore, this work will evaluate if it is necessary for a classification model to make a distinction between colors, meaning it will have a label for each chess piece and color, or if the color can be ignored by the classification model and thus reducing the amount of labels by half. This in turn would require another way to infer the color of each piece. A flexible approach for color identification will be evaluated, to not limit this work to only function on black and white pieces.

1.7.3. Methodologies

For the first two objectives, a convolutional neural network was chosen to solve the multiclass image recognition problem. To make a fair comparison for the first objective, the same model architecture will be used for the two approaches. Said model architecture will be a ResNet18 pre-trained on the ImageNet dataset.

The reasoning behind the selection of methodologies for this project can be better understood by first taking a look at the contextual background of the problem at hand: image recognition.

The paper published by David Hubel and Torsten Wisel in 1959 [8] marked a before and after in the field of computer vision and image classification. In their work, they managed to find basic and advanced neurons in the primary visual cortex of a cat's brain while also showing that the identification of simple structures such as horizontal and vertical lines is the foundation of visual processing. Their work consisted in exposing a car

to visual stimulus while recording electrical signals from the visual area of its brain using an electrode.

Inspired by their work, many researchers in the 80s started to experiment with deep learning algorithms with convolutional layers for computer vision. Before, computer vision algorithms required manual feature engineering, which is costly and time-consuming, especially when it was becoming apparent that convolutional layers could potentially perform feature engineering on their own. One of the first successful convolutional neural networks belongs to Y. Le Cun and was baptized as LeNet-5 in 1989 [9].

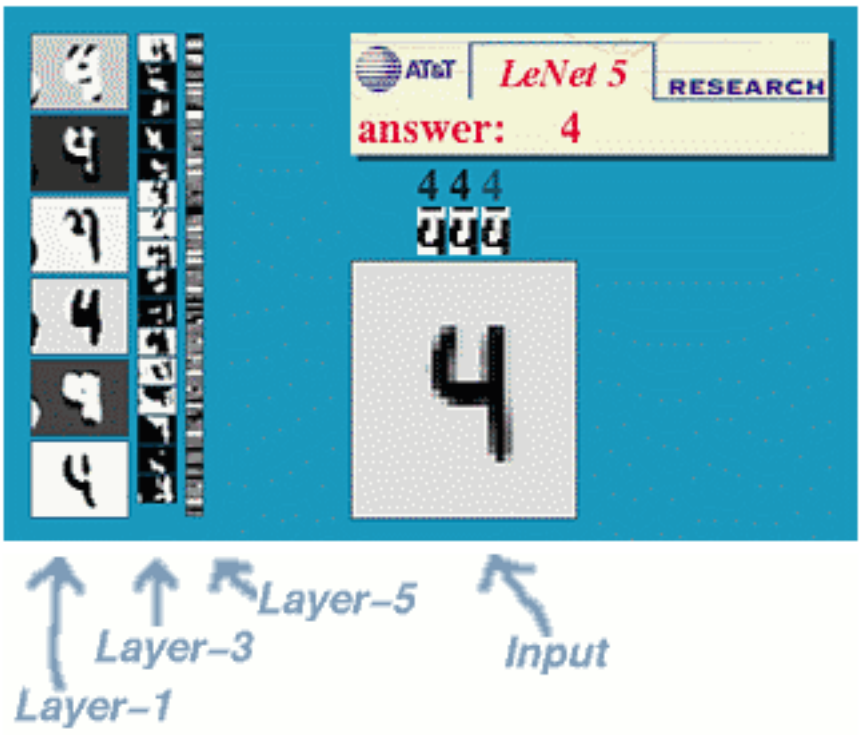

Fig. 1.5. LeNet-5 in action. Taken from [10]

As the field of image classification gained more traction, researchers started to tackle more complex problems like object recognition with state-of-the-art results. With a sudden upswing in the popularity of the field, researchers became in need of a benchmark for comparing and evaluating their computer vision approaches.

As a response to this necessity came the PASCAL VOC Challenge [11] with 20.00 images and 20 target labels, which would soon be overshadowed by Imagenet [12] in 2009 and its annual competition, the ImageNet Large Scale Visual Recognition Challenge or ILSVRC [13], containing over one million images spanning across one thousand different labels.

After a team from the University of Toronto managed to score an error rate of only 16% at ILSVRC 2012, several points below their peers, using convolutional neural networks, the latter became the technique by default for solving image classification problems from then on, as seen in figure 1.6.

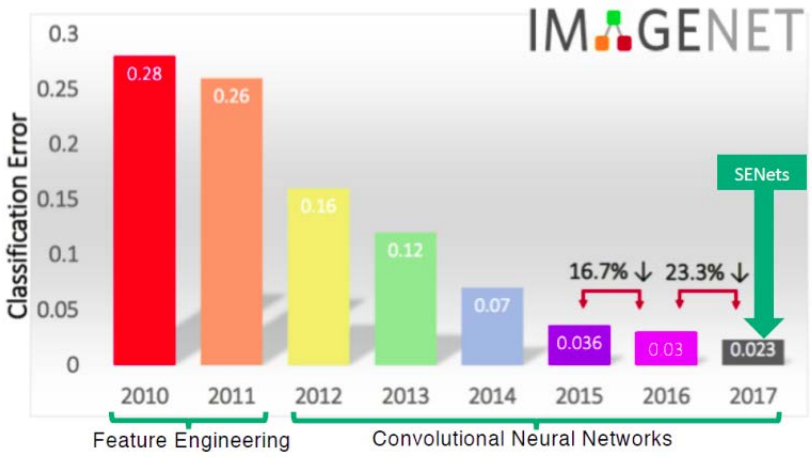

Fig. 1.6. Best error rate per year in LSVRC together with model type. Taken from [14]

The base of a convolutional neural network, in special the first layers, focus on extracting primitive features, such as lines and edges, before moving on to more complex features on deeper layers. Figure 1.7 shows on the left the weights of the filters for the first layer of a convolutional neural network and on the right the images that most strongly matched each filter. The first layer of a convolutional neural network focuses primarily on identifying primitive features, such as straight lines and edges, that will serve as the building blocks for more advanced features.

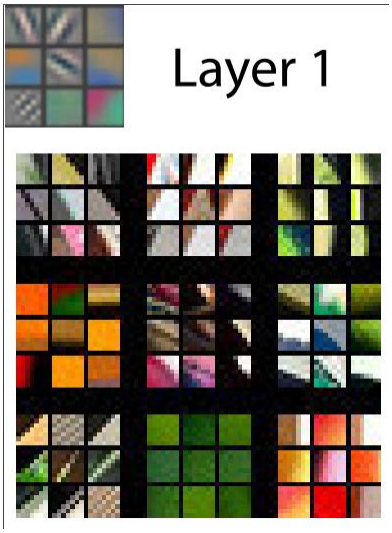

Fig. 1.7. Features captured for layer depth. Taken from [15]

Looking at figure 1.8, it can be observed how the features captured by these deeper layers represent abstract real-world entities like dogs and owls, using information captured by previous convolutional layers.

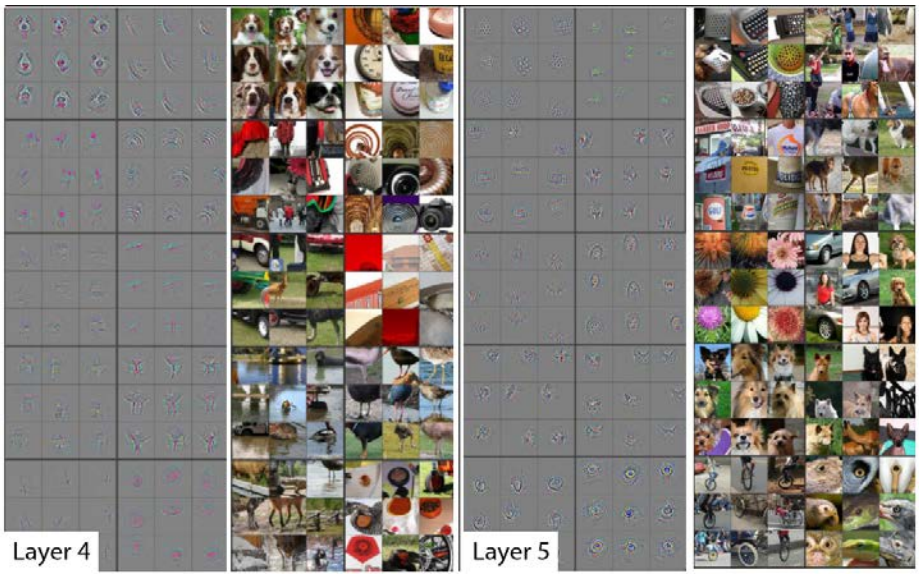

Fig. 1.8. Features captured for layer depth. Taken from [15]

The features learned by the base of a convolutional neural network, in particular by its earliest layers, are domain agnostic in the field of image classification, as learning how to identify straight lines and simple geometric patterns serves to identify more complex patterns. The same cannot be said for the head of such models, as it has been optimized to map from the features learned from the base of the model to the target classes of the given domain. So, while the weights of the head can not be reused, the opposite occurs for the base of the model.

The feature representations that are learned by the base on a given domain can be used with greater success on any other image classification domain than a randomly initialized convolutional neural network. Putting it in simpler terms, a toddler with some knowledge about lines, curves, and edges has a higher chance of identifying objects than a newborn baby with no previous knowledge of the world he lives in. Therefore, it is possible to train a convolutional neural network on a generic domain to then discard its head and reuse its convolutional base for a more specific domain. This process is known as transfer learning.

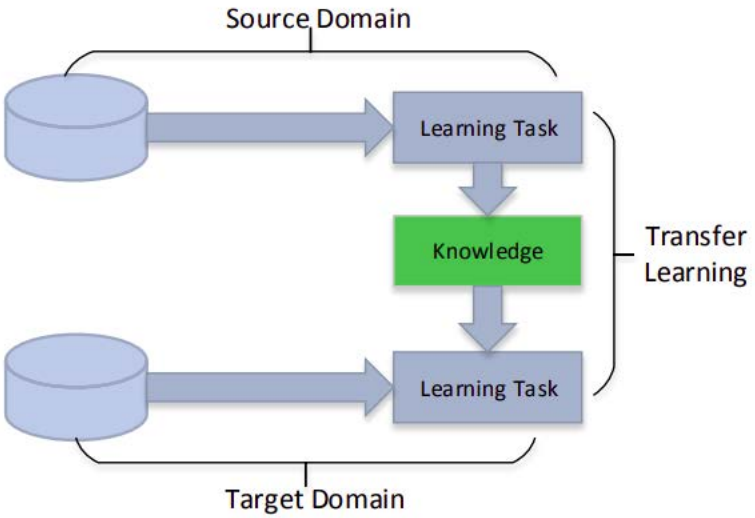

Fig. 1.9. Transfer learning diagram. Taken from [16]

When reusing the base of a convolutional neural network, a randomly initialized head must be attached to it, as the head trained on the generic domain learned representations that can not be extrapolated to a specific domain. Because of the random nature of the head, the model will have a high error during the first epochs of training. Taking into account that the base of the model is barely contributing to this error, it is recommended to freeze its weights during the first epochs of training, until the head's weights converge. After that, fine-tuning can be performed, which consists of continuing training the model with the base unfrozen, to fine-tune the base of the model as the name of this technique suggests. The whole process of transferring the base of a convolution neural network into a new one to make use of its knowledge to further fine-tune it to a more specific domain is known as transfer learning.

In 2009, a group of researchers from Standford University and Princeton University led by Li Fei-Fei and Jia Deng published the revolutionary image database known as ImageNet [12]. This database is yet today the biggest body of human-classified images, composed of over fourteen million images with over 1000 target labels. ImageNet completely revolutionized the computer vision field, as for the first time researchers had access to a vast human annotated dataset. This dataset has been used as a benchmark for models, in the already mentioned ILSVRC, but more importantly, it was used as a generic dataset to train convolutional neural networks and apply transfer learning.

As seen in figure 1.10, performing transfer learning on models pre-trained on ImageNet yield better results than models trained from scratch or on other less generic datasets such as iNat2021 [17] on a wide array of image classification tasks.

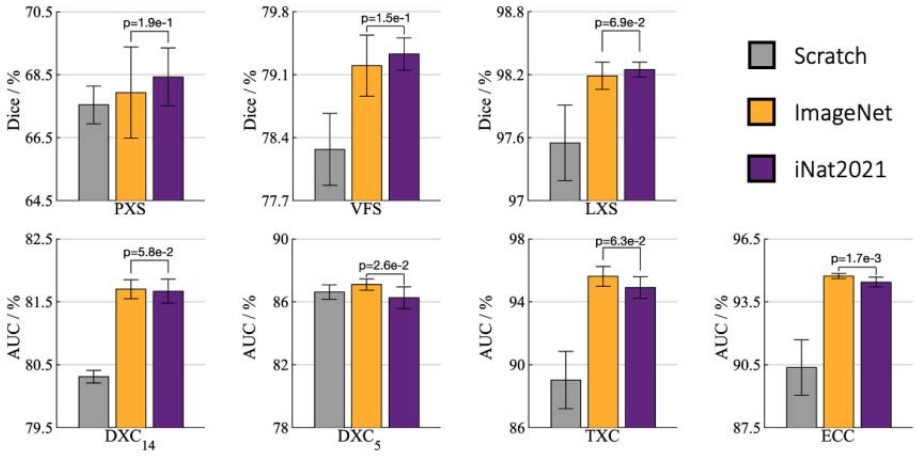

Fig. 1.10. Transfer learning benchmark. Taken from [18]

Transfer learning also enables learning representations for specific domains with small datasets, as all the important representations were learned on the generic domain and all that is left is to perform fine-tuning. This has enabled this project to achieve its current results without having to invest huge amounts of time into forming a large dataset.

Another technique that also yields great results, especially when working with small datasets is data augmentation. Data augmentation refers to the process of creating random variations to the images in our datasets such that they change their appearances but not their underlying meaning. Such variations can be produced by geometric operations such as rotation and translation as well as changes in the lighting as seen in figure 1.11.

Fig. 1.11. Data augmentation example. Taken from [15]

Data augmentation not only helps in enlarging the dataset, it also improves the representation of the underlying data distribution by smoothing possible bias when creating the dataset. For example, if an image dataset of cars was only created by taking pictures of cars from the same given angle, a model trained on said dataset would probably fail to recognize a picture of a car taken from a different angle. Data augmentation reduces the risk of this occurring by making the dataset more expressive.

Another interesting technique to mention for the case of hyper-parameter tuning is early stopping. Early stopping monitors a training metric, usually validation loss or validation error and halts training when no improvement has been made to the chosen metric after a given number of epochs, known as patience and returning the weights to their best epoch. Early stopping acts as a regularizer, as it refrains the model from overfitting the training data.

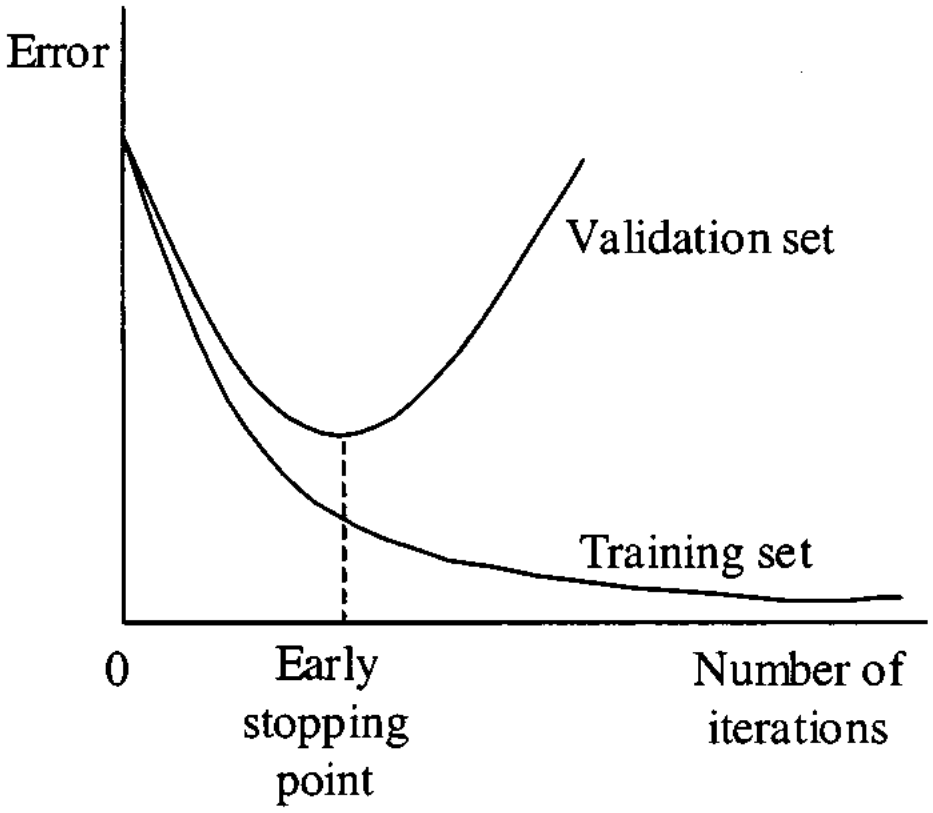

Fig. 1.12. Early stopping example. Taken from [19]

For all the reasons mentioned above, for the piece classification task a convolutional neural network, pre-trained on ImageNet using data augmentation and early stopping was used. As for the color classification of the pieces in the 7 labels scenario, an distance-based approach will be used, as shall be explained in further detail in section 2.3.

2. DEVELOPMENT

2.1. Data

As the classification task is approached from a machine learning viewpoint, data is required to both train and evaluate the resulting models. Since the result is to reproduce chess positions from a bird's eyes perspective, the input to the classification model will be a bird's eye view of a single chess piece.

Chess pieces lack standardization. Each chess set has an almost unique style represented in its pieces. Humans can recognize the essential features of any individual chess piece, but this high degree of artistic liberty at the time of crafting and designing chess pieces is most certainly a problem for machine learning models. In the best case scenario, the machine learning models learn to properly abstract the key features of any individual chess piece while ignoring their artistic ornaments. On the other hand, in the worst case scenario either the models do not learn any relevant features or they focus on irrelevant, artistic details.

This inevitably calls for a dataset composed of chess pieces proceeding from different chess sets, in hopes that the models can generalize the key features present in any chess piece despite their artistic takes on them. Later I will show how increasing the diversity of chess sets affects the generalization ability of the models and thus their performance.

The dataset will be composed of 13 labels: 12 for the chess pieces discriminating by color plus an extra label representing the empty squares. This last label is crucial for recreating chess board positions where empty squares are abundant.

In order to gauge the models generalization power, a validation set is also necessary, with it being formed by chess pieces proceeding from a different chess set than the ones used for generating the training data. This validation set will be composed of 64 images per chess piece, discriminating by color, plus 64 images of empty board squares.

The training dataset will be composed of 64 images per chess piece, discriminating by color, per chess set. Each piece will be photographed from all the squares of the board with a static camera to capture all its angles. Said camera has been arranged with the help of a tripod to have its focal point perpendicular to the board and on its center, in a bird's eyes perspective.

As the number of different chess sets used in training increases, and so does the training data available, the effects of increasing the dataset in size and diversity can be observed. In total, up to 3 chess sets were used during training, which makes a total of 4 chess sets used during the construction of the classifier. This number was bounded by time and resthece restrictions, although the results obtained yield enough information to gauge the importance of having a diverse and representative dataset.

For illustration purposes, here are samples from the chess sets used:

- **Training chess set 1**

Fig. 2.1. Training dataset sample from chess set 1.

- **Training chess set 2**

Fig. 2.2. Training dataset sample from chess set 2.

- **Training chess set 3**

Fig. 2.3. Training dataset sample from chess set 3.

- **Validation chess set**

Fig. 2.4. Validation dataset chess set sample.

Table 2.1 illustrates more clearly the amount of training and validation data available. As mentioned before, each piece per chess set will be photographed from each of the 64 squares that make up a chess board to capture all its angles and features. As for the empty squares, it is only necessary to photograph them once, since all the pieces regarding from which set they come will be photographed on the same board. Therefore, it would be redundant to further add more empty squares to the dataset.

Label	Training board 1	Training board 2	Training board 3	Validation board
black_bishop	64	64	64	64
black_king	64	64	64	64
black_knight	64	64	64	64
black_pawn	64	64	64	64
black_queen	64	64	64	64
black_rook	64	64	64	64
empty	64	0	0	64
white_bishop	64	64	64	64
white_king	64	64	64	64
white_knight	64	64	64	64
white_pawn	64	64	64	64
white_queen	64	64	64	64
white_rook	64	64	64	64

TABLE 2.1. NUMBER OF IMAGES PER LABEL

2.2. Chess piece classification

2.2.1. Model Selection

In this section, the three different models that were trained are presented. The main goal of these models is to test if the models must classify each square into one of the twelve different chess pieces that exist or as empty, and thus having 13 labels, or if on the other hand, it can ignore the color of the piece and simple predict its class, thus having only 7 labels. The latter approach has twice the amount of training data per class for the chess pieces, while not increasing the amount of data for the empty class.

Using the 7 labels approach comes at a cost: finding another way to infer the color of a piece. The color could be predicted by calculating the distance between a piece‘s mean pixel value to the average mean pixel value of both black and white pieces. However, this approach would prove unable to predict the color of chess boards using other colors different from white and black, which is not common but also not unlikely. In section 2.3 I shall present a solution to this problem, not to different from this initial distance-based approach, but much more flexible.

After comparing the 13 label and the 7 label approach, a One vs All model will the constructed using the more solid approach of the two. A One vs All model creates as many classifiers as labels, with the *i* classifier being trained to discriminate between the *i* label and the rest of the training dataset.

