Generating FEN Descriptions of Chess Boards by Using Two Important Pre-trained CNN Architectures

Cansu YANIK
Transportation, Security, Energy &
Automation Systems Business Sector
Software Engineering Department
ASELSAN Inc.
Ankara, TURKEY
cyanik@aselsan.com.tr

***Abstract*— In the article titled "Going deeper with convolutions" published in 2014, it was emphasized that the GoogleNet model is much better than AlexNet [1]. In this project, it is aimed to show with a real dataset that GoogleNet is a better model than AlexNet. The pre-trained models which are AlexNet and GoogleNet will be used and finetuning methods will be applied to these models. The models would be expected to generate the FEN description of an instant board taken from a chess game. It is also aimed to show how this situation changes with the using different hyper parameters (learning rate, optimizer and batch size). Algorithms will be analyzed and compared in terms of the evaluation metrics which are accuracy and cross-entropy loss. The strengths and weaknesses of these two pre-trained architectures will be discussed.**

Keywords—finetuning, pretrained models, hyperparameter optimization, FEN notation, chess board

I. INTRODUCTION

With the increasing interest in the field of computer vision, the work done in this field has increased. People have aimed to build models with better performance. The ImageNet [2] project has encouraged such studies. The ImageNet project is a large visual database designed for use in visual object recognition software research. Between 2010 and 2017, competitions (ILSVRC) were organized in the fields of object detection and classification and it was aimed to find the best algorithms. The winner of 2012, the AlexNet [3] model was an important architecture that raised interest in CNN architectures at that time. But the number of parameters and processing cost was high. GoogleNet, the winner of 2014, stated that they offered a deeper and more computationally low-cost algorithm compared to AlexNet [1].

Inspired by the increasing popularity of the game of chess, its extensive history, the interest in this game, and machine learning/artificial intelligence studies in the field of chess, it was decided to use a dataset and give a work in this field. In the paper "Going deeper with convolutions" published in 2014, it was emphasized that the GoogleNet model is much better than AlexNet [1]. Therefore, in this project, it is intended to show with a real dataset that GoogleNet is a better model than AlexNet. It is also aimed to show how this situation changes with different hyper parameters (learning rate, optimizer, and batch size). Thus, two different pre-trained models will be used in this project. These are AlexNet and GoogleNet. The accuracies (train, validation) of the algorithms are measured by using different hyper parameters. By analyzing all outputs and observations, performances of algorithms are commented and evaluated in terms of the efficiency.

In this project, it is planned to apply finetuning, hyper parameter optimization, learning rate, optimizer and batch size selection and evaluation.

In the rest of the document, related works about the project are given and then there is given more detailed project explanations with the information about dataset, methods used, experiments and the results of the experiments. Also, some evaluations and comments will be done. At the conclusion stage, A brief summary of what has been done throughout the entire project will be presented. Finally, a reference list is found at the end of the paper.

II. RELATED WORK

AlexNet [3] is an important study in 2012 that made convolutional neural network models and deep learning popular again. At that time, they presented a state of art architecture that gave very good results according to traditional machine learning and computer vision algorithms. AlexNet is an 8-layer structure (5: Conv - 3 FC). It has more filter and convolution layers per layer [3]. It consists of 11x11, 5x5, 3x3, convolutions, max-pooling, dropout, data augmentation, ReLU activation, SGD momentum. ReLU activations are added after each convolution fully connected layer.

It is a breaking point in the image classification problem by providing a sudden increase in classification. The network achieved a top-5 error of 15.3% compared to the second place top-5 error rate of 26.2% in ImageNet ILSVRC competition.

The winner of the ILSVRC 2014 contest was GoogLeNet (Inception V1) from Google [1]. The first 5 error rate is reached to 6.67%. They presented both a deep and wide architecture. The architecture consisted of inception modules. 1x1, 3x3, 5x5 convolutions and max pooling are used in the module. They used 1x1 filters called bottleneck to reduce the dimension and so that the number of parameters. Thus, the computational cost was lower. The architecture consists of 22 layers (27 layers with pooling layers). The first layers are simple convolution layers. Then it consists of 9 inception modules. Last layers are dropout and linear (softmax) classification layers.