13 Labels

By using 12 labels for chess pieces plus and extra one for the empty squares, this model discriminates by color: a black pawn belongs to a different category than a white pawn. It is to be noted that a model that discriminates by color is vulnerable to color changes, as it will fail to correctly classify chess pieces with different colors other than black and white.

Fig. 2.5. 13 Label training sample.

7 Labels

By using only 6 labels plus and extra one for the empty squares, this model does not discriminate by color, meaning that a black pawn and a white pawn both belong to the same category: pawn. In section 2.3 it shall be explained how the color of each piece will be inferred.

Fig. 2.6. 7 Label training sample.

By using a model with 7 labels, it shall be evaluated if the model benefits from a reduction in the output labels. Doing so doubles the amount of training data per class for the chess pieces and almost halves the number of output classes.

2.2.2. Model Architecture

As mentioned in section 1.7.3, a convolutional neural network is to be preferred versus a standard fully connected network for image classification. Although fully connected networks can get the job done, they do so at a high computational cost, primarily due to the size of the input image. As it shall be seen now, convolutional neural networks change this by compressing the image in hopes of leaving behind only the most meaningful information behind. This process takes place in what is called the base of the network and automatizes image processing and feature selection. The resulting features are then feed to a fully connected network, also called the head of the network, to perform the final classification. As it can be seen, the benefit of convolutional neural networks reside on their base, which reduces the shape of the input until it can be managed by a fully connected network. Said base is typically composed of convolutional layers followed by pooling layers.

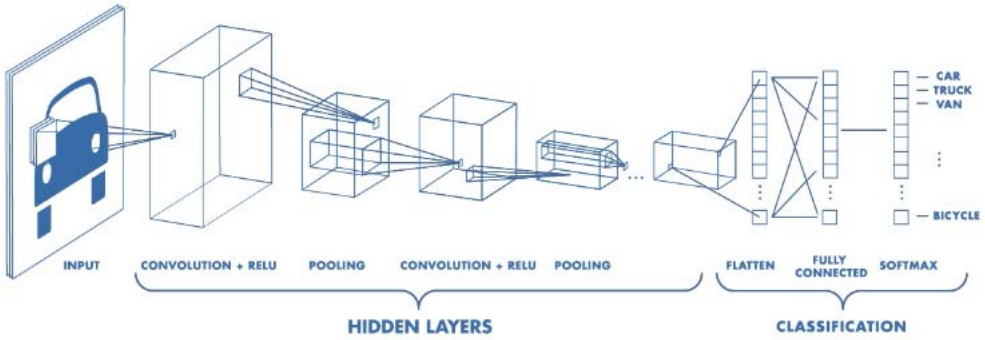

Fig. 2.7. Structure of a convolutional neural network. Taken from [20].

Figure 2.7 illustrates how the base of a convolutional neural network reduces the size of the input until it is fed into the head of the layer for the final layer prediction.

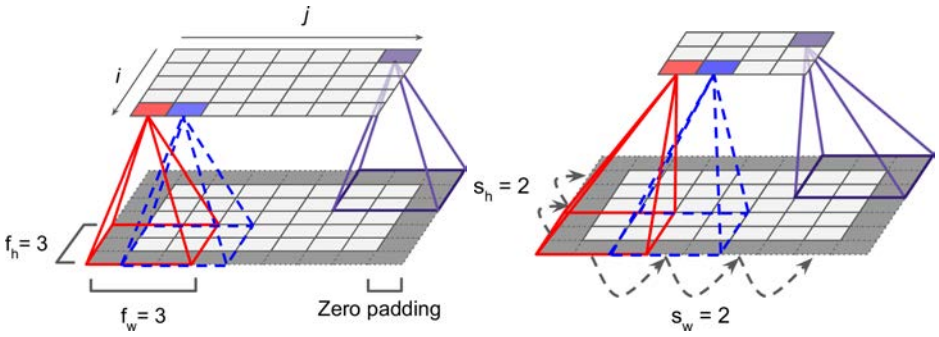

Fig. 2.8. Convolution with zero padding and stride two for a 3x3 filter. Taken from [21].

Convolutions can be defined as 2D matrices which are then applied to the image as a filter. For each pixel of the image, the resulting pixel of the output is going to equate to the element-wise multiplication of the filter and the surrounding pixels of the given one. For the pixels near the borders of an image that do not have surrounding pixels in any of their directions, the picture is padded typically with either zeros or ones. The size of the output is subject to the stride of the filter. The stride of the filter is the number of pixels the filter moves Refer to figure 2.8 for a visual intuition of how convolutions works, or to the following formula if you prefer the mathematical definition, using zero-based indexing and a filter of dimension 3x3.

$$output_{i,j} = \sum_{-1<x<1}^{x} \sum_{-1<y<1}^{y} input_{i+x,j+y} * filter_{x+1,y+1} \tag{2.1}$$

Therefore each pixel of the resulting output encodes spatial information about the surrounding pixels of the input, condensing the image into a smaller size. Convolution layers are almost always followed by a pooling layer. Pooling layers also reduce the size of the input but without the need for a filter, thus not having any trainable parameters. Figure 2.9 shows the two most typical types of pooling layers.

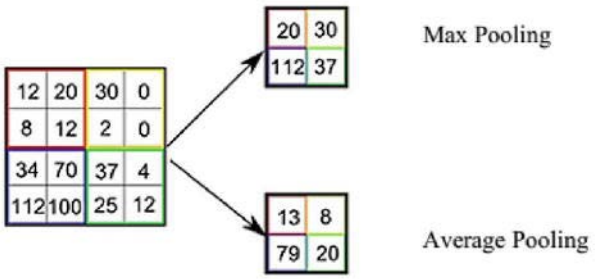

Fig. 2.9. Pooling layer, taken from [22].

The base of a convolutional neural network is formed by several concatenations of convolutional layers followed by pooling layers, after which the final output is flattened before being fed into the head of the model. The head, formed by a fully connected network, then maps the features obtained from the base of the model to the target labels.

As convolutional neural networks get deeper, the more abstract the features they can identify. Nevertheless, this comes at a price: loss of information from previous layers. As convolutions compress the input of the previous layer, inevitable information is lost to pay for a higher level of abstraction. In 2015, researchers from Microsoft led by Kaiming He found a way to prevent information loss while maintaining rich features by introducing deep residual learning for image recognition [23]

A deep residual model or resnet for short, as can be seen in figure 2.10, functions as a normal convolutional neural network layer but at the end of the layer, it recovers information from the previous layer via what is known as shortcut connections. Several models belong to the resnet family, each one with a given number of residual blocks.

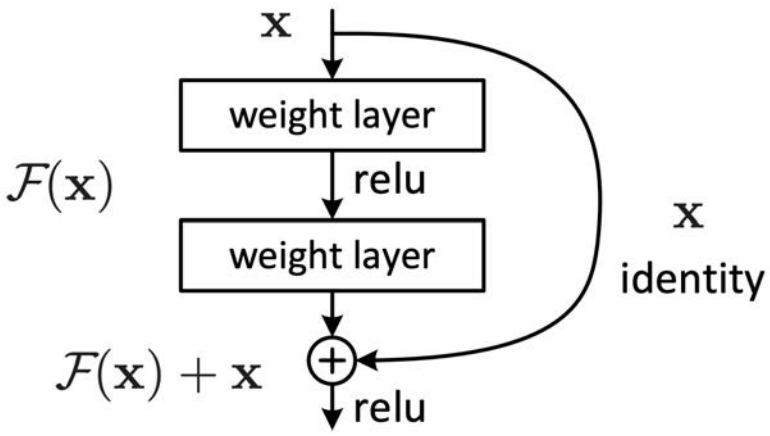

Fig. 2.10. Building block of resnets, residual block. Taken from [23].

To evaluate their proposed resnet models, Kaiming He et al benchmarked two resnet models, resnet18 & resnet34 on the ImageNet 2012 classification dataset from ILSVRC [13] against standard convolutional models of the same depth, referred to as plain models. The models were trained on 1.28 million training images and evaluated on the 50.000

validation images, providing the final top-1 error rates on the 100.000 test images. The results can be seen both on figure 2.11 and table 2.2

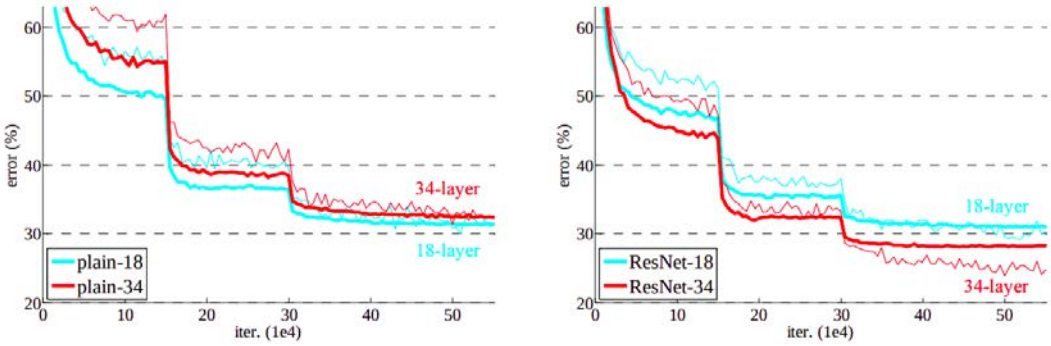

Fig. 2.11. Benchmark of resnet models. Taken from [23].

Depth	Plain	ResNet
18 layers	27.94	27.88
34 layers	28.54	**25.03**

Sthece: [23]

TABLE 2.2. TOP-1 ERROR RATE FOR MODELS ON ILSVRC 2012.

As it can be observed, both ResNet18 & ResNet34 outperform their plain counterparts, especially the latter. This serves as proof of the effectiveness of the ResNet model family.

To make set the stage for a fair comparison of the 13 labels vs the 7 labels approach, both classifiers will be trained on a ResNet18 pre-trained on the ImageNet dataset. The simplest model of the ResNet family has been chosen to use a small hypothesis space for the classifiers. This will help us gauge if the model is too powerful or if there is room for improvement with more expressive model architectures.

By using a pre-trained model, general knowledge about images, such as identifying straight lines or curves, is being transferred into the models. This way, less data is required and shorter training times are needed for the model to learn all the key features.

2.2.3. Model Training

The training of the classifiers was conducted on a pre-trained network under a newly instantiated final layer with random weights. Due to the randomness of this layer, high loss values and thus weight updates can be expected. Nevertheless, these first weight updates caused due to the random nature of the final prediction layer, are to be avoided since they would alter the weights of the already trained base of the model.

Therefore, to prevent the base of the model to update during the first few epochs, its weights are frozen and only the ones of the final layer are updated. This occurs during 10 epochs, after which the model at the epoch with the lowest validation loss will be saved.

Then resulting model is then fined tuned. This means training the whole model, including its previously frozen base, for 20 epochs. Once again, the model at the epoch with the lowest validation loss will be saved to prevent overfitting. If more than 10 epochs pass since the best validation loss is obtained, training is ceased. This technique is called early stopping and helps prevent overfitting of the training data.

2.3. Color Identification

Since different chess sets use different colors to differentiate one player from another, to classify the color of a piece, its mean pixel value cannot be directly compared with the mean pixel value of both the black and white pieces of the training dataset, since this approach would only work for black and white chess sets.

Given a bird's eyes view of a whole chess board, a better approach would be to run a k-means algorithm with 2 clusters on the mean pixel value of all the squares where the piece classifier is confident there is indeed a piece and not just an empty square. Let's first explain the k-means algorithm, developed by Stuart Lloyd in 1957 for Bell Labs and published outside the company in 1957 in the following paper [24].

K-means is a clustering algorithm which given an unlabeled set of points, it clusters them into k clusters. K-means is an unsupervised algorithm since it does not require any labeling of the points, but instead produces them. K-means instantiates k centroids randomly in the point space, and assigns each point to the nearest centroid, forming an initial cluster. Then, each centroid is updated to the center of mass of its cluster and the process is repeated until there is a minimum change in the centroids position, indicating that the algorithm has converge and that the clusters have been found. An example run of k-means can be seen in figure 2.12.

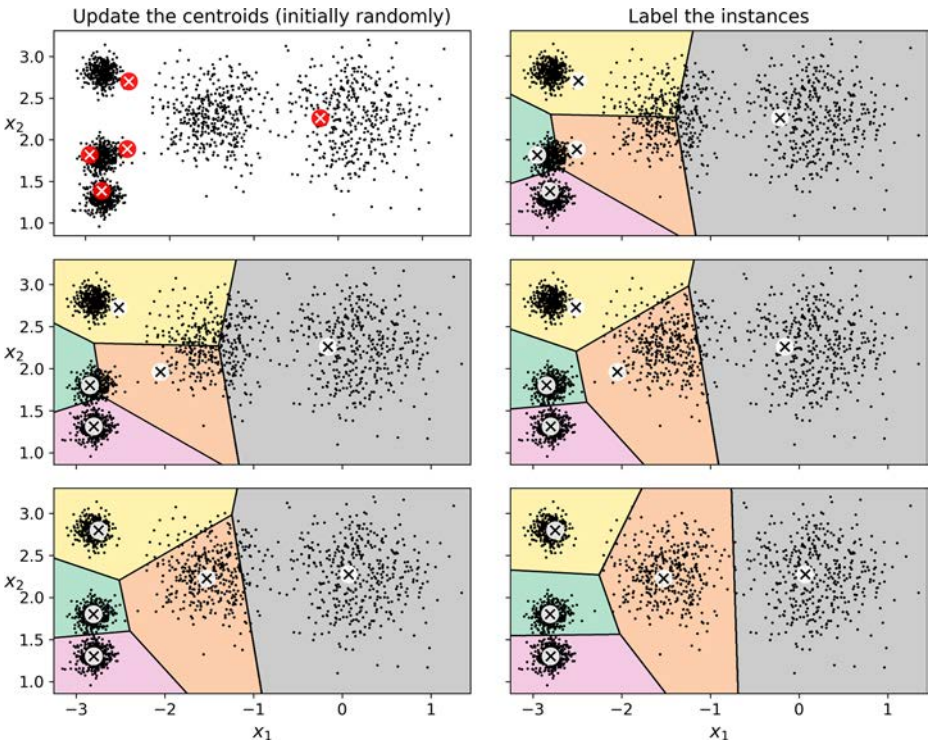

Fig. 2.12. K-means example run. Taken from [21].

K-means algorithm is guaranteed to converge, as the distance from each point to its assigned centroid will only decrease in between iterations. However, k-means is highly sensitive to centroid initialization, as the same set of points may result in vastly different clusters with two different centroid initialization, as can be seen in figure 2.13.

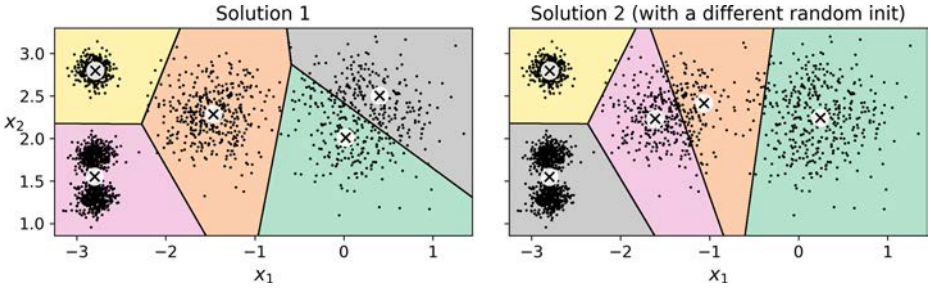

Fig. 2.13. K-means results for different centroid initialization. Taken from [21].

Knowing that the colors chosen to decorate the chess pieces need to be easily differentiable by the human eye, the clusters can be initialized at the extremes of the RGB color spectrum: 0 and 255, thus removing the random nature of k-means. The objective then is for the k-means algorithm to find suitable centroids for black mean pixel values and white mean pixel values respectively. The distance to those centroids of the mean pixel value of an image can then be used to identify its color.

However, it is important to note that these chess pieces are standing in squares of the same colors used to differentiate them. It is then necessary to somehow remove the

background for every square, or else the color of the square will influence the mean pixel value of the piece and thus possibly leading into and incorrect color classification.

To remove the background, a mask of the same size as the images was applied, in this case, 181x181 pixels. To create the mask, a function that preserves the values of the pixels when they are close to the center of an image, while dropping them to zero when moving towards pixels on the edge of a picture, is of great use. Function 2.2 does the exact opposite:

$$f(x) = \frac{x^a}{x^a + (1-x)^a} \tag{2.2}$$

With $a = 10$, the graph of f can be seen in figure 2.14:

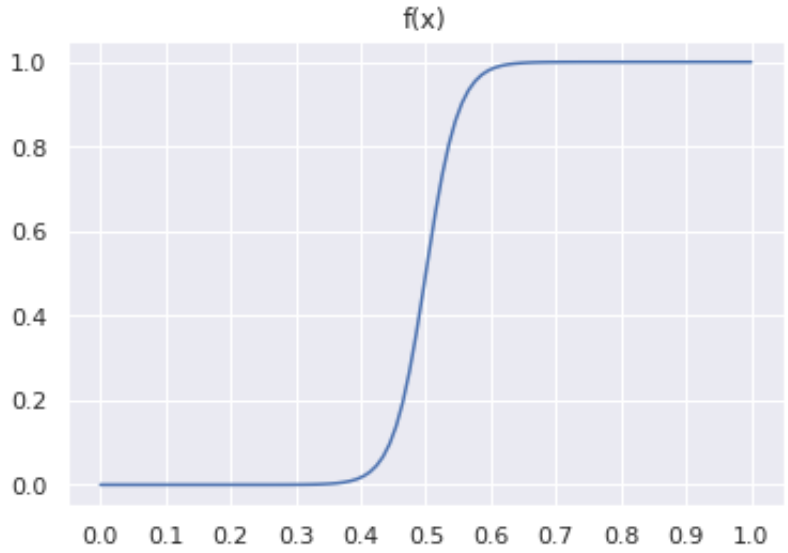

Fig. 2.14. Graph of the function used for masking color.

As it can be seen from this graph, f almost resolves to 0 for $x \in [0, 0.4]$ and from there it increases until reaching one for $x \in [0.6, 1]$. Let M be defined as a matrix of $width = 181$ and $height = 181$, thought this shape will vary depending on future input. The center values of M shall be close to 1, while the ones near the edge to be close to 0. For any given entry of M i, j, i, j need to be mapped to the range $-1, 1$, thus the center indices will have an absolute value closer to 0 with this mapping, while the outer indices will have absolute values close to 1. Even though we are dealing with two indices, i, j, we only need to take into account the one with the greatest absolute value after the mapping, because it is the index that indicates how far from the center that position is. If we apply f to the index with the highest absolute value after the mapping, we will be left with center positions having values close to 0 while outer positions would have values closer to 1. As we desire the opposite, we will subtract 1 the result of f, thus leaving us with the desired matrix. Mathematically, $M_{i,j}$ equals equation 2.3.

$$M_{i,j} = 1 - f(max(abs(\frac{i}{width} * 2 - 1), abs(\frac{j}{height} * 2 - 1))) \tag{2.3}$$

Which can be visualized in figure 2.15.

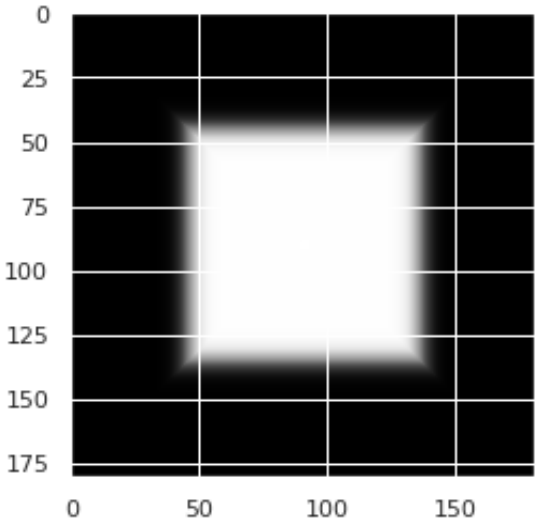

Fig. 2.15. Mask used for color identification.

Applying the mask to a given chess piece, or rather perform and element-wise multiplication between the mask and the target image produces the results displayed in figure 2.16.

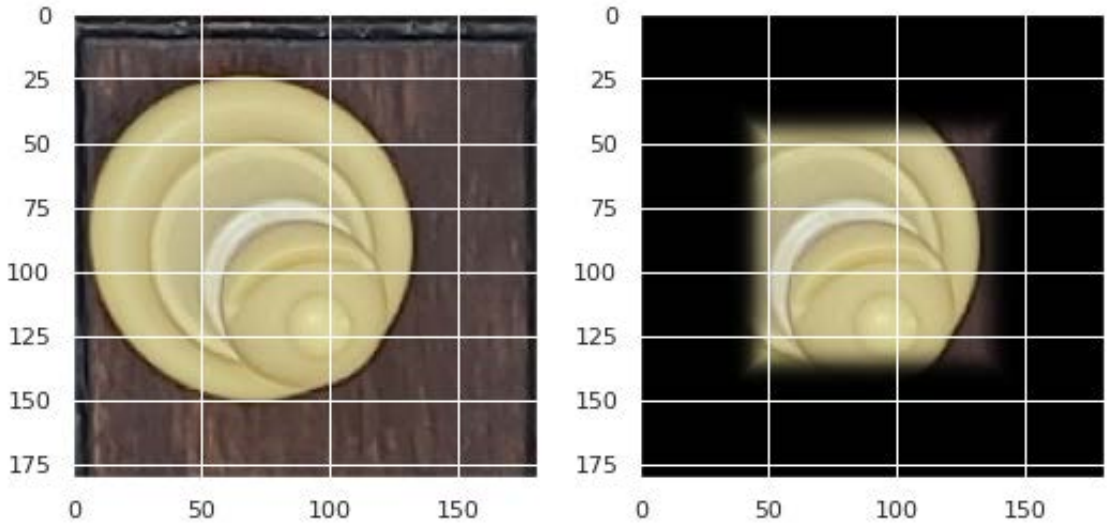

Fig. 2.16. Chess piece before and after being applied the color identification mask.

Which successfully removes the background of an image, only leaving behind the chess piece.

2.4. Chess board recreation

To recreate a chess board position in digital form the piece classifier from section 2.2 shall be used. Said classifier takes as input a bird's eye view of each piece and outputs the chess piece. Depending on the outcome of the experiments presented in section 3.1, the 13 or the 7 labels approach shall be chosen.

If the 13 labels approach is to be chosen, the chess board recreation becomes much simpler. After taking a bird's eyes view of a chess board position, the image must be broken down into its 64 squares. Each one of this squares shall then be feed to the 13 label model, which will output its prediction together with its confidence of said prediction. After casting a prediction for each of the 64 squares, all the predictions can be joined together to form the digital equivalent of the chess board. The process described here can be observed in figure 2.17.

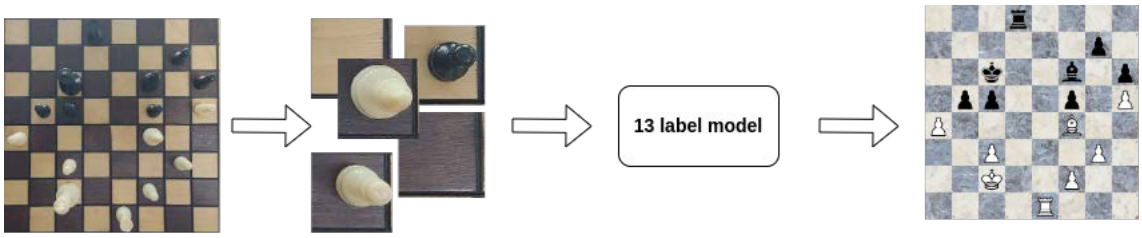

Fig. 2.17. Chess board recreation pipeline with the 13 label piece classifier.

On the other hand, if the 7 labels approach is to be favored, recreating a digital version of the chess board is suddenly not so trivial. The first part of the process is identical to the 13 labels approach: the board is broken down into 64 squares and each one of them is labeled either as empty or as a piece, depending on the confidence of the 7 labels model prediction. But the color of the piece is yet to be decided. All the squares that are labeled as containing a piece will be fed into the color system presented in section 2.3, where two clusters of color shall be found. Having each piece assigned to one of the two color clusters, the darker one shall be labeled as the black pieces and the remaining cluster as the white pieces. Given the complexity of this system, even if the 7 labels model performs better than the 13 label model individually, it is to be tested if 7 label model together with the color identification model still performs better.

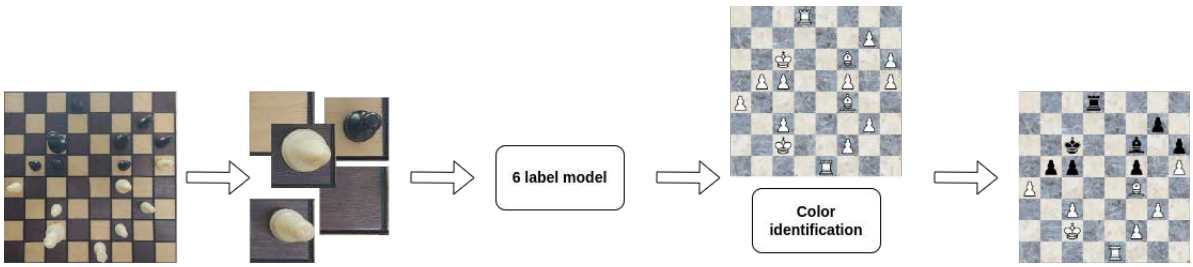

Fig. 2.18. Chess board recreation pipeline with the 7 label piece classifier and color identification.

Given that the chess board is recreated locally by predicting which piece goes into each square, it could be more than possible to obtain a prediction with more than one king per color. In the case of the 13 labels model, the model will remember the model's confidence in its prediction and the best alternative for all predictions of both black and white king. After all 64 predictions, we will keep the kings with the highest confidence for both black and white, and replace all other kings with their alternatives.

For the 7 labels model, it is not known if a piece will be black or white until after the color selection algorithm. For this reason, no distinction can be made for kings, as it s unknown if it will be a white or black king. Therefore all the same data as the 13 label model shall be stored about the kings, but until after all the pieces have an assigned color a king can not be chosen for both colors.

3. EVALUATION

3.1. Piece Classification

This section will compare the results obtained by both the 13 and 7 label models. The best one shall then be used as the building block for the One vs All model, which results will be presented in section 3.1.2.

3.1.1. 13 labels vs 7 labels

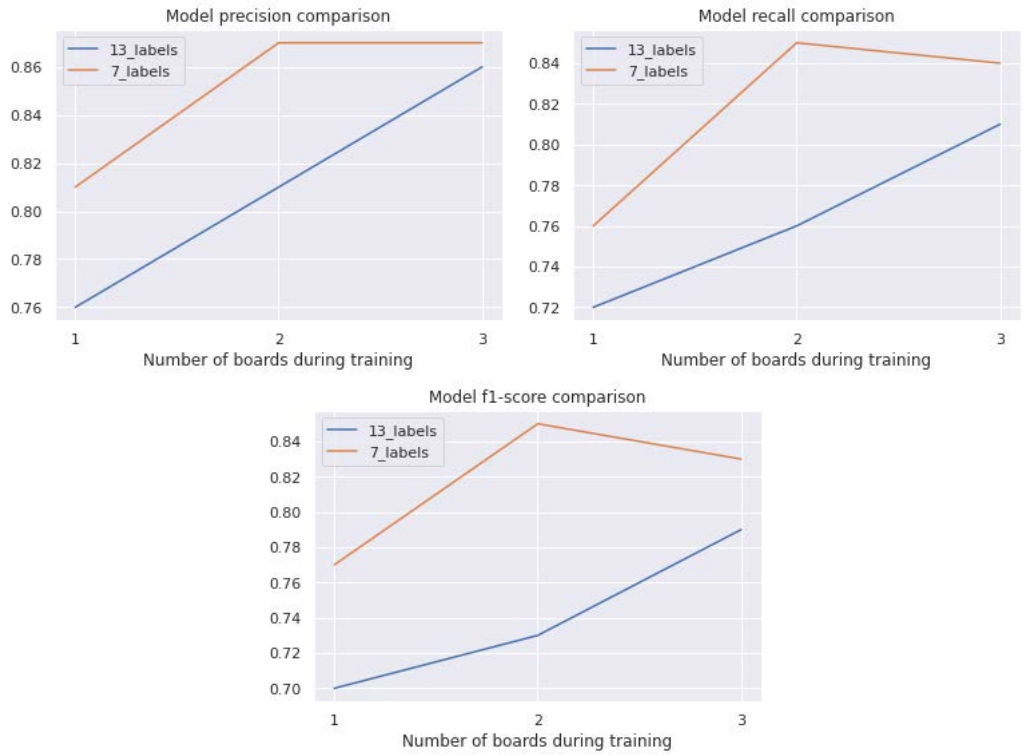

Fig. 3.1. 13 vs 7 label model metrics

The first thing that the reader should notice from figure 3.1 is the difference in both models when a second chess set is added into the training dataset. With the models being trained with chess pieces coming from a single chess set, the models could not extrapolate the features that define each chess piece as the training data was not representative enough of the underlying distribution. By adding a second chess set, the models could focus on the features that where present in pieces of both sets, therefore gaining generalization power.

The addition of the third chess set to the training dataset, while also improving the overall performance of the models, it does so ever so slight less than the second set. With the third chess set being the one with a higher degree of artistic freedom, its chess pieces

look significantly different from the other two chess sets. This may result in the model having trouble generalizing for this dataset. This phenomena can be explained with a simpler reasoning: the models learned to generalize enough with the second chess set, reducing the possible impact of adding more diverse data to the training dataset.

Overall, it seems that the 7 labels model performs better, especially when training data is scarce.

3.1.2. One vs All

Given the superiority of the 7 labels model let's now evaluate the performance of the One vs All model constructed following the 7 label approach.

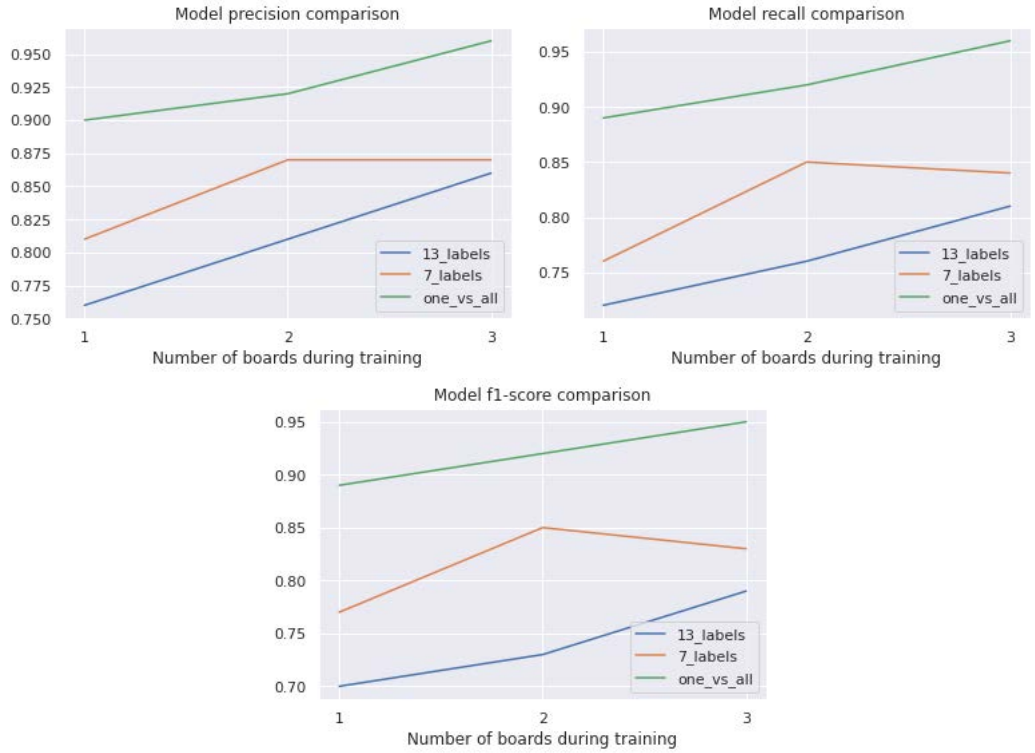

Fig. 3.2. One vs All model metrics

As we can see from figure 3.2, the One vs All ensemble model outperforms both the 13 and 7 label models. Nevertheless, the same cannot be said for each of the 7 models that form the One vs All ensemble model. Let us evaluate the difference between the results of each one of the 7 models vs its greater One vs All model.

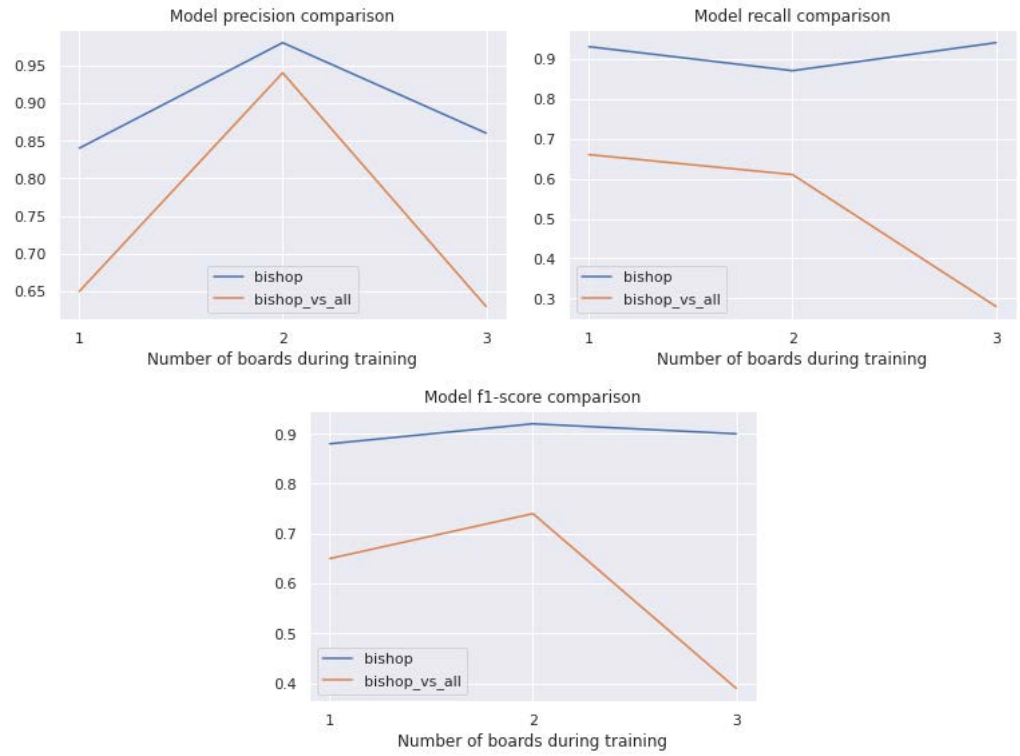

Fig. 3.3. Bishop vs Bishop vs All metrics

As seen in figure 3.3, the Bishop vs All model serves as the perfect argument in favor of ensemble models like the One vs All, where the resulting model is far greater than the sum of its parts. While the Bishop vs All model does a poor job identifying the bishop class on its own, it is most certainly confident in its predictions. This results in much higher metrics for the class of the ensemble model vs the results obtained by the Bishop vs All model.

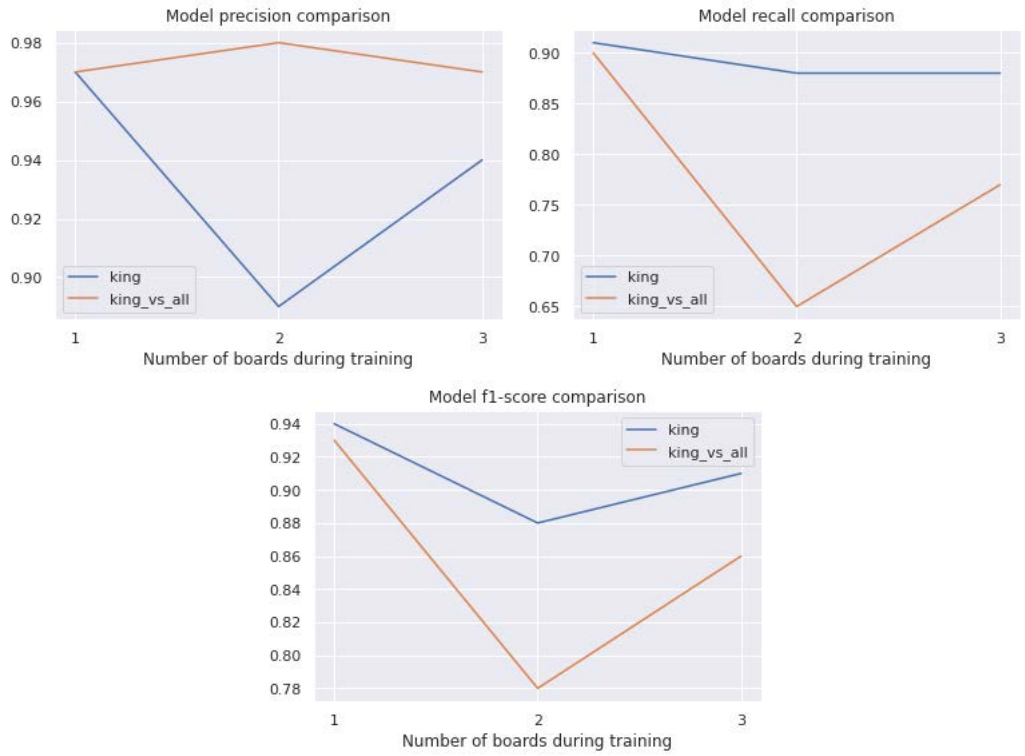

Fig. 3.4. King vs King vs All

From figure 3.4 we can again see the same pattern as with the bishop class, the sum of week classifiers results on strong models. It is to be noted that while the One vs All model outperforms the individual King vs All in both recall and f1-score, it performs slightly worse on precision.

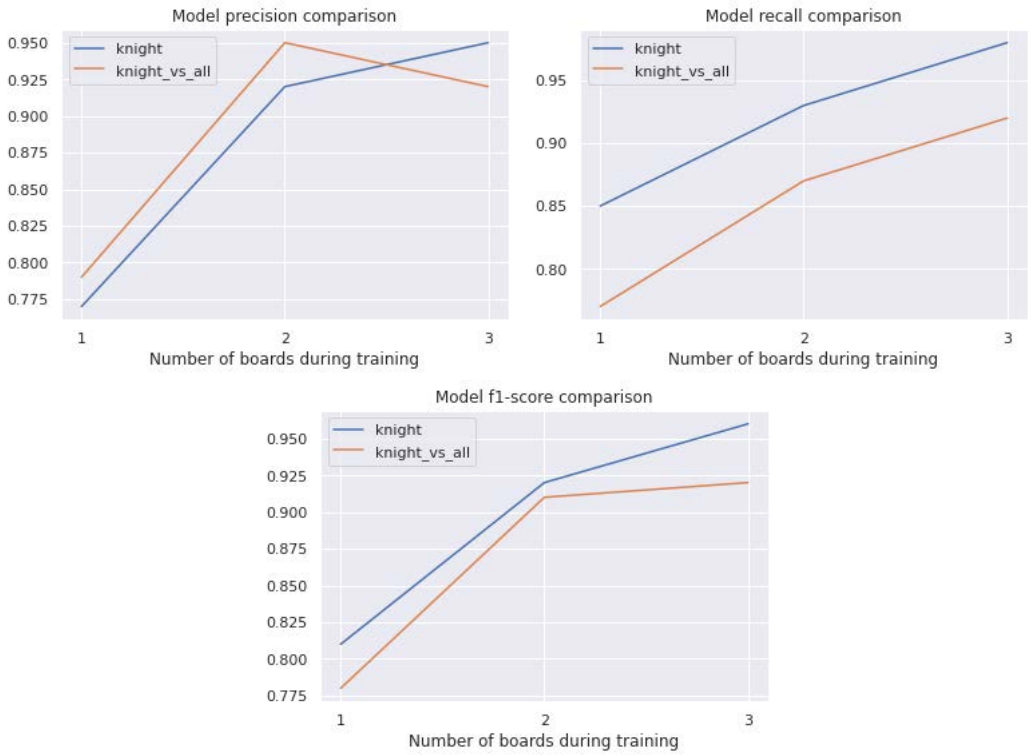

Fig. 3.5. Knight vs Knight vs All

The knight for example serves as the exception to this rule, as seen in figure 3.5. The Knight vs All classifier was most certainly not weak at all. This could be due to the distinctive nature of the piece, which is the only piece in the game that does not follow a cylindrical pattern, making it stand out in turn. Nevertheless, it still performs slightly worse than the One vs All ensemble model.

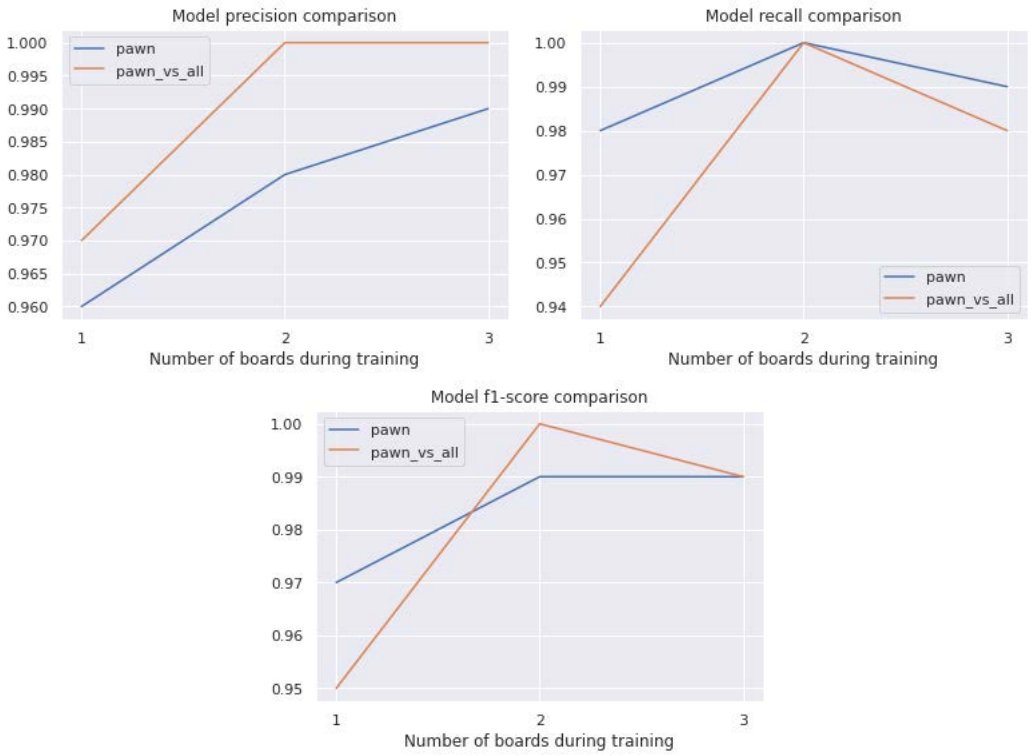

Fig. 3.6. Pawn vs Pawn vs All

From figure 3.6 we can observe that both the Pawn vs All and the One vs All model had a great performance for the class pawn. While the Pawn vs All seems to outperform the One vs All model, it does so slightly with no significant difference.