These two important architectures in CNN algorithms have been used by those concerned in computer vision areas. Since these architectures are trained with high parameter numbers and by using strong hardware powers, they offer us ready and pre-trained models. Thus, we can use these ready-made models to solve different problems. This is called transfer learning. A comprehensive review on transfer learning is provided by Pan & Yang (2010) [4]. This article

shows how to implement a transfer learning solution for image classification problems. Transfer learning is a popular method of computer vision because it saves us time and allows us to create accurate models (Rawat & Wang 2017) [5]. While solving a different problem, we take advantage of previous learning and do not have to start from scratch. Thus, we use pre-trained models as transfer learning in computer vision areas. That's why AlexNet and GoogleNet are used as pre-trained models in this project. Since the first layers extract the general features, the low layers in the models will be frozen. Yosinski et al. (2014) stated this [6]: "if first-layer features are general and last-layer features are specific, then there must be a transition from general to specific somewhere in the network".

In the paper titled as "Evaluation of Pre-Trained Convolutional Neural Network Models for Object Recognition", AlexNet and GoogleNet pre-trained models were used for image detection problem. According to this paper, even though AlexNet and GoogLeNet yield similar accuracy which is 99.65%, GoogleNet achieved better results [7]. In my project, I will plan to use these models for image classification problem. On the other hand, according to the Rahmathunneesa and Muneer's work, AlexNet performed faster and gave better results in terms of performance metrics compared to the GoogleNet for the categorization of glioma grades (brain tumors) [8]. Also, they said that, GoogLeNet performs intermediate performance in terms of all metrics. In another work which is "Transfer learning with pre-trained deep convolutional neural networks for serous cell classification", GoogleNet gave better performance compared to AlexNet in classification problem [9]. In addition, to improve the classification performance, the number of the training samples was increased by using data augmentation techniques.

III. DATASET

The dataset used in the project belongs to the Chess Positions dataset put by Pavel Koryakin on the Kaggle website [10]. The whole dataset includes 100000 images of a randomly generated chess positions of 5-15 pieces. For this project, it is planned to use less amount of images that will be randomly selected for train, validation and test datasets. The features of the dataset are as follows [10].

- All images are 400 by 400 pixels.
- Training set: 80000 images
- Test set: 20000 images
- Pieces were generated with the following probability distribution:
    - 30% for Pawn
    - 20% for Bishop
    - 20% for Knight
    - 20% for Rook
    - 0% for Queen
- 2 Kings are guaranteed to be on the board.
- Labels are in a filename in Forsyth–Edwards Notation format, but with dashes instead of slashes.

An example image format in the dataset is given in Fig. 1.

Fig. 1. Three example images and labels from train dataset

A. *Forsyth–Edwards Notation (FEN)*

Forsyth - Edwards Notation (FEN) is a standard notation that describes a particular board position of a chess game with a text line using only the ASCII characters. The goal of FEN is to obtain all the necessary information to restart a game from a specific location. FEN was discovered by journalist David Forsyth and developed by Steven J. Edwards with the use of computers.

FEN notation consists of six fields as given in Fig. 2.

A FEN record contains six fields. The separator between fields is a space. The fields are:

1. Piece placement (from White's perspective). Each rank is described, *starting with rank 8* and ending with rank 1; within each rank, the contents of each square are described from file "a" through file "h". Following the Standard Algebraic Notation (SAN), each piece is identified by a single letter taken from the standard English names (pawn = "P", knight = "N", bishop = "B", rook = "R", queen = "Q" and king = "K"). White pieces are designated using upper-case letters ("PNBRQK") while black pieces use lowercase ("pnbrqk"). Empty squares are noted using digits 1 through 8 (the number of empty squares), and "/" separates ranks.
2. Active color. "w" means White moves next; "b" means Black moves next.
3. Castling availability. If neither side can castle, this is "-". Otherwise, this has one or more letters: "K" (White can castle kingside), "Q" (White can castle queenside), "k" (Black can castle kingside), and/or "q" (Black can castle queenside). A move that temporarily prevents castling does not negate this notation.
4. En passant target square in algebraic notation. If there's no en passant target square, this is "-". If a pawn has just made a two-square move, this is the position "behind" the pawn. This is recorded regardless of whether there is a pawn in position to make an en passant capture.[6]
5. Halfmove clock: This is the number of halfmoves since the last capture or pawn advance. The reason for this field is that the value is used in the fifty-move rule.
6. Fullmove number: The number of the full move. It starts at 1, and is incremented after Black's move.