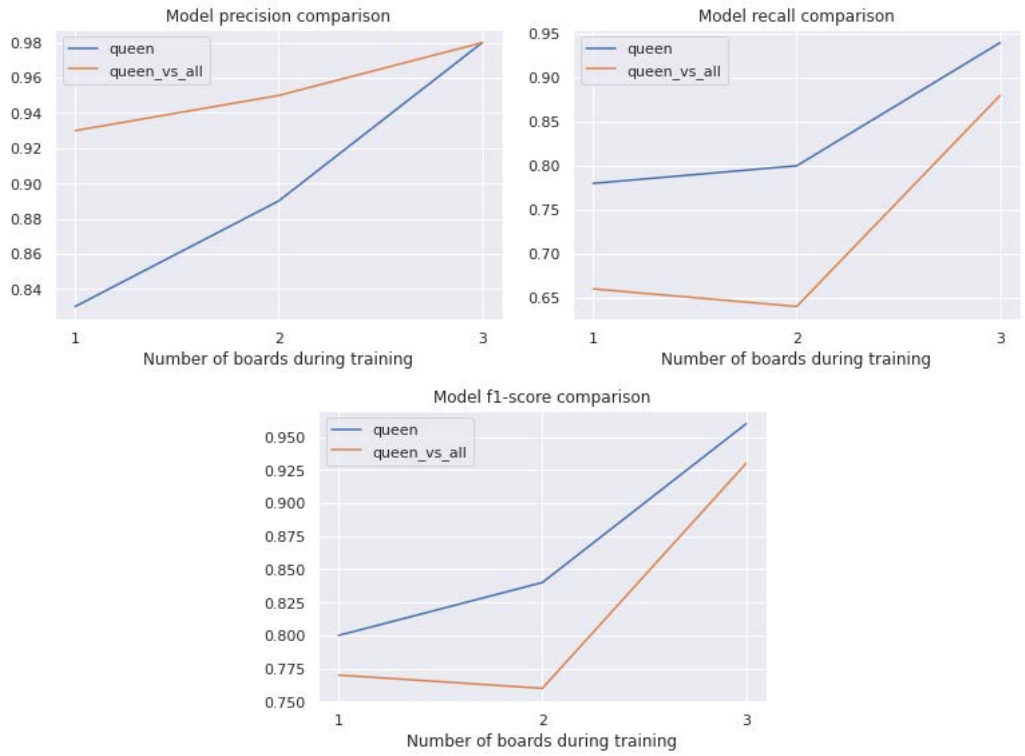

Fig. 3.7. Queen vs Queen vs All

In similar fashion as the Pawn vs All model, figure 3.7 shows how both the Queen vs All model and the One vs All model had very similar results, with no real significant difference when trained with three different chess sets.

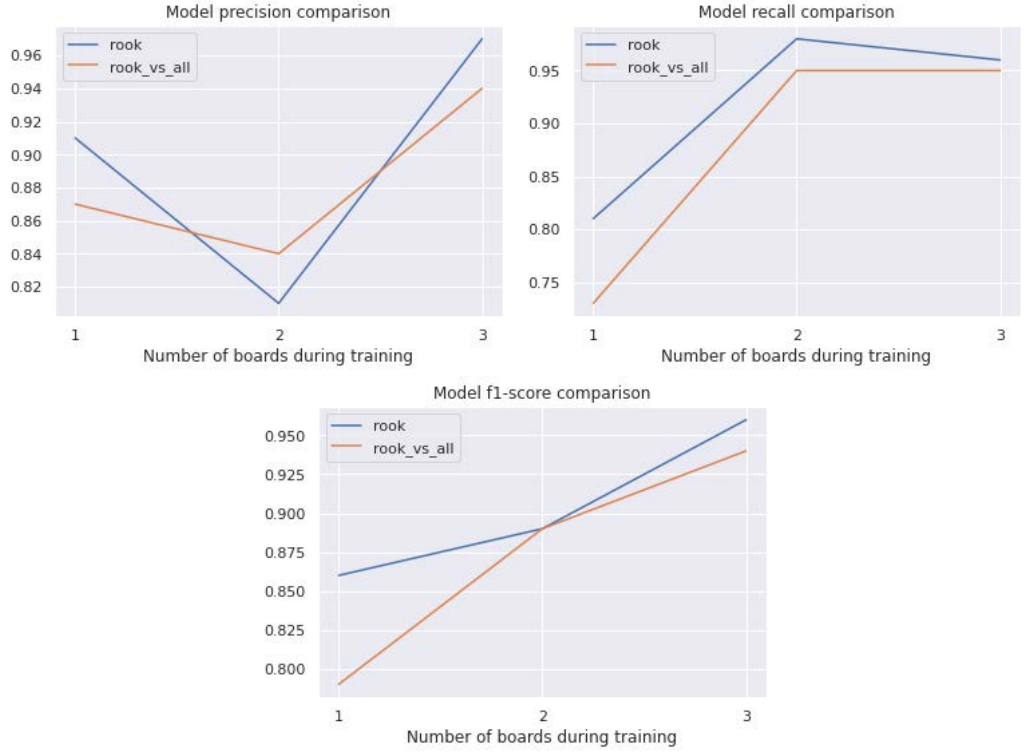

Fig. 3.8. Rook vs Rook vs All

Again, figure 3.8 tells a similar story. The Rook vs All is a strong classifier that results in a strong performance of the class in the overall One vs All resulting model.

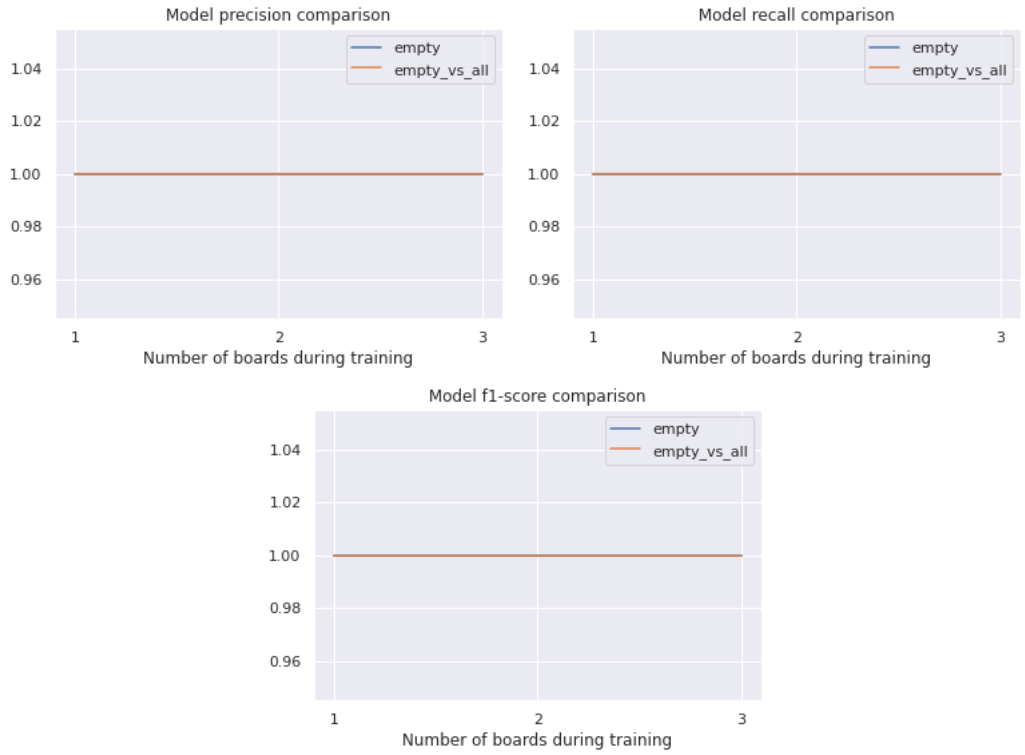

Fig. 3.9. Empty vs Empty vs All

Lastly, figure 3.9 shows that neither the Empty vs All nor the One vs All models had any trouble whatsoever identifying the empty squares, as they bored scored perfect metrics independently of the size of the training dataset.

Fig. 3.10. One vs All confusion Matrix

In conclusion, figure 3.10 reflects the outstanding performance of the One vs All model when trained with all the training data available. The one significant mistake it makes is confusing a small number of kings for bishops and vice versa. Nevertheless, the performance of the One vs All model is outstanding.

3.2. Color identification

After running the K-means algorithm on the mean pixel value of the training dataset after being applied mask M defined in section 2.3, with 2 centroids initialized at 0 and 255 respectively. The results obtained can be seen in figure 3.11.

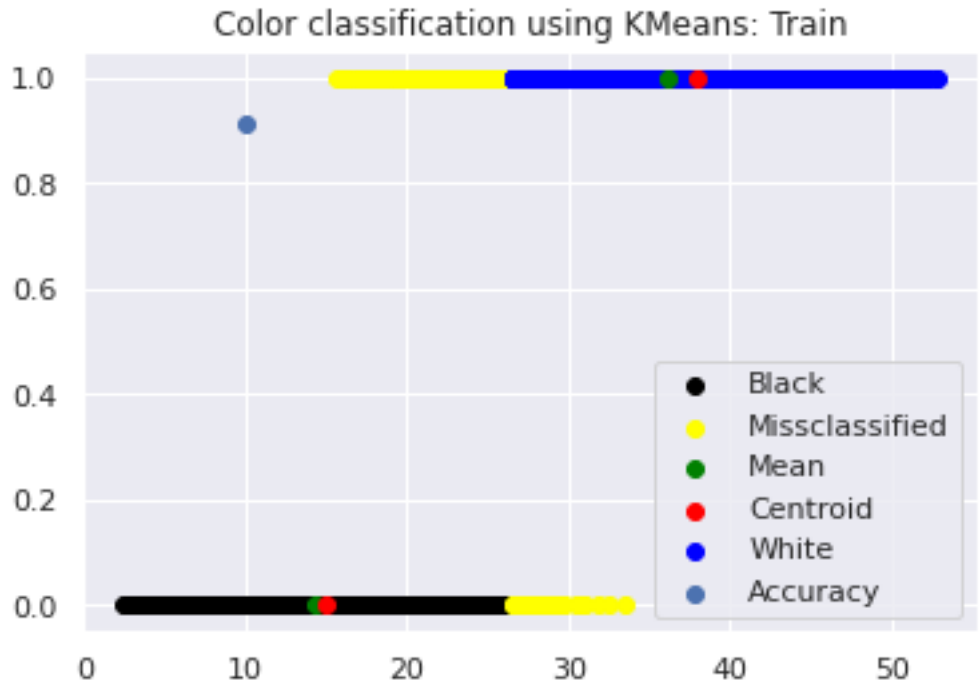

Fig. 3.11. K-means on training images for color identification.

It is to be noted that the centroids that the K-means algorithm found are close to the mean value of the black images and white images respectively. Also worth mentioning are all the miss classified images, highlighted with yellow. As one would expect, the miss classified images can be found on the extremes of the black images and white images that are closer to each other.

Nevertheless, the accuracy obtained by classifying the color of a chess piece based on the distance of its mean pixel value after being applied M is 82%. Let us observe what happens when we try to identify the color of the validation dataset using the found centroids, represented in figure 3.12.

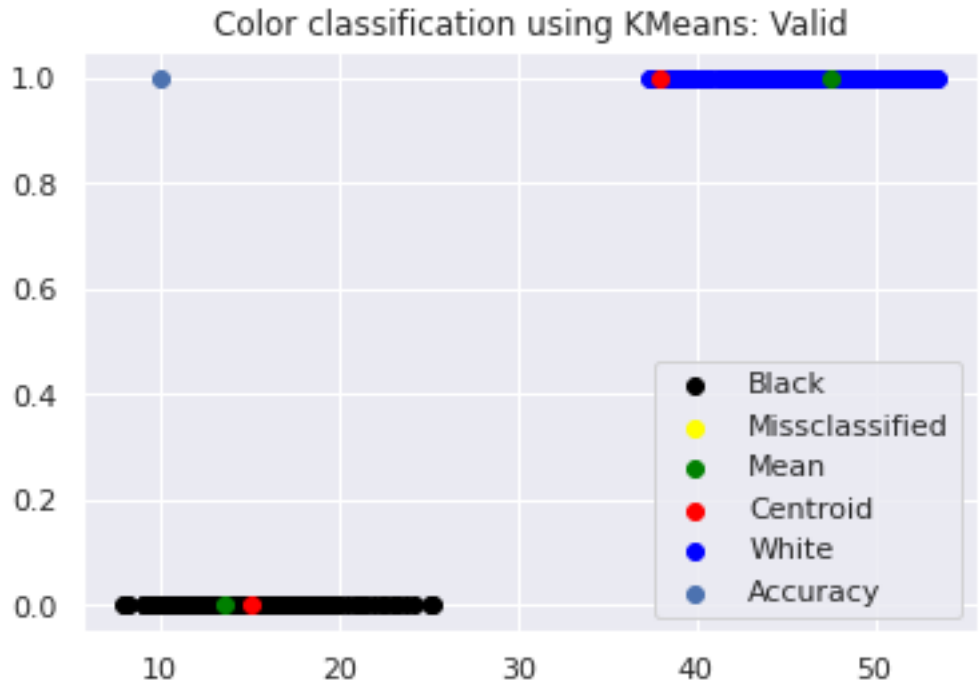

Fig. 3.12. Color identification on validation set using training clusters.

Although the centroids are far away from the mean values of the validation data, they suffice as it obtains perfect accuracy in identifying the color of this chess set. Nevertheless, each chess board recreation will need to run k-means to find its own two color clusters from scratch for the algorithm to adapt to the color set of each board. Let us observe the results of running k-means on the validation set, as seen in figure 3.13.

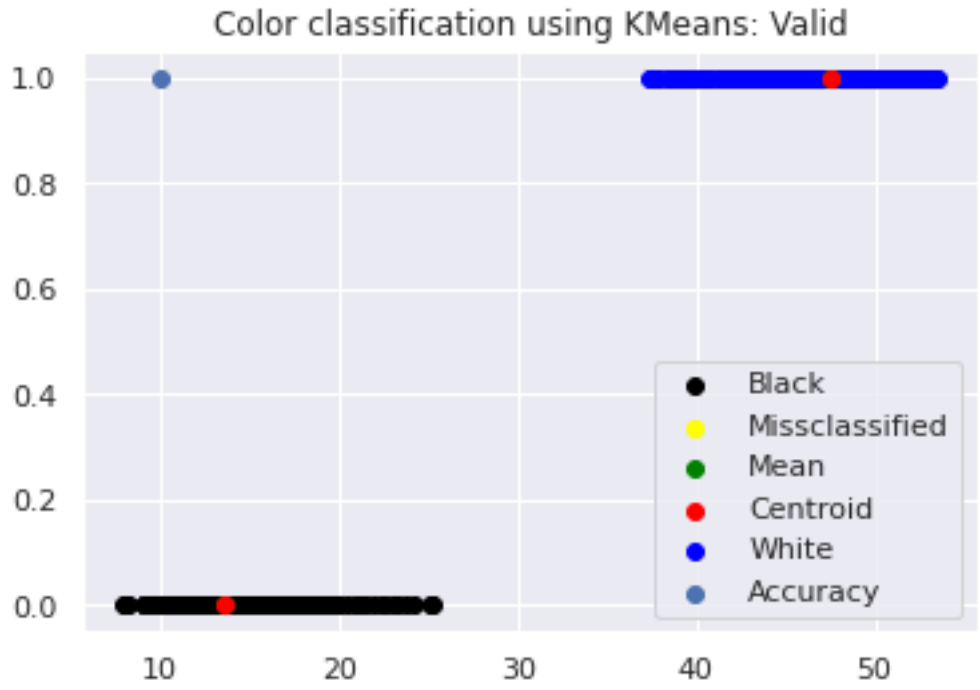

Fig. 3.13. K-means on validation images for color identification.

The same accuracy is achieved as when the validation set was tested against the clusters found using the training data: 100%. This time, however, the centroid of each cluster matches exactly the mean of the color they represent. Given that the colors used to differentiate the pieces are different enough, this approach will succeed in identifying both color clusters and classifying pieces accordingly.

3.3. Chess board recreation

Figure 3.14 is the chess board position that I shall attempt to recreate using both the 13 labels model and the 7 labels ensemble model, together with its actual digital representation.

Fig. 3.14. Picture of chess board position to be recreated digitally (left) and its digital representation (right). Taken from K. Berndtsson Kullberg - S. J. Bjurulf, Sweden 1920, move 6 white to play.

Figure 3.15 is the digital representation of the presented game made by the 13 label model. It makes some important mistakes, as failing to identify the black king. It does not fail at identifying the color of each piece, but is does make some important mistakes on the piece type.

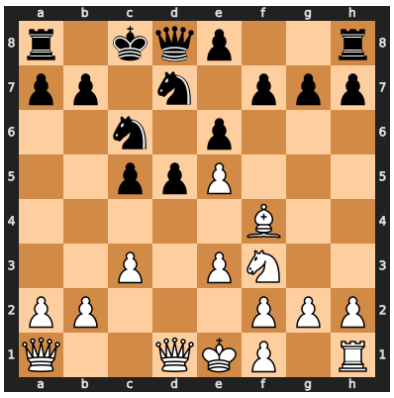

Fig. 3.15. Digital representation made by the 13 label model.

Figure 3.15 is the digital representation of the presented game made by the 13 labels model. It makes some important mistakes, such as failing to identify the black king. It does not fail at identifying the color of each piece but is does make some important mistakes on the piece type.

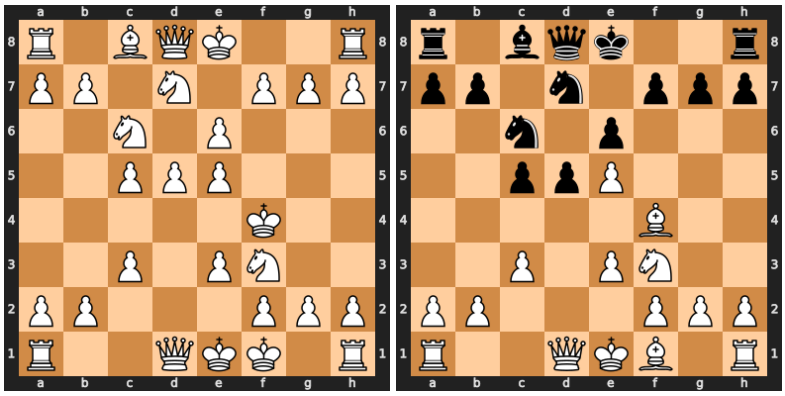

Fig. 3.16. Digital representation made by the 7 label ensemble model, uncolored (left) and colored (right).

4. CONCLUSIONS AND FUTURE WORK

4.1. Conclusions

Based on the results presented in the previous section, it is safe to say that the three objectives proposed for this project have been met with great success.

The first objective has been completely met. As the number of chess sets used during training increased, so did the model's performance on the independent validation set. With the increase in performance being highly significant for the 13 and 7 labels models, as well as for the One vs All ensemble model, it is safe to say that adding different chess sets to the training set enabled the classification models to identify the key features of each chess piece.

For the second objective, the work here presented illustrates that the 7 labels model outperformed the 13 labels model. The reason behind this increase in performance can be explained by the 7 label model having double the amount of data per class as a result of having to predict half as many chess pieces. This discovery is of great importance, as it heavily reduces the time and effort necessary to create a well-rounded dataset for this task, as to have as many images per label as the 7 labels model for the 13 labels model, a dataset of twice the size would be needed.

The project also highlights the importance of a rich and diverse dataset, as the models have been proven to perform better as the number of different training boards used during training increased. This can only mean that the more different types of chess pieces the model see, the more it can abstract its key features. Since the validation dataset was formed by a chess board not present in training, the models power of abstraction has been correctly tested, with great results as noted.

Moreover, the research has also found that using an ensemble model, in this case, a One vs All architecture following the 7 labels approach yields incredible results, heavily outperforming both the 7 and 13 labels models.

Continuing with the color identification approach necessary for the second objective, the research indicates that it is possible to infer the color of the chess pieces using the k-means clustering algorithm. Without this certainty, the 7 labels approach would be rendered of no use, as it completely depends on another system to find the color of each piece. The main advantage of the k-means algorithm is that it works for any two colors different enough in the color spectrum, thus not limiting the model to only being able to recognize black and white chess pieces.

Finishing with the last objective, this project proves that it is not necessary to first identify the chess board before identifying the chess pieces, as the works consulted in section 1.7.1 suggested. Identifying the chess board comes at an additional cost, both

computationally and of accessibility, as the boards usually have to be of different colors than black and white or have an inscription on their corners to aid detection. Therefore by eliminating this step, this work also eliminates barriers between the project and a future where it is easily used by everyday chess enthusiasts, with no previous computer vision knowledge. More on usability and a non-programmatic interface with the project in section 4.2.2.

Overall, this project has proven that previous restrictions on this task, such as the color of the chess pieces and the design of the chess board are not necessary to achieve competitive results in recreating chess board positions.

4.2. Future work

4.2.1. Enhancing predictions

As it has been discussed previously, the chess board recreation is conducted locally, square by square rather than by the whole board at once. This brings many problems, such as the possible appearance of several kings, for which a solution was proposed in section 2.4.

However this is only a small example of a wider problem, which is that the model currently has no means to prevent illegal positions. For example, the board recreation made by the 13 label model showed a black pawn on the eighth row, as seen in figure 3.15.

A possible solution could be to store all the probability vectors computed by the model for each square, and compute the legal position with the highest probability. This topic has not been attempted for this project, but it could be done in the future. I pose that search algorithms could prove useful in solving this task.

4.2.2. Usability

As of now the work done for this project is only accessible via a programmatic interface. That is, only programmatically a user can feed the model a picture of a chess board for it to recreate its position in digital format.

Needless to say, this is highly impractical beyond the academic world. A better interface for taking advantage of the work conducted in this project is through a web or mobile app, which could easily take a picture of a chess board and feed it to the model.

This solution could even work without the need for a production server. The traditional server architecture would suggest a solution where the user interface would send the picture to a server hosting the model and return its prediction. However, there exist deep learning libraries for running deep learning models in popular UI programming languages such as Javascript [25], eliminating the need for a sever and thus reducing the cost of developing and maintaining such an app.

BIBLIOGRAPHY

[1] P. T. Campos, *Chess piece dataset for image classification*, version v1.0.0, Zenodo, Jun. 2022. DOI: 10.5281/zenodo.6656212. [Online]. Available: https://doi.org/10.5281/zenodo.6656212.

[2] F. J. Díaz-Pernas, M. Martínez-Zarzuela, M. Antón-Rodríguez, and D. González-Ortega, "A deep learning approach for brain tumor classification and segmentation using a multiscale convolutional neural network," *Healthcare*, vol. 9, no. 2, 2021. DOI: 10.3390/healthcare9020153. [Online]. Available: https://www.mdpi.com/2227-9032/9/2/153.

[3] A. Underwood, *Board game image recognition using neural networks*, 2020. [Online]. Available: https://towardsdatascience.com/board-game-image-recognition-using-neural-networks-116fc876dafa.

[4] K. Tam, J. Lay, and D. Levy, "Automatic grid segmentation of populated chessboard taken at a lower angle view," in *2008 Digital Image Computing: Techniques and Applications*, 2008, pp. 294–299. DOI: 10.1109/DICTA.2008.40.

[5] C. Koray and E. Sümer, "Chessvision: Chess board and piece recognition," *21st Computer Vision Winter Workshop, Rimske Toplice, Slovenia*, 2016. [Online]. Available: https://vision.fe.uni-lj.si/cvww2016/proceedings/papers/21.pdf.

[6] C. Danner and M. Kafafy, *Visual chess recognition*, 2016. [Online]. Available: https://web.stanford.edu/class/ee368/Project_Spring_1415/Reports/Danner_Kafafy.pdf.

[7] J. Ding, *Chessvision: Chess board and piece recognition*, 2016. [Online]. Available: https://web.stanford.edu/class/cs231a/prev_projects_2016/CS_231A_Final_Report.pdf.

[8] D. H. Hubel and T. N. Wiesel, "Receptive fields of single neurones in the cat's striate cortex," *The Journal of physiology*, Oct. 1959. DOI: 10.1113/jphysiol.1959.sp006308.

[9] Y. Le Cun *et al.*, "Handwritten digit recognition: Applications of neural network chips and automatic learning," *IEEE Communications Magazine*, vol. 27, no. 11, pp. 41–46, 1989. DOI: 10.1109/35.41400.

[10] Y. LeCun. "Lenet-5, convolutional neural networks." (), [Online]. Available: http://yann.lecun.com/exdb/lenet/ (visited on 05/2022).

[11] M. Everingham, L. V. Gool, C. K. I. Williams, J. M. Winn, and A. Zisserman, "The pascal visual object classes (voc) challenge," *International Journal of Computer Vision*, vol. 88, pp. 303–338, 2009.

[12] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 2009, pp. 248–255. DOI: 10.1109/CVPR.2009.5206848.

[13] O. Russakovsky *et al.*, "Imagenet large scale visual recognition challenge," *International Journal of Computer Vision*, vol. 115, Sep. 2014. DOI: 10.1007/s11263-015-0816-y.

[14] ImageNet. "Imagenet webpage." (), [Online]. Available: https://image-net.org (visited on 05/2022).

[15] J. Howard and S. Gugger, *Deep Learning for Coders with fastai and PyTorch*. O'Reilly Media, Inc., 2020.

[16] C. Tan, F. Sun, T. Kong, W. Zhang, C. Yang, and C. Liu, *A survey on deep transfer learning*, 2018. DOI: 10.48550/ARXIV.1808.01974. [Online]. Available: https://arxiv.org/abs/1808.01974.

[17] G. Van Horn, E. Cole, S. Beery, K. Wilber, S. Belongie, and O. Mac Aodha, *Benchmarking representation learning for natural world image collections*, 2021. DOI: 10.48550/ARXIV.2103.16483. [Online]. Available: https://arxiv.org/abs/2103.16483.

[18] M. R. H. Taher, F. Haghighi, R. Feng, M. B. Gotway, and J. Liang, *A systematic benchmarking analysis of transfer learning for medical image analysis*, 2021. DOI: 10.48550/ARXIV.2108.05930. [Online]. Available: https://arxiv.org/abs/2108.05930.

[19] R. Gençay and M. Qi, "Pricing and hedging derivative securities with neural networks: Bayesian regularization, early stopping, and bagging," *Neural Networks, IEEE Transactions on*, vol. 12, pp. 726–734, Aug. 2001. DOI: 10.1109/72.935086.

[20] MathWorks. "Introduction to deep learning: What are convolutional neural networks?" (2017), [Online]. Available: https://www.mathworks.com/videos/introduction-to-deep-learning-what-are-convolutional-neural-networks--1489512765771.html (visited on 05/2022).

[21] A. Géron, *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, 2nd Edition*. O'Reilly Media, Inc., 2019.

[22] A. Raju and S. Thirunavukkarasu, "Convolutional neural network demystified for a comprehensive learning with industrial application," in *Dynamic Data Assimilation*, D. G. Harkut, Ed., Rijeka: IntechOpen, 2020, ch. 3. DOI: 10.5772/intechopen.92091. [Online]. Available: https://doi.org/10.5772/intechopen.92091.

[23] K. He, X. Zhang, S. Ren, and J. Sun, *Deep residual learning for image recognition*, 2015. DOI: 10.48550/ARXIV.1512.03385. [Online]. Available: https://arxiv.org/abs/1512.03385.

[24] S. Lloyd, "Least squares quantization in pcm," *IEEE Transactions on Information Theory*, vol. 28, no. 2, pp. 129–137, 1982. DOI: 10.1109/TIT.1982.1056489.

[25] TensorFlow. "Tensorflow.js." (), [Online]. Available: https://www.tensorflow.org/js (visited on 05/2022).

[26] M. A. Czyzewski, A. Laskowski, and S. Wasik, *Chessboard and chess piece recognition with the support of neural networks*, 2017. DOI: 10.48550/ARXIV.1708.03898. [Online]. Available: https://arxiv.org/abs/1708.03898.

APPENDIX I: OTHER FIGURES

Confusion matrices

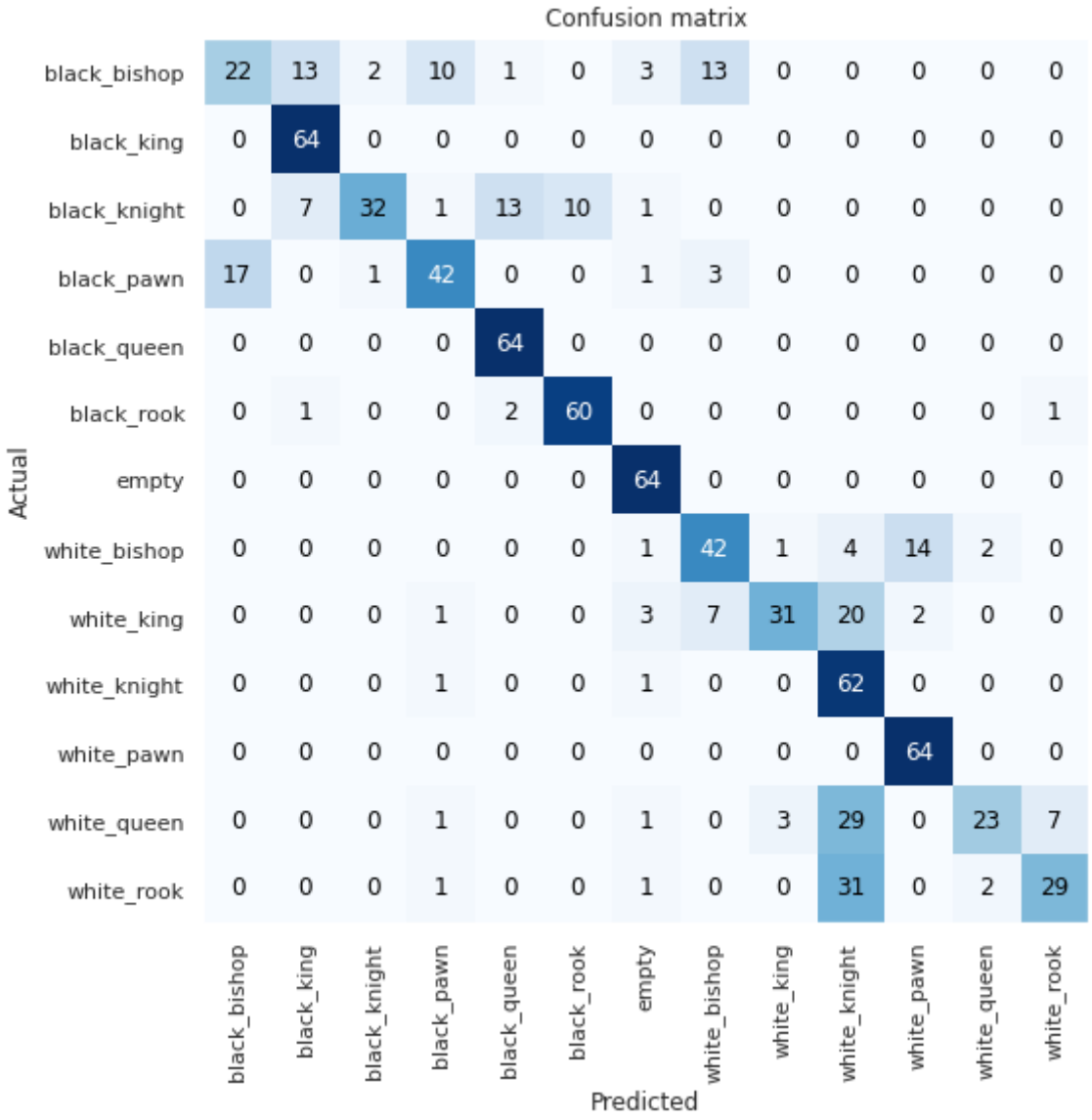

Fig. 4.1. 12 labels confusion matrix trained with 1 chess board.

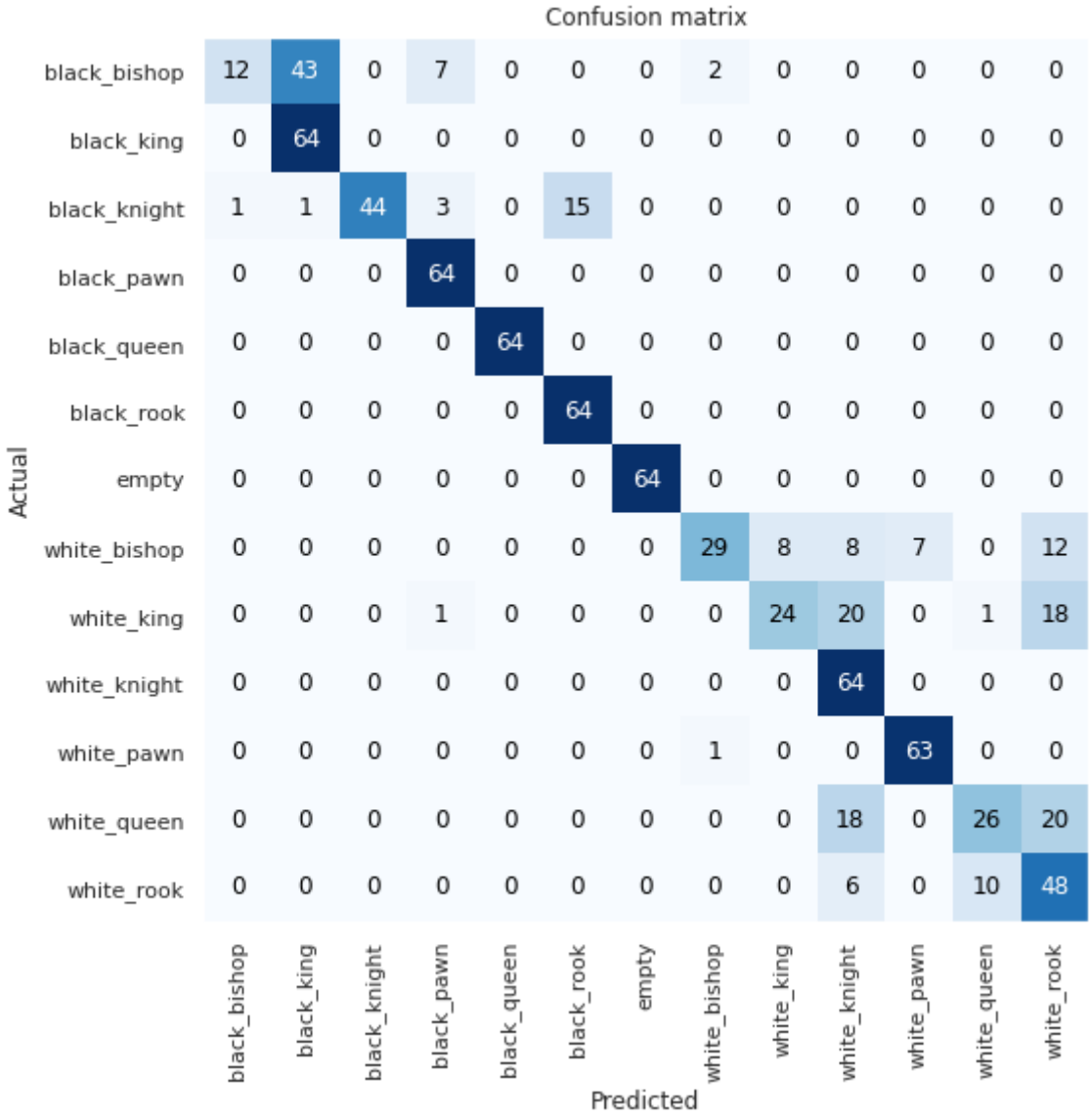

Fig. 4.2. 12 labels confusion matrix trained with 2 chess boards.

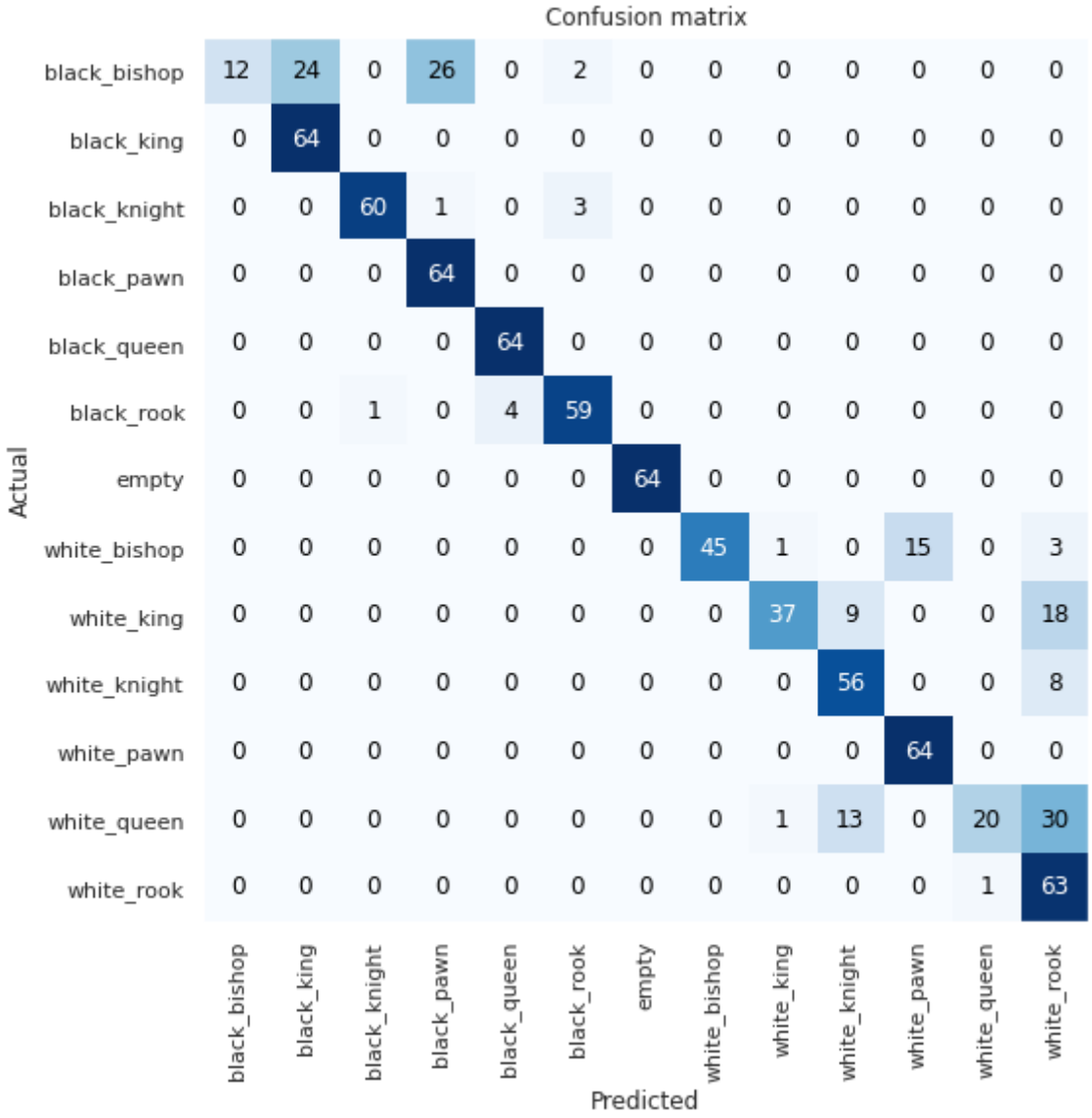

Fig. 4.3. 12 labels confusion matrix trained with 3 chess boards.

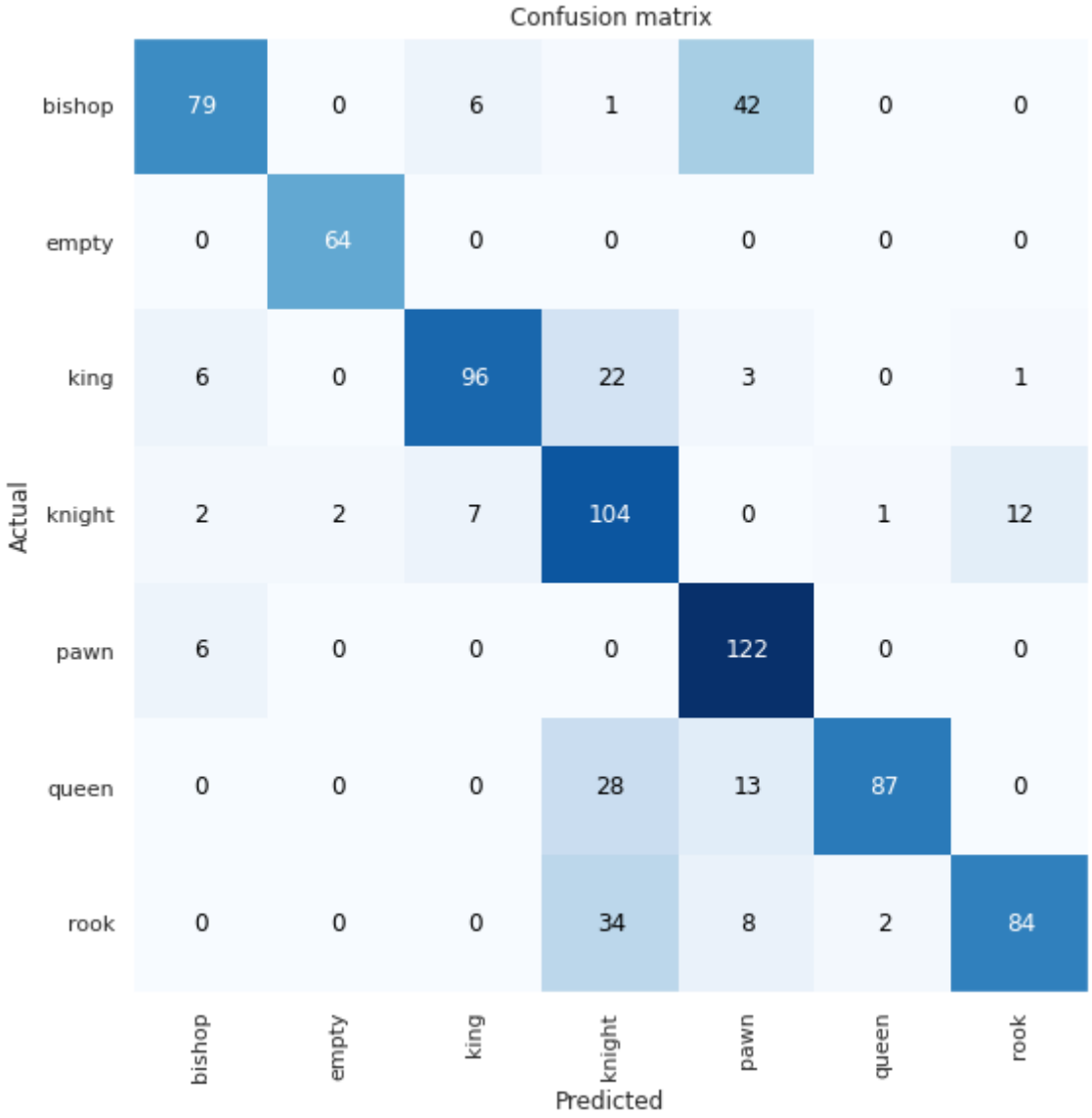

Fig. 4.4. 6 labels confusion matrix trained with 1 chess board.