Fig. 2. Explanations of the FEN Fields (Directly taken from [11])

As mentioned in Fig. 2, the notation starts from the white side's perspective. Each piece has a code. Each grid that is not a piece is counted and written as a digit. If there is a black piece in a grid, the code of that piece is written in lower case letter, and if there is a white piece, it is written in capital case-letters.

In this project, the 4th, 5th and 6th fields are not used in the FEN labels used. In addition, labels are in a filename in Forsyth–Edwards Notation format, but with dashes instead of slashes.

The FEN notation of a sample board is given in Fig. 3.

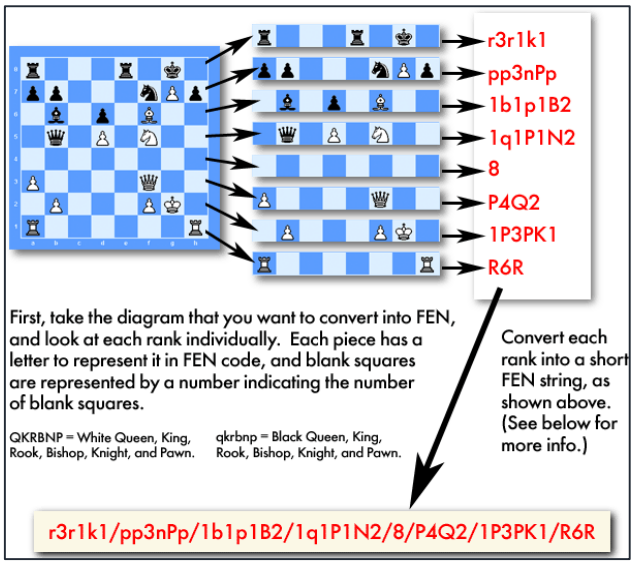

Fig. 3. An example FEN output of a chess board [12]

IV. METHOD

The steps and methods applied in this project consist of ten parts as roughly given below.

1) *Random image selection from dataset*
2) *Image Preprocessing*
3) *Creating Custom Dataset*
4) *Train and validation dataset separation*
5) *Model Finetuning*
6) *Parameters Selection*
7) *Training*
8) *Evaluation of Model using Test dataset*
9) *Doing some predictions*
10) *Comparison of Models*

First, a certain amount of random data was selected from the dataset. The dataset consisted of 80,000 trains and 20,000 test data. 1,000 board images were selected for train and validation, and 200 for testing. The amount of data may seem small, but in fact it is not. The reason for this will be explained later in the report.

In the image preprocessing part, the images are made suitable for giving to the models. First, the FEN notations, which are the names of each image, are converted to labels. Counting the twelve chess pieces and the empty grid, there are thirteen labels in total.

Pieces and their corresponding labels are shown in Fig. 4.

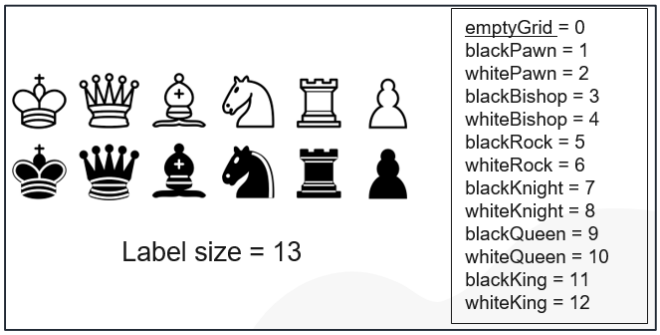

Fig. 4. Pieces and Labels

Then the images are converted to gray scale. The reason is this: There are many different colored boards and pieces in the dataset. Thus, by eliminating the color parameter, all images are transformed into a common format.

In this part, it is necessary to transform the problem into a classification task. For this reason, each board image is divided into grids. Thus, *64* images are obtained for each board. That's why the data set is actually not small. The size of the datasets used is as follows.

Chess Board Image Size for Training & Validation:

1000 Board Image

Chess Board Image Size for Test:

200 Board Image

Dataset for Training & Validation:

1000 x 64 = 64000 Image

Dataset for Test:

200 x 64 = 12800 Image

The pictures are resized to *(224, 224)* dimensions to fit the input size of the models, and they are converted into three channels because the models accept three channel images. A board image with preprocessing is given in Fig. 5.

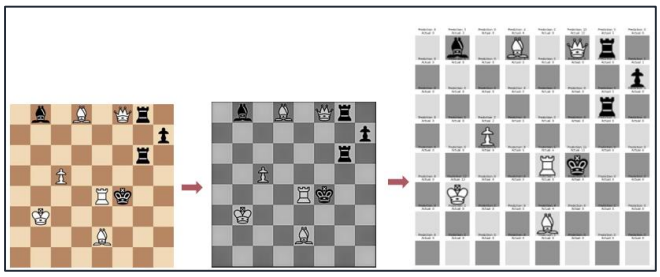

Fig. 5. Preprocessing on Board Image

In the next step, the dataset is randomly divided into train and validation data using a 0.2 split ratio.

In the next stage, AlexNet and GoogleNet models are created. Finetuning is applied to the models. The first layers of both models are frozen, their weight values are preserved, and the classifier layers are trained. The finetuning method used is shown in Fig. 6.

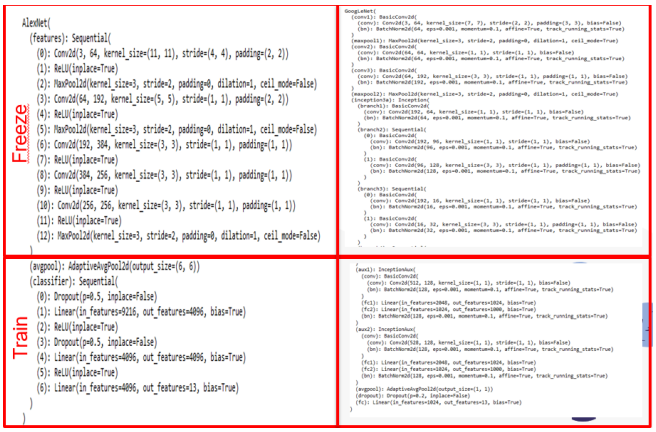

Fig. 6. Finetuning AlexNet and GoogleNet Models

After the models creation, the models are trained. Common parameters used in model training throughout the entire project are given in table 1.

TABLE I. COMMON PARAMETERS

Parameter Name	Value
Cross-Entropy Loss	-
Epoch Number	10
Momentum	0.9
Weight Decay	0.0000001

The following steps are carried out in the training part of the models.

For Each epoch
Set model to training mode for trainig, eval mode for validation
Iterate through all batches of images and labels
Take model output
Calculate the loss using CrossEntropyLoss
Empties the gradient tensors from previous batch in training mode
Perform back-propagation in training mode
Update the weight parameters in training mode
Calculate train and val loss
Calculate train and val accuracy

After model training, test data is used, model results are obtained and evaluated according to performance metrics. These steps remain the same and are implemented throughout the project. These steps are also applied in the use of different hyper parameters.

During the training of the models, the train and validation losses and accuracies are calculated in each epoch. In addition, the accuracy of the models is calculated using the test data after the model training.

A. *Cross-Entropy Loss*

Cross-entropy loss (log loss), measures the performance of a classification model whose output has a probability value between 0 and 1. The further the estimated probability is from the actual value, the greater the cross-entropy loss. In a perfect model, there would be 0 log loss. Cross-entropy loss is one of the most important and widely used loss functions for multi-category classification problems. For this reason, cross entropy loss is used in this project. Equation (1) is the loss formula used in binary classifications ($M = 2$). Equation (2) is used for multiple classifications ($M > 2$). For this reason, Equation (2) is used in this project.