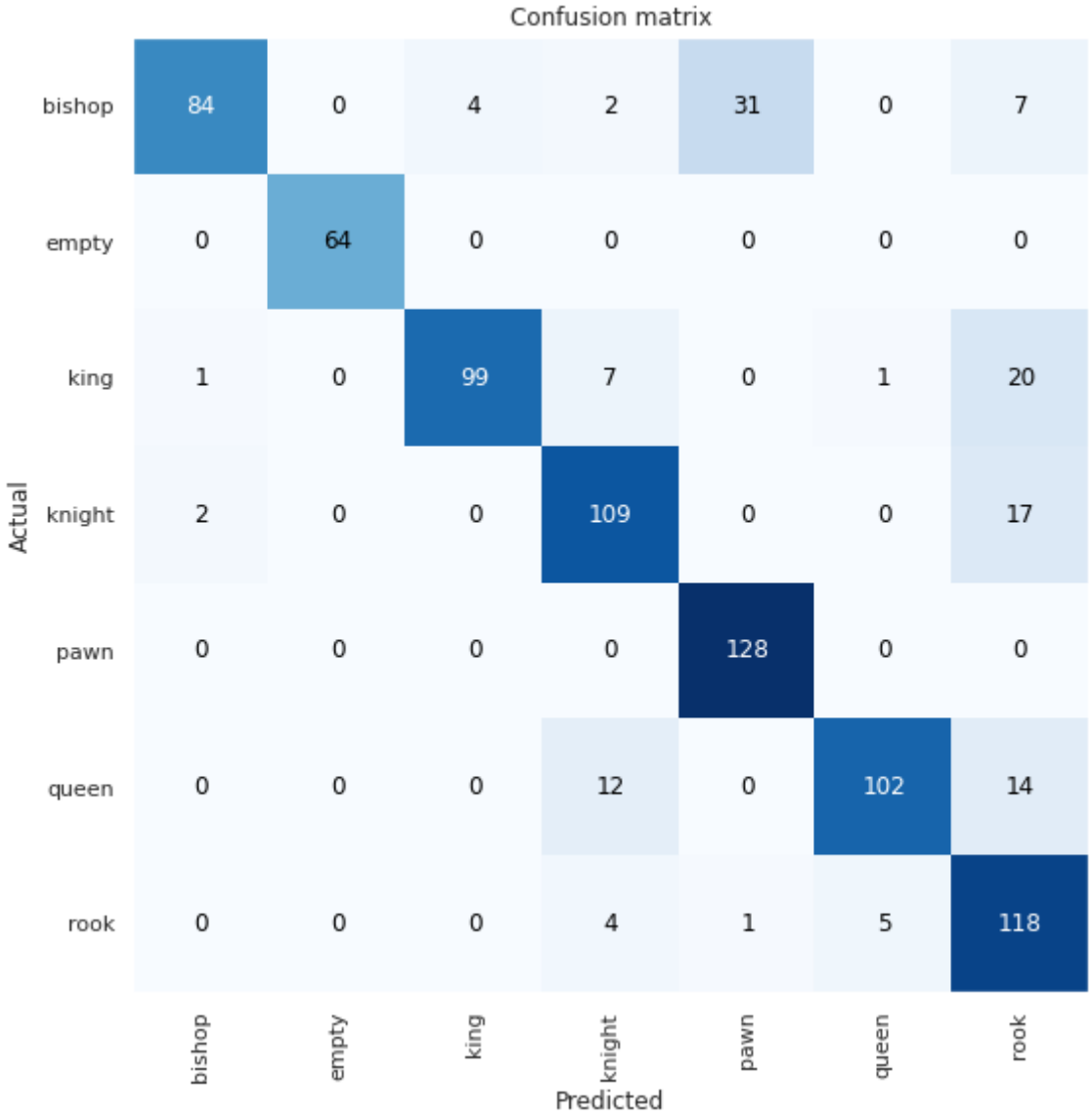

Fig. 4.5. 6 labels confusion matrix trained with 2 chess boards.

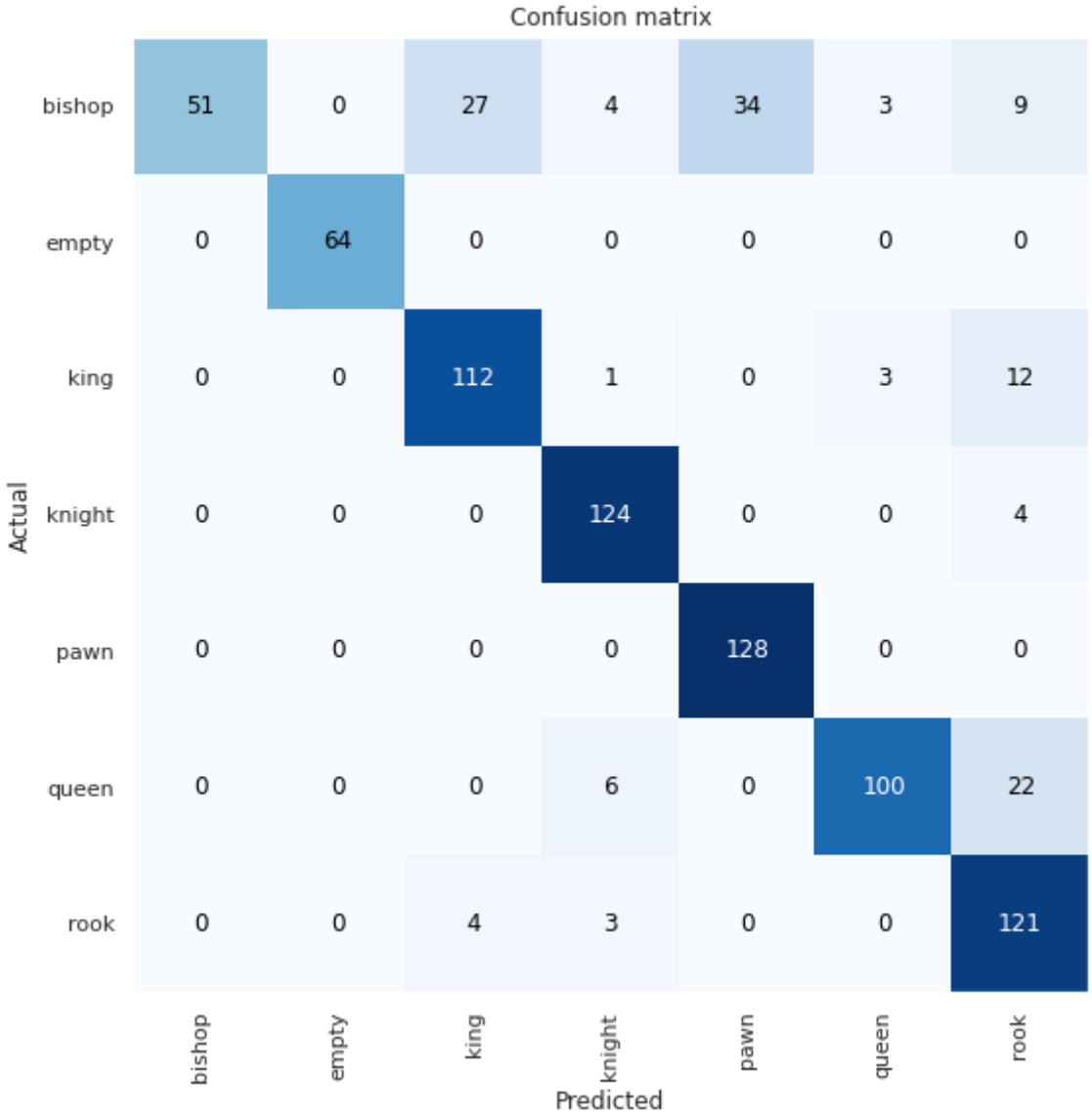

Fig. 4.6. 6 labels confusion matrix trained with 3 chess boards.

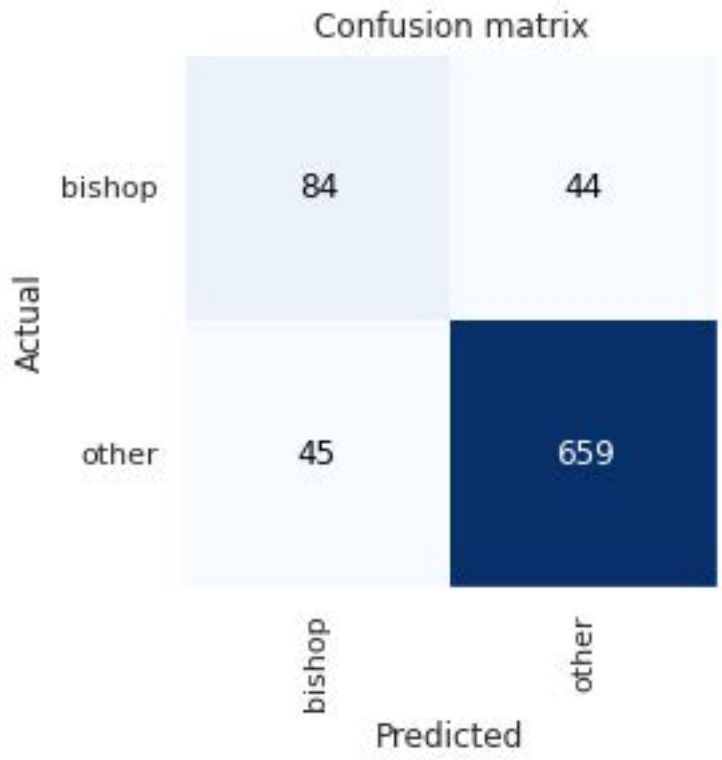

Fig. 4.7. Bishop vs All confusion matrix trained with 1 chess board.

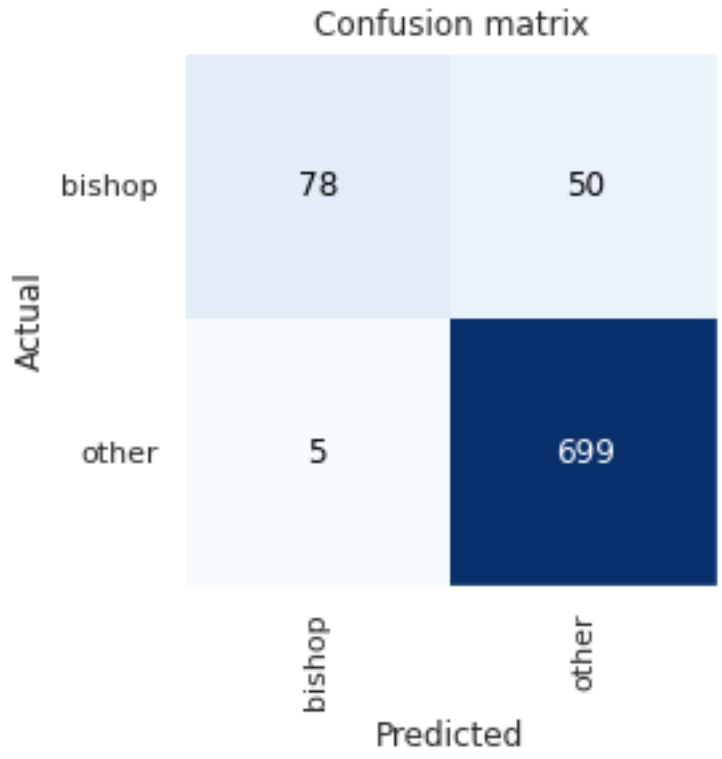

Fig. 4.8. Bishop vs All confusion matrix trained with 2 chess boards.

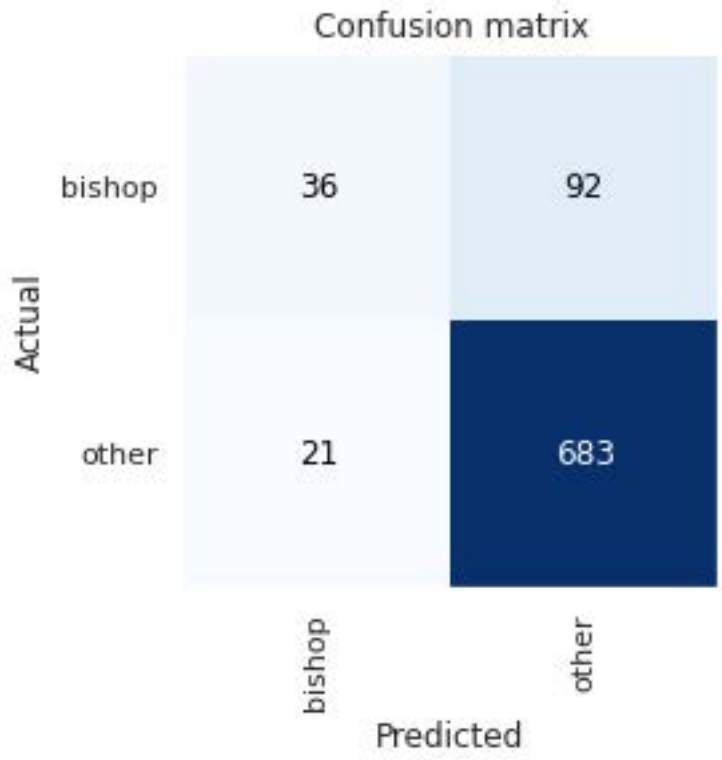

Fig. 4.9. Bishop vs All confusion matrix trained with 3 chess boards.

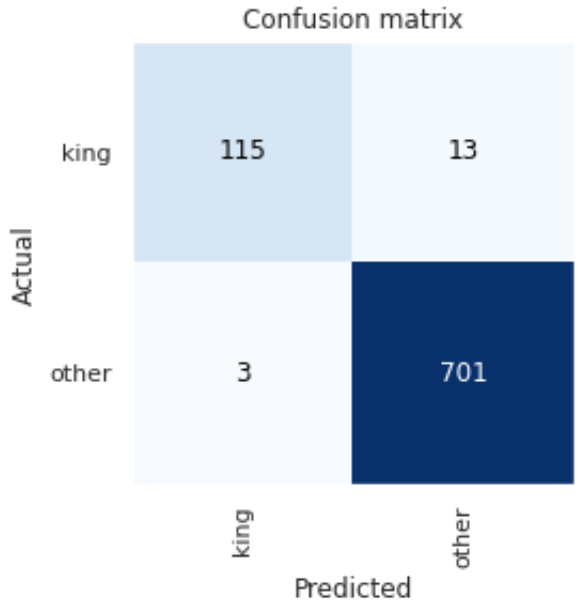

Fig. 4.10. King vs All confusion matrix trained with 1 chess board.

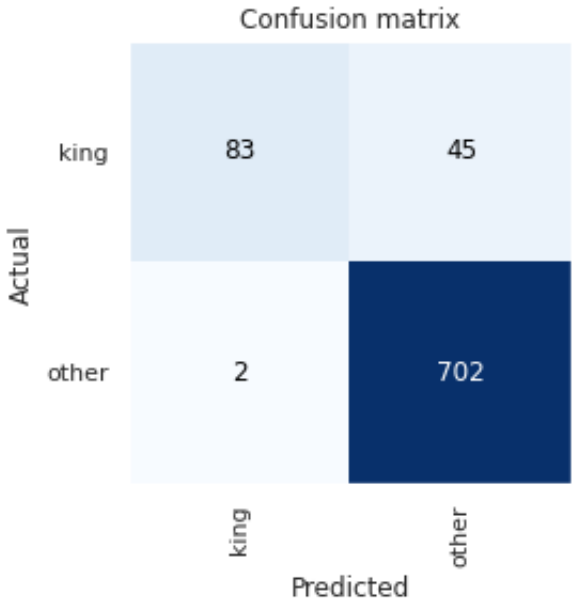

Fig. 4.11. King vs All confusion matrix trained with 2 chess boards.

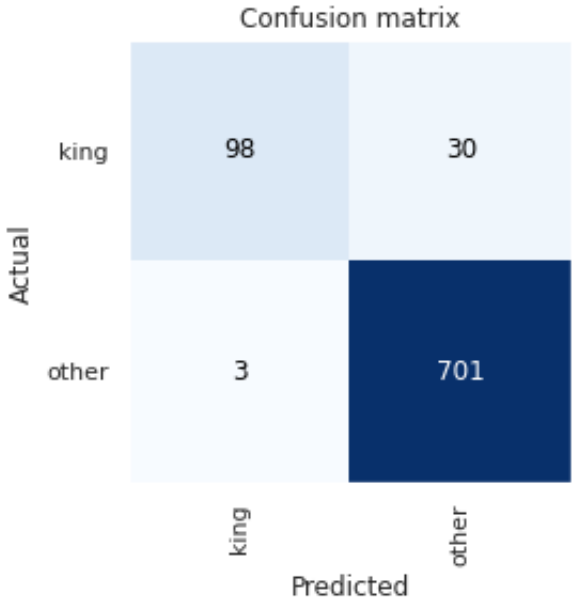

Fig. 4.12. King vs All confusion matrix trained with 3 chess boards.

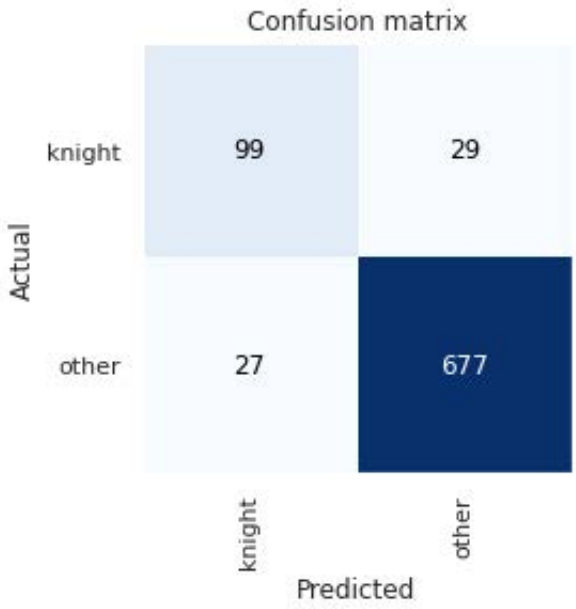

Fig. 4.13. Knight vs All confusion matrix trained with 1 chess board.

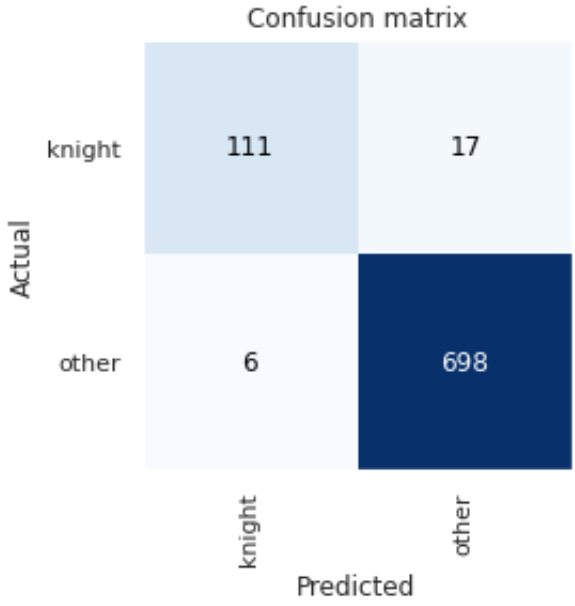

Fig. 4.14. Knight vs All confusion matrix trained with 2 chess boards.

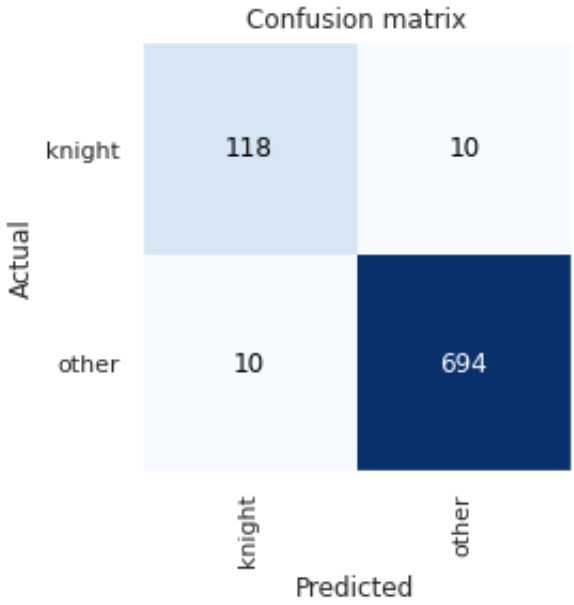

Fig. 4.15. Knight vs All confusion matrix trained with 3 chess boards.