$$-(ylog(p) + (1-y)\log(1-p)) \quad (1)$$

$$-\sum_{c=1}^{M} y_{o,c}\log(p_{o,c}) \quad (2)$$

M *- number of classes (dog, cat, fish)*

log *- the natural log*

y *- binary indicator (0 or 1) if class label cc is the correct classification for observation* ***o***

p *- predicted probability observation* ***o*** *is of class* ***c***

B. *Accuracy*

Accuracy is the ratio of predictions that a classification model is correct. Confusion Matrix table is given in Fig. 7 given below. This table shows the actual and predicted values in a classification problem. Since accuracy is easy to understand and use for binary and multiclass classification problem, in this project accuracy is used as a performance metric. How the accuracy value is calculated is given is Equation (3).

		Actual	
		Positive	Negative
Predicted	Positive	True Positive	False Positive
	Negative	False Negative	True Negative

Fig. 7. Confusion Matrix

$$(TP + TN) / (TP + FP + FN + TN) \quad (3)$$

C. *The System Used*

The system, processor and graphics processing unit used for training the models in this project are as follows:

Windows 10 Pro 64 Bit (10.0, Build 19042)
AMD Ryzen 5 3600 6-Core Processor
NVIDIA GeForce GTX 1660 SUPER 6 GB
16 GB 3200 MHz RAM
AMD Ryzen 5 3600 6-Core Processor 4.2 GHz

V. EXPERIMENT

A. *Experiment 1: Comparing Pretrained Models*

The project generally consists of two experiments. In the first experiment, it is aimed to show whether the GoogleNet model is a better model than AlexNet, as mentioned in the article "Going Deeper with Convolutions"[1] (which is one of the purpose of this project). Since the purpose of this

experiment is to compare models, the same hyper parameters are given to both models. The values in table 1 and table 2 are given to the models.

TABLE II. COMMON PARAMETERS

Parameter Name	Value
Learning Rate	0.001
Optimizer	Adam
Batch Size	64

The training of the models is completed and the results are obtained. The graphs of the train and validation loss, and train and validation accuracy are given in Fig. 8.

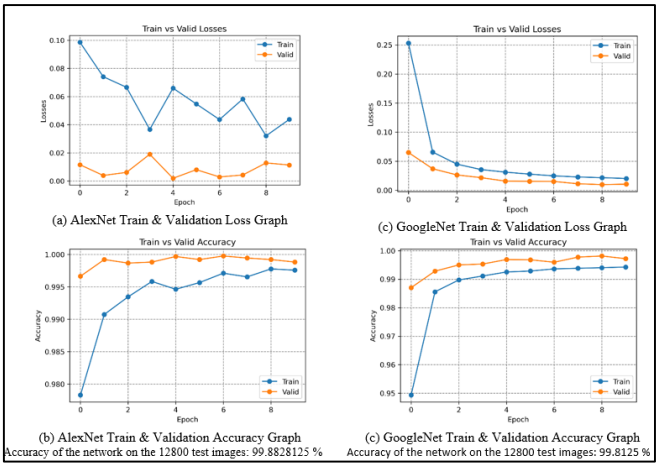

Fig. 8. Graphs of the Loss and Accuracy Values of the Models

It is easily seen that the curves of the GoogleNet model are more smooth and stable. But there is a gap between train and validation losses in AlexNet. This could be a problem for the performance of the model. Since validation loss is less than train loss, this can be underfitting. In both models, overfitting condition is not visible.

A little noise can be mentioned in the Alexnet model losses. Also, validation loss is much lower than train. This may indicate that the validation dataset is easier to predict than the training dataset. So there may be unrepresentative validation dataset problem here.

When these results are collected, inferences in table 3 can be made. According to table 3, the GoogleNet model is seen as a more stable and smooth model than AlexNet.

TABLE III. COMPARISON BETWEEN ALEXNET AND GOOGLENET

Inference /Models	*AlexNet*	*GoogleNet*
Better Accuracy	✓	✗
No GAP between Losses and Overfit	✗	✓
No Underfit	✗	✓
Validation dataset is easier to predict	✗	✓
Unrepresentative Validation Dataset	✗	✓
More parameters to learn	✗	✓

B. *Experiment 2: Applying Different HyperParameters*

The second experiment intended to be carried out in the project is the application of different hyperparameters to the models. The results of the models will be taken, the loss and accuracy curves will be analyzed and the most suitable configuration for both models has been determined. The hyper-parameters that are planned to be changed are the learning rate value, trying different optimizers and applying different batch sizes.

TABLE IV. DIFFERENT HYPERPARAMETERS

Parameters	**Values**		
Model	AlexNet	GoogleNet	
Learning Rate	0,1	0,01	0,001
Optimizer	Adam	SGD	
Batch Size	32	64	128

Model trainings are carried out as much as the combination of parameters given in table 4. In this case, the model trainings are done as much as the result of the *2*3*2*3* operation *(36 case)*. Two graphs are created for each model training and a total of seventy-two graphs are analyzed.

a) AlexNet Model Analyzing

Thirty-six different configuration results applied to the AlexNet model are given in the graphics in Fig. 9 and test accuracy values of each configuration, respectively, are given in Fig. 10. The hyper parameter values used are given as the titles of the plots.

Looking at the graphs in general, AlexNet has noisy curves for some configurations. For example, when the learning rate (LR) value is 0.01, while the optimizer (OPT) is Adam, there is noise in the accuracy curves regardless of the Batch sizes (BS). However, when LR is 0.001, OPT is Adam and BS is 64, this time loss curves have some noise. This may mean that the validation dataset is easier to predict than the training dataset, and there may be a non-representational validation dataset issue.

In addition, there are gaps between the accuracy curves in some configurations. For example, for LR: 0.1, OPT: Adam, BS: 32, a strong overfitting is seen in the model, while for LR: 0.1, OPT: Adam, BS: 128, there is a little overfitting.

In addition, since the train loss is slightly higher than the validation loss in the loss charts for LR: 0.001, OPT: Adam, BS: 64, and LR: 0.001, OPT: Adam, BS: 128, a small underfitting situation can be mentioned in these configurations.

Looking at both graphs and the test accuracy value, LR: 0.001, OPT: SGD, BS: 64 (configuration number is 15) can be selected as the configuration in which the AlexNet model will perform better. Because the graphics look much smoother and more stable than the others. Also, its accuracy value is one of the high values (which is 99.56 %).

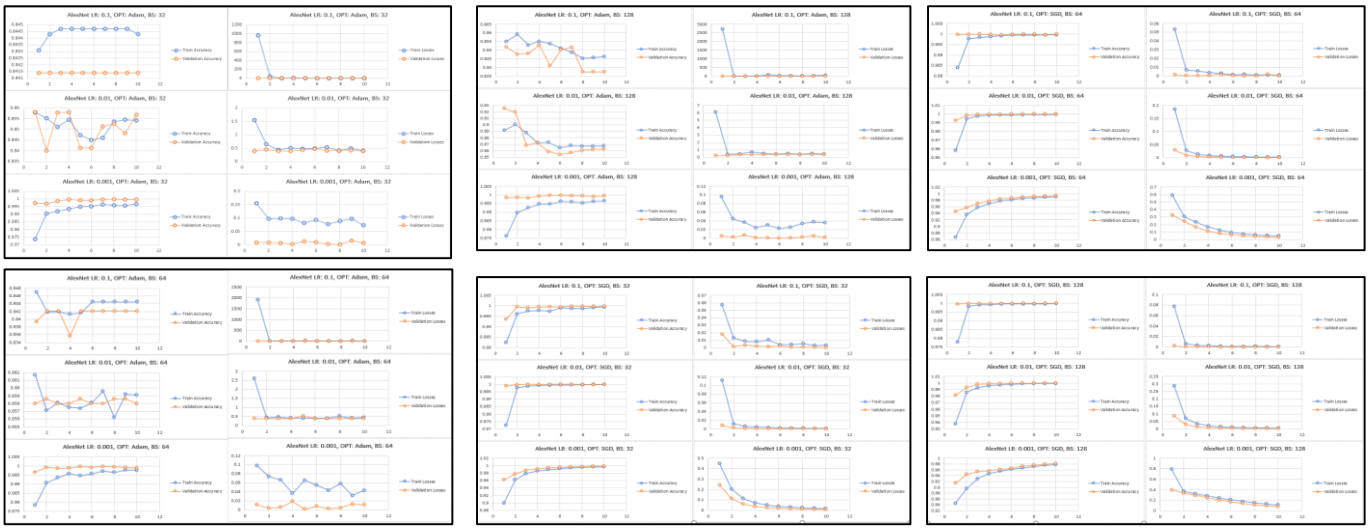

Fig 9. Plots of the Loss and Accuracy Values of the AlexNet Model for Different Hyperparameters