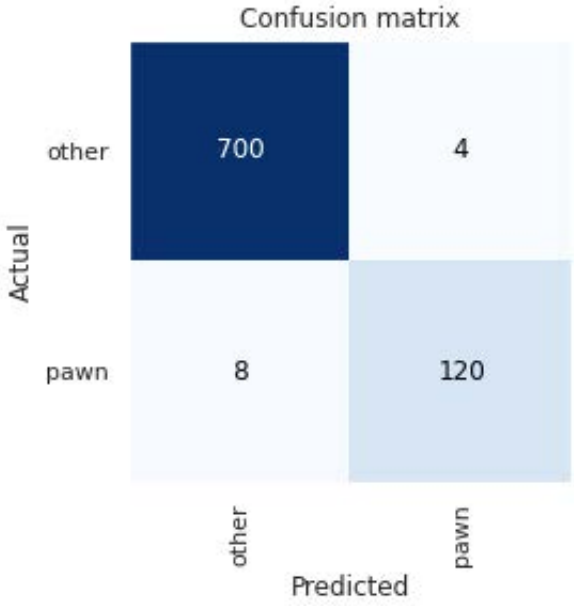

Fig. 4.16. Pawn vs All confusion matrix trained with 1 chess board.

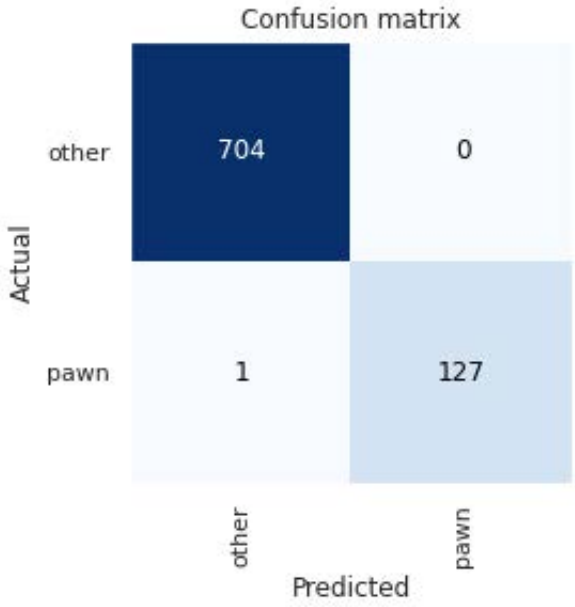

Fig. 4.17. Pawn vs All confusion matrix trained with 2 chess boards.

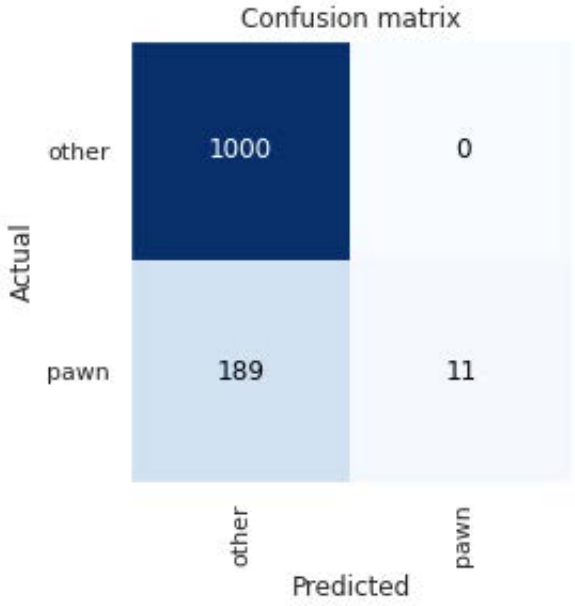

Fig. 4.18. Pawn vs All confusion matrix trained with 3 chess boards.

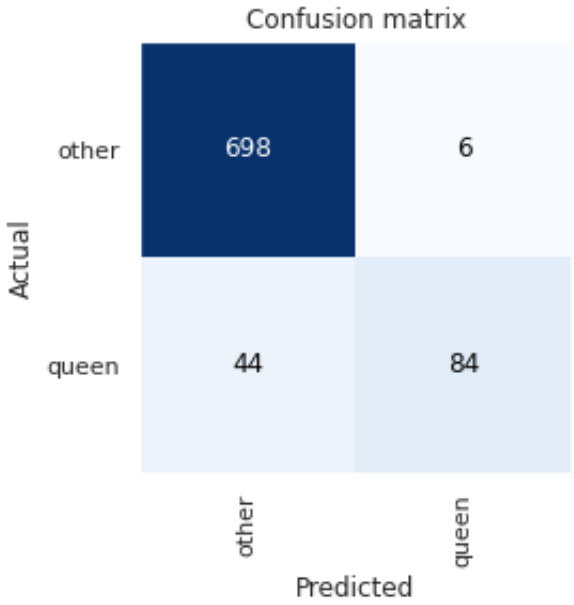

Fig. 4.19. Queen vs All confusion matrix trained with 1 chess board.

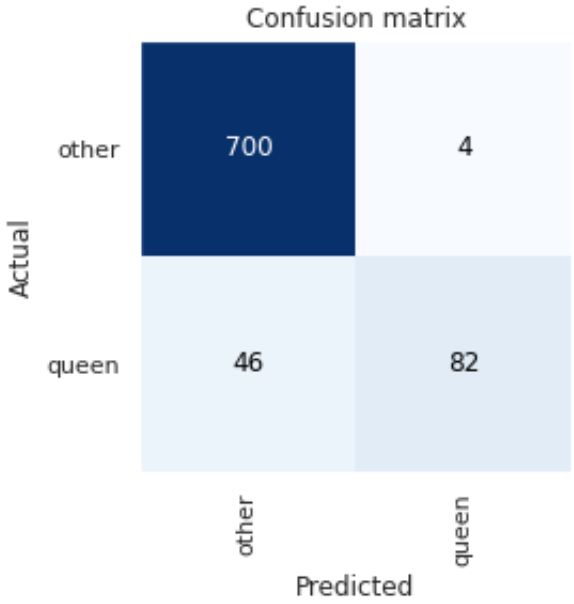

Fig. 4.20. Queen vs All confusion matrix trained with 2 chess boards.

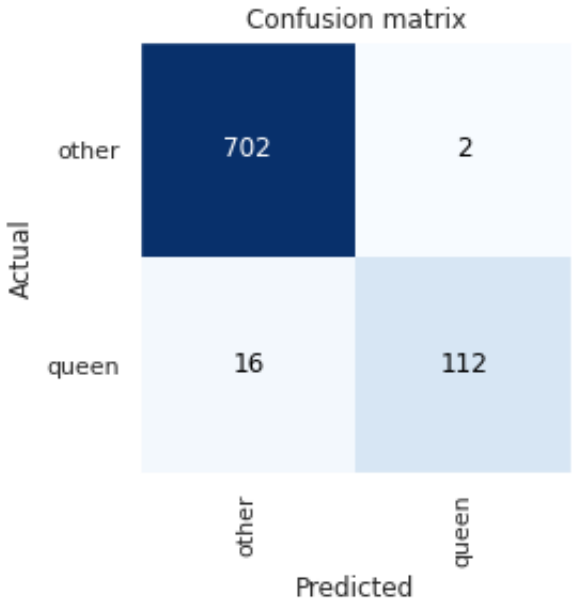

Fig. 4.21. Queen vs All confusion matrix trained with 3 chess boards.

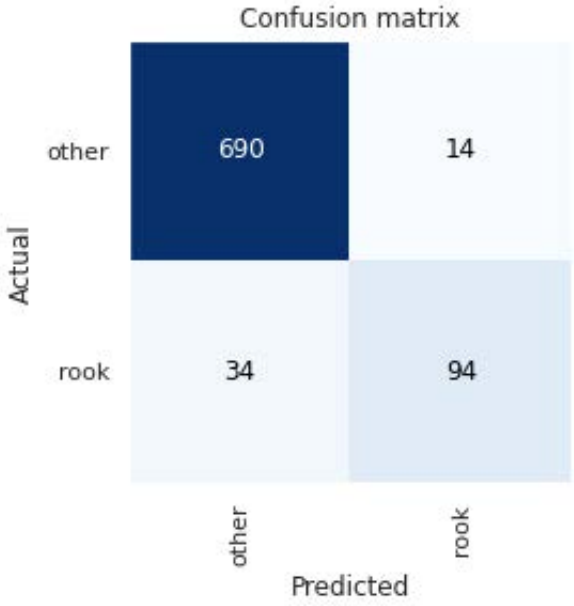

Fig. 4.22. Rook vs All confusion matrix trained with 1 chess board.

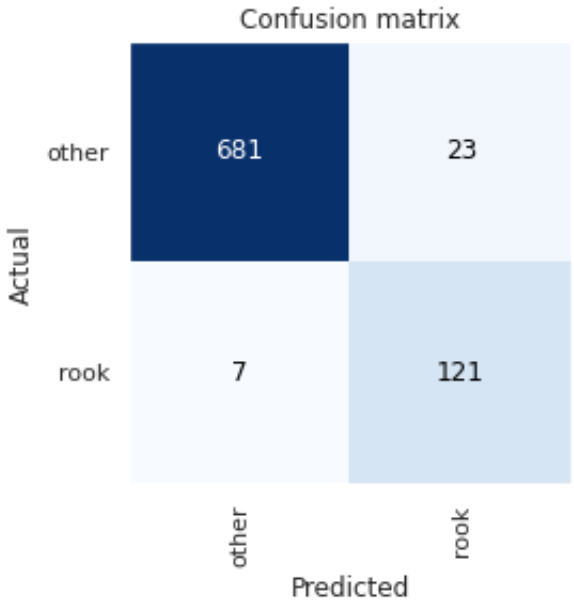

Fig. 4.23. Rook vs All confusion matrix trained with 2 chess boards.

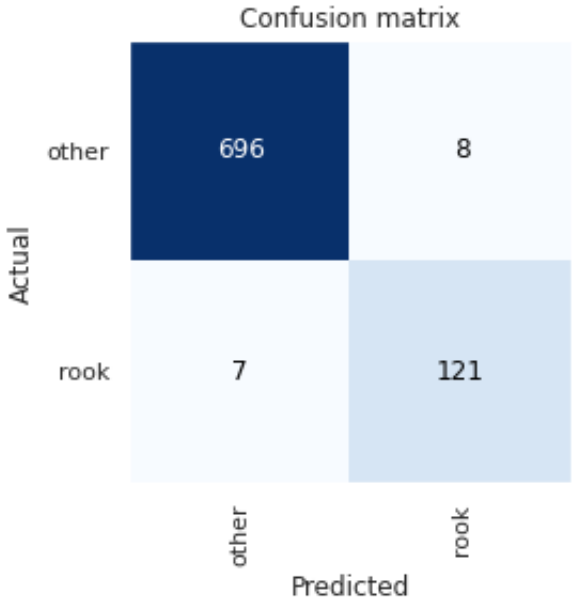

Fig. 4.24. Rook vs All confusion matrix trained with 3 chess boards.

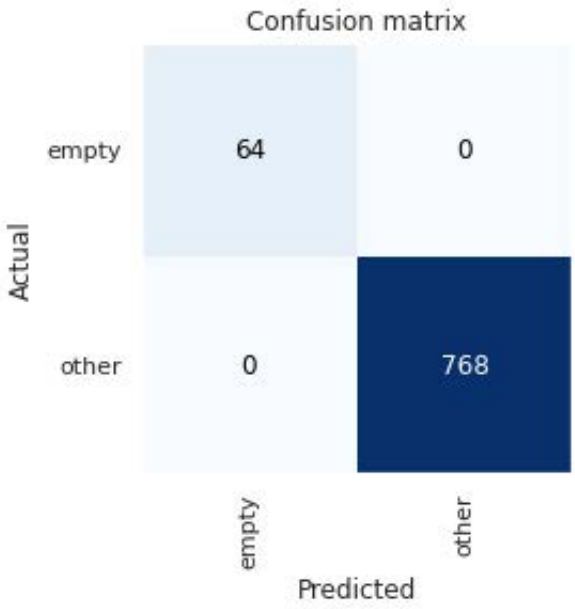

Fig. 4.25. Empty vs All confusion matrix trained with 1 chess board.

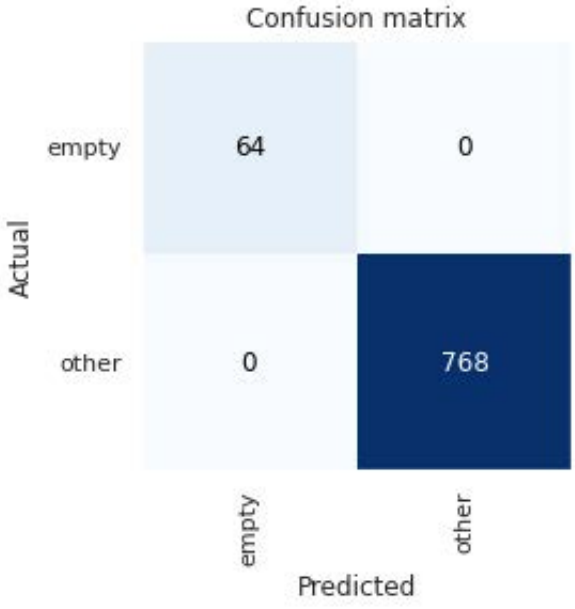

Fig. 4.26. Empty vs All confusion matrix trained with 2 chess boards.

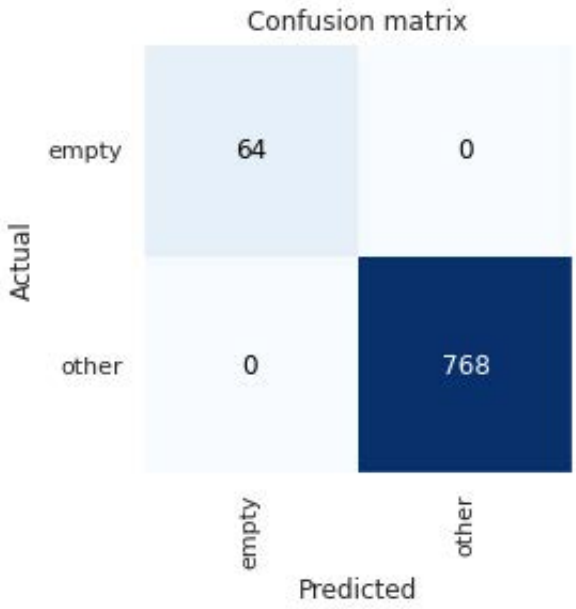

Fig. 4.27. Empty vs All confusion matrix trained with 3 chess boards.

Prediction Samples

Fig. 4.28. 12 labels prediction sample trained with 3 chess boards.

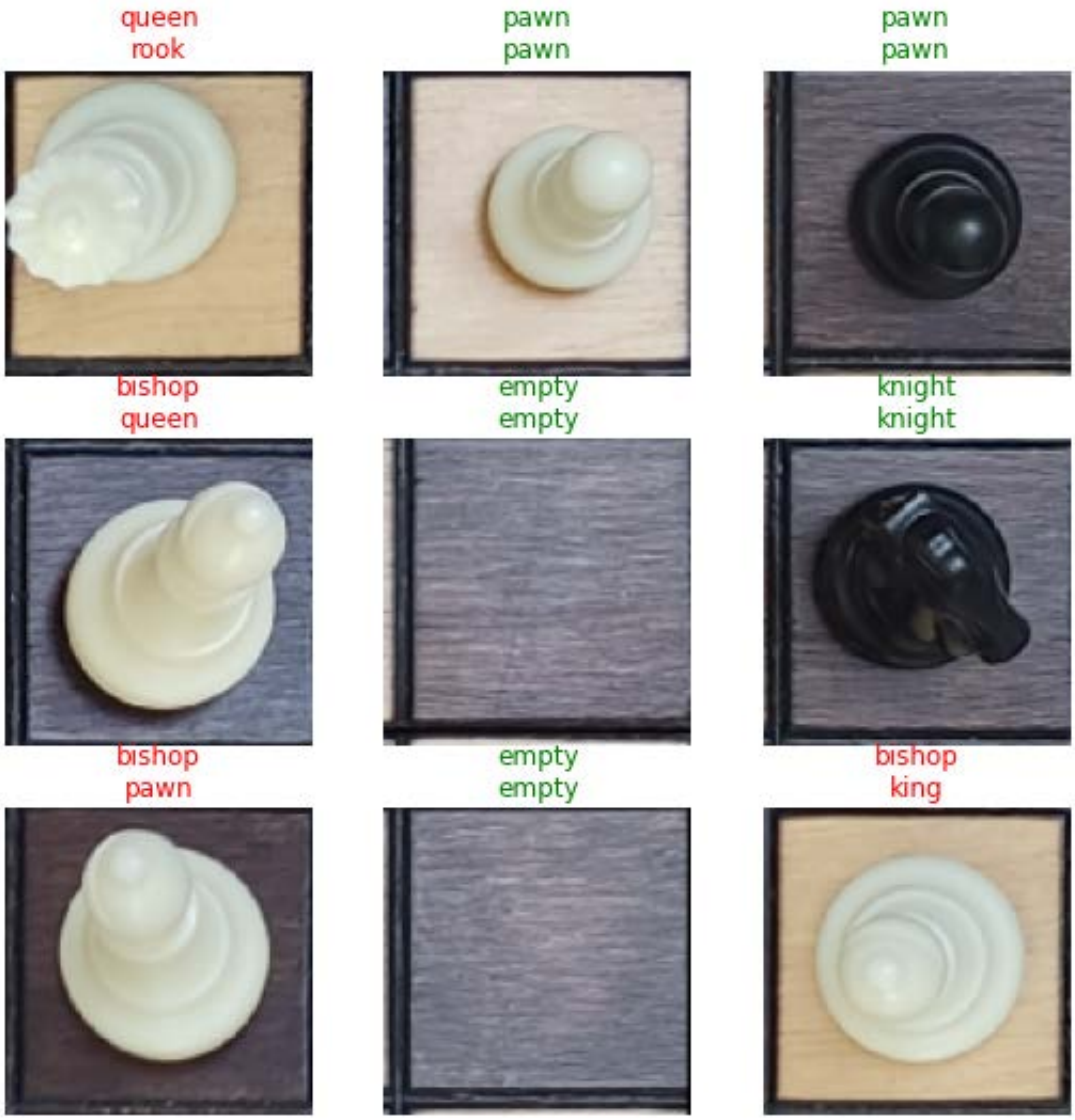

Fig. 4.29. 6 labels prediction sample trained with 3 chess boards.

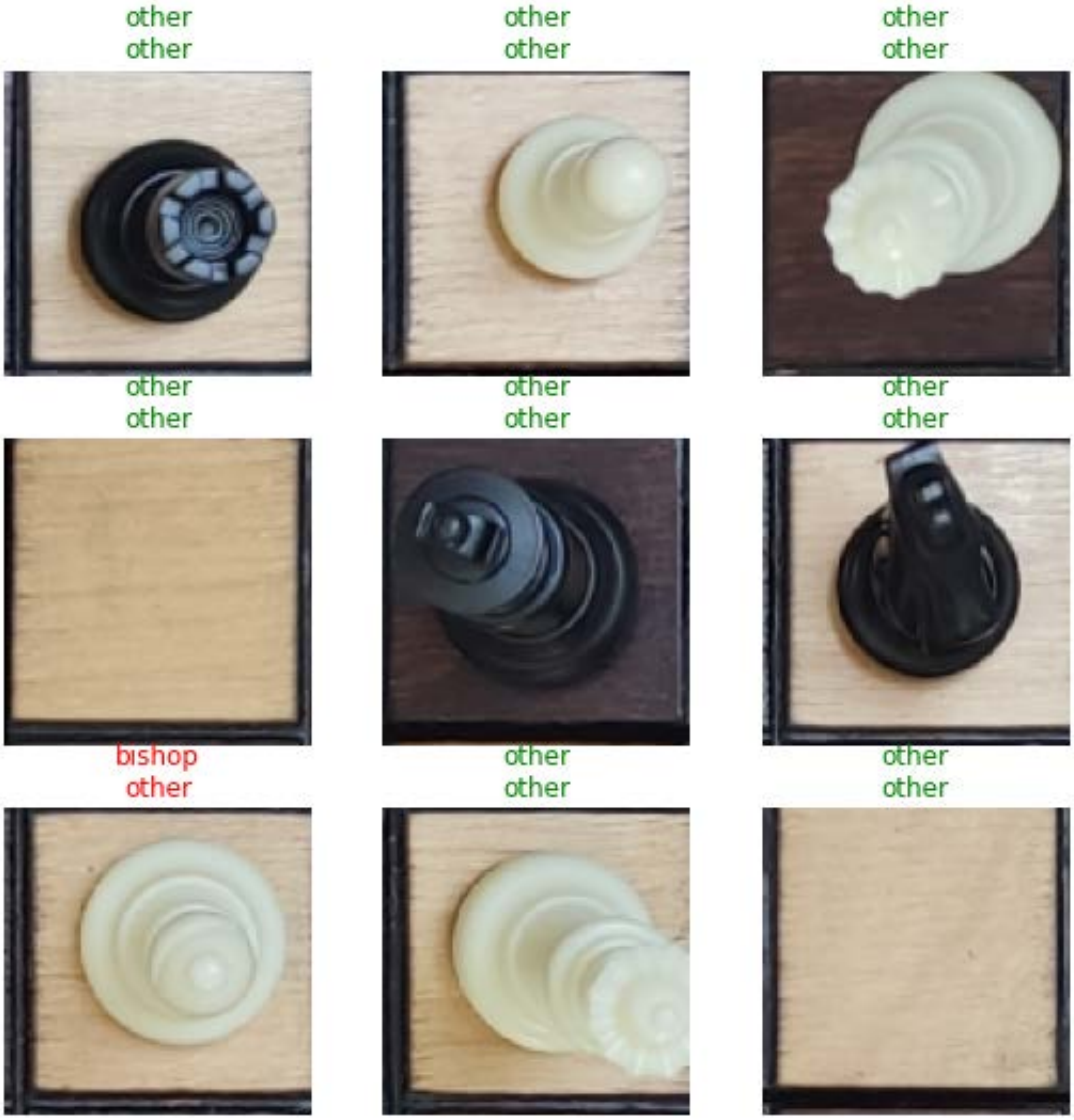

Fig. 4.30. Bishop vs All prediction sample trained with 3 chess boards.

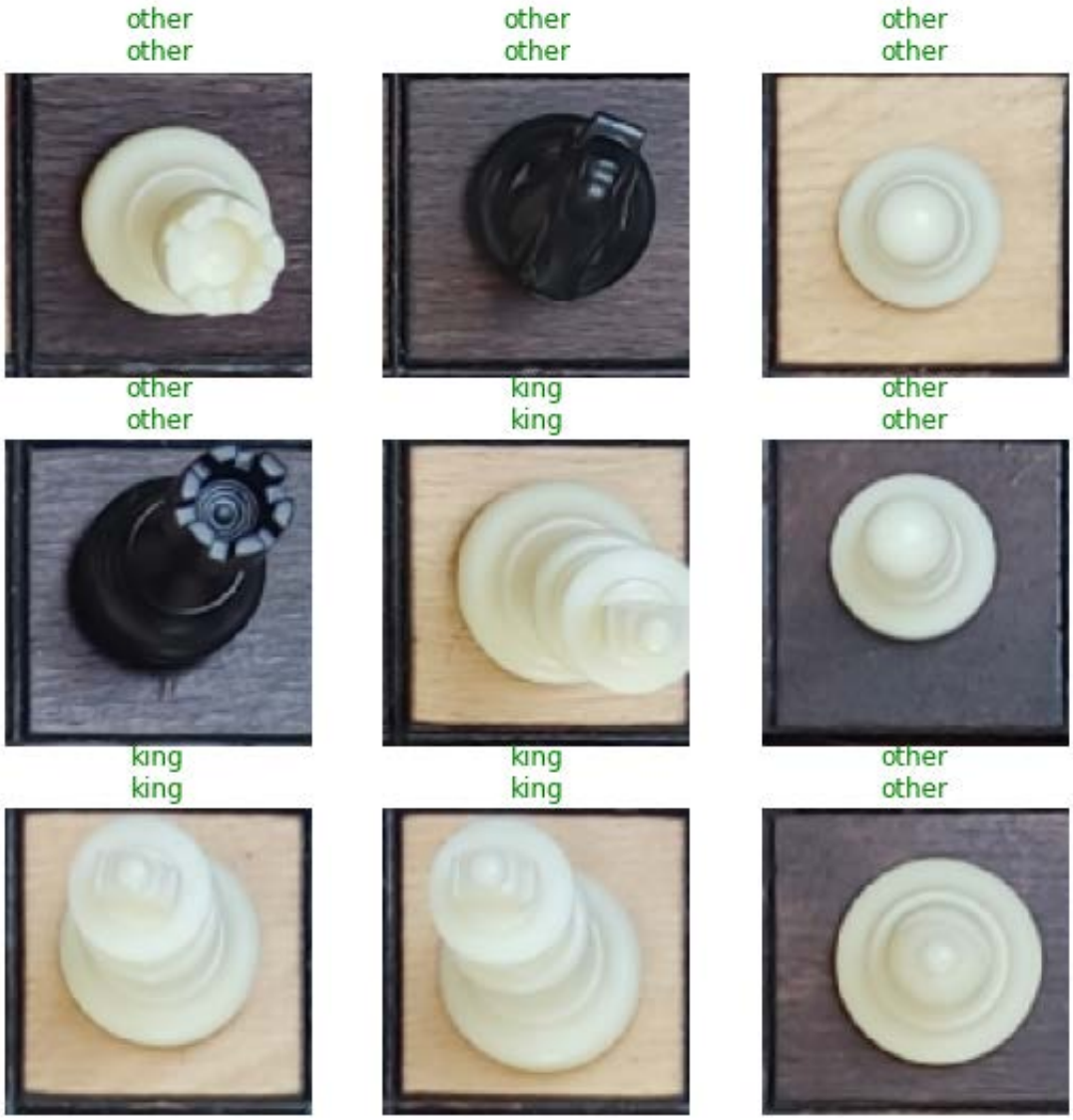

Fig. 4.31. King vs All prediction sample trained with 3 chess boards.

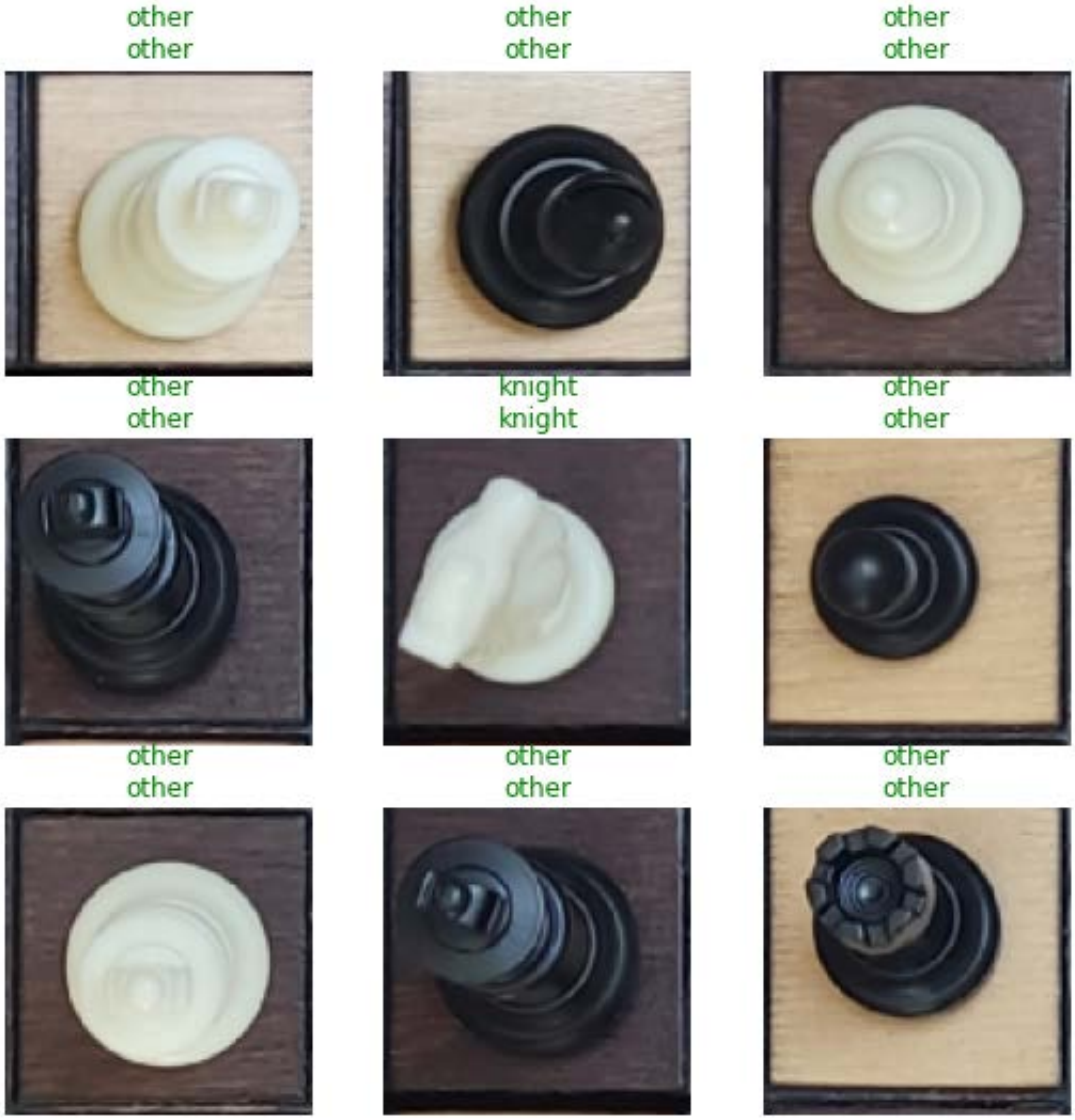

Fig. 4.32. Knight vs All prediction sample trained with 3 chess boards.

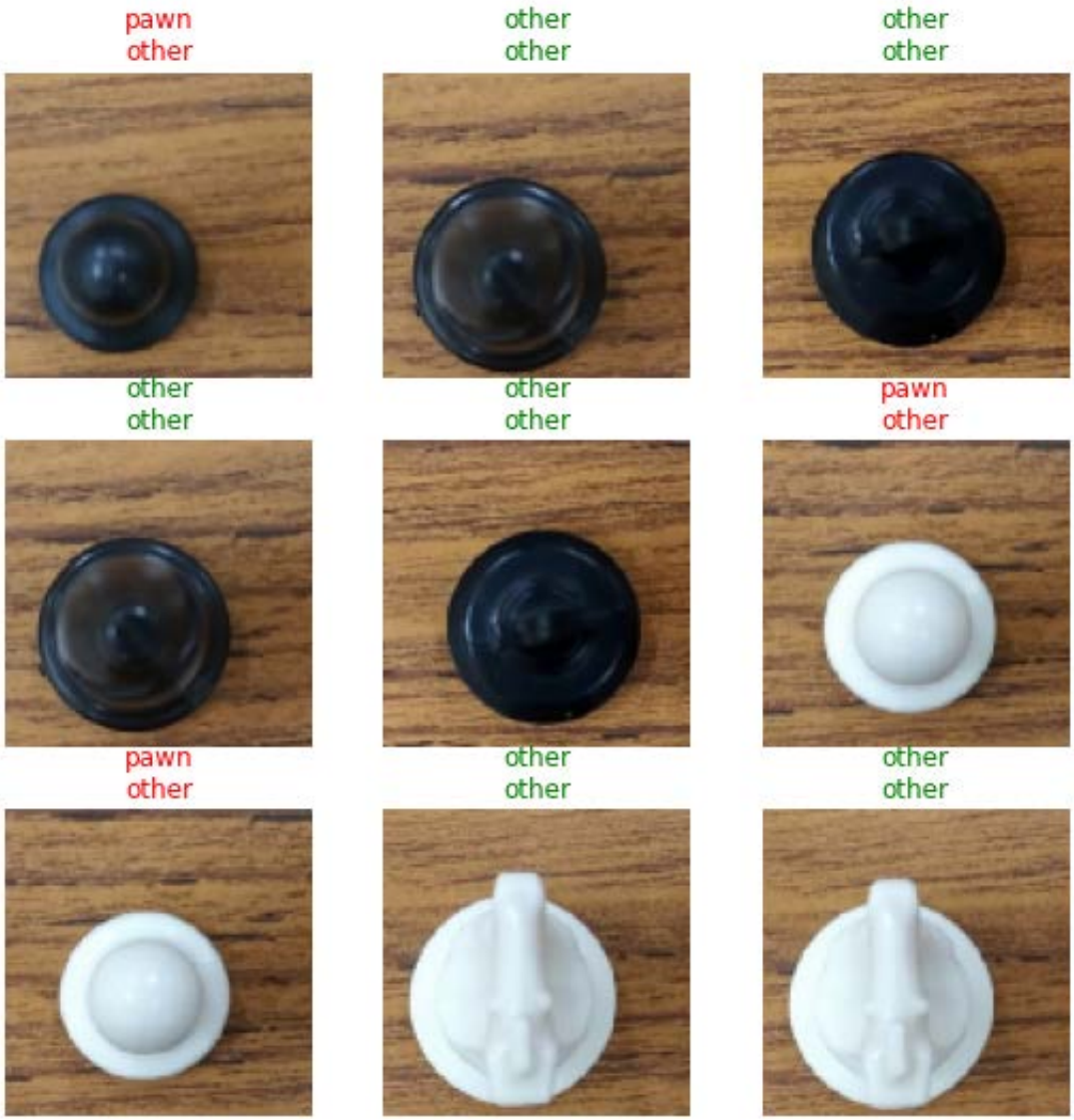

Fig. 4.33. Pawn vs All prediction sample trained with 3 chess boards.

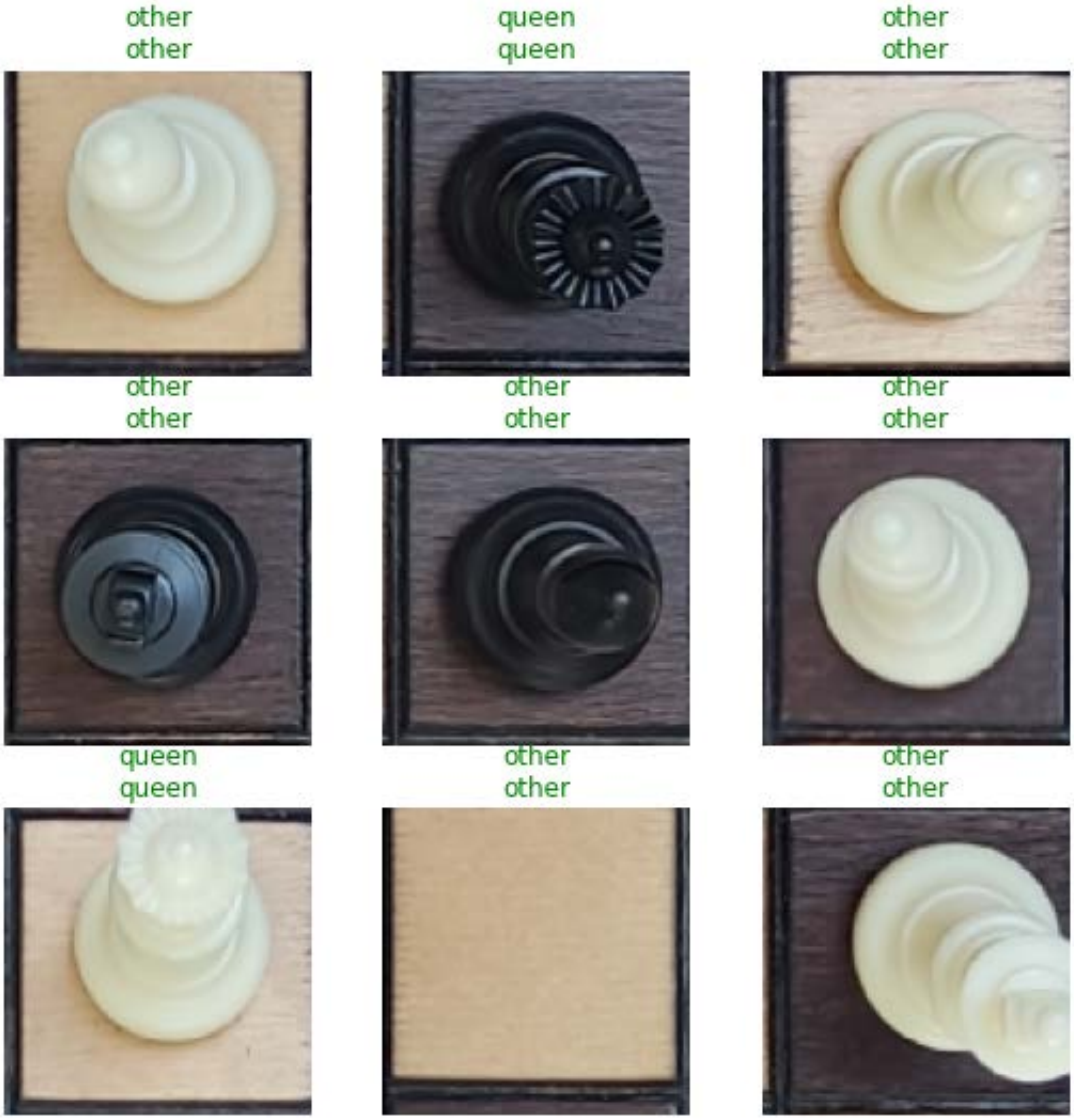

Fig. 4.34. Queen vs All prediction sample trained with 3 chess boards.

Fig. 4.35. Rook vs All prediction sample trained with 3 chess boards.

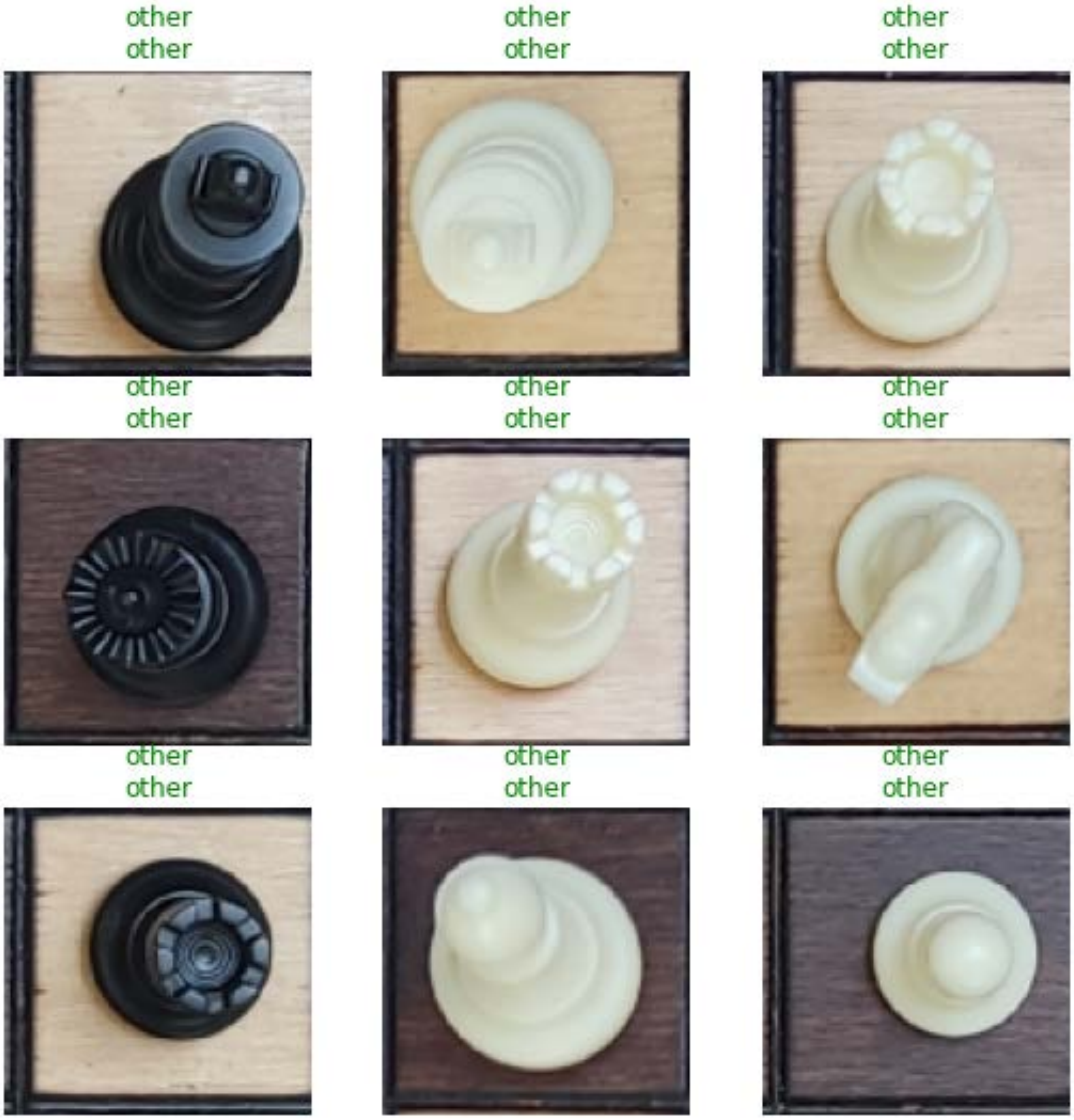

Fig. 4.36. Empty vs All prediction sample trained with 3 chess boards.

APPENDIX II: PROJECT BUDGET

This section contains a breakdown of all the expenses carried out for the development of this project. The expenses are divided intro two categories, human resources and equipment. While software is an usual category to classify expenses in, no money was spent in software as all the libraries and platforms used where free for non-commercial research. Let it be noted that the time my mentor devoted to this project is not being taken into consideration. Table 4.1 displays a breakdown of all project cost category wise.

Category	**Cost**
Equipment	134.81€
Human Resources	6325.02€
Total	6459.83€

TABLE 4.1. TOTAL COSTS

Equipment

Since the equipment used for this project has a longer lifespan, it is appropriate to compute its amortization cost rather than using its purchase prize. The amortization cost of a piece of equipment, knowing its usage period and life span can be computed following equation 4.1.

$$AmortizationCost = PurchasePrice * \frac{UsagePeriod}{LifeSpan} \tag{4.1}$$

Table 4.2 shows the individual expenses that are categorized as equipment costs.

Item	Purchase Price (m)	Usage Period (m)	Life Span	Amortization Cost
Laptop	599.99€	10	48	125.00€
Training board 1 [a]	29.99€	3	48	1.87€
Training board 2 [b]	9.99€	3	24	1.25€
Training board 3	79.99€	3	60	4.00€
Validation board	9.99€	3	24	1.25€
Tripod [c]	22.98€	3	48	1.44€
Total				134.81€

TABLE 4.2. EQUIPMENT COSTS

[a] https://www.amazon.es/dp/B07MXHL1ZZ
[b] https://www.amazon.es/gp/B085XTJQ5X
[c] https://www.amazon.es/gp/B09GM5NG9Z

Human Resources

To compute the cost of human resources, the amount of hours per week, the duration of the project in weeks and the latest hourly salary perceived by the author where used. The total amount is displayed in table 4.3

Hours per week	Duration of project (w)	Hourly rate	Cost
9	34	20,67€	6325.02€
Total			6325.02€

TABLE 4.3. HUMAN RESOURCES COSTS