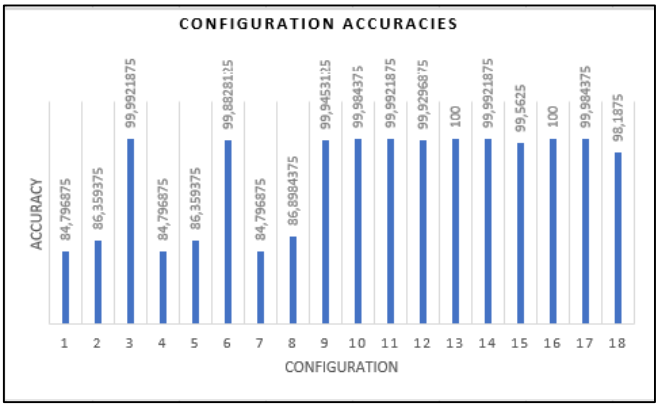

Fig. 10. AlexNet Configuration Accuracies

b) GoogleNet Model Analyzing

As in AlexNet, thirty-six different configuration results applied to the GoogleNet model given in the graphics in Fig. 11 and test accuracy values of each configuration, respectively, are given in Fig. 12. The hyper parameter values used are given as the titles of the plots.

When the graphs are examined in general, it is seen that the GoogleNet model generally gives smoother curves than AlexNet. However, for some configurations (e.g. LR: 0.1 OPT: Adam BS: 32 and LR: 0.1 OPT: Adam BS: 64) the curves appear to be noisy.

It seems that the model performs less at low learning rates and performs well when the optimizer is SGD.

There are no extreme gaps between the curves. But, For LR: 0.01, OPT: Adam, BS: 32; In LR: 0.1, OPT: Adam, BS: 64 and LR: 0.1, OPT: Adam, BS: 128 configurations, the validation loss seems less than the train loss and causes a gap. A minor underfitting situation can be mentioned here. However, when looking at graphs, it can be said that there is no overfitting situation.

Finally, when using the SGD optimizer, there is almost no noisy situation (at least very little) in the graphics. For this reason, it has been decided that SGD is the most suitable optimizer for GoogleNet.

Looking at both graphs and the test accuracy value, LR: 0.001, OPT: Adam, BS: 128 (configuration number is 9) can be selected as the configuration in which the GoogleNet model will perform better. Because the graphics look much smoother and more stable than the others. Also, its accuracy value is one of the high values (which is 99.69 %).

C. *Predictions*

At this stage, predictions are made to the models with the configurations selected for both models. Fig. 13 shows the images given to models and the label results. As seen in Fig. 13, three different types of images are given to the models.

When the prediction results are examined, the following inferences can be made. Models predict well when the board looks simple and there are few pieces on the board. But when there is a different piece style that is not in the dataset or has very few examples, the models tend to make the wrong guess. When it is more crowded in terms of pieces, there are cases where GoogleNet made a wrong guess while all of AlexNet's predictions are correct.

VI. Conclusion

In this project, two important CNN architectures, AlexNet and GoogleNet were used in the classification problem and these architectures were evaluated in terms of performance. The strengths and weaknesses of these two pre-trained architecture were discussed. Finetuning methods were applied to the models. The first layers were frozen, classifier layers were trained. In addition, the models were retrained using different hyper parameters (learning rate, optimizer and batch size) and the results were compared. Thus, it was observed whether there will be an improvement in the performance of the models if different parameters are used.

Acknowledgment

As the author, I would like to thank ASELSAN Company that I am working at; Istanbul Technical University (ITU) and dear teacher Hazım Kemal EKENEL.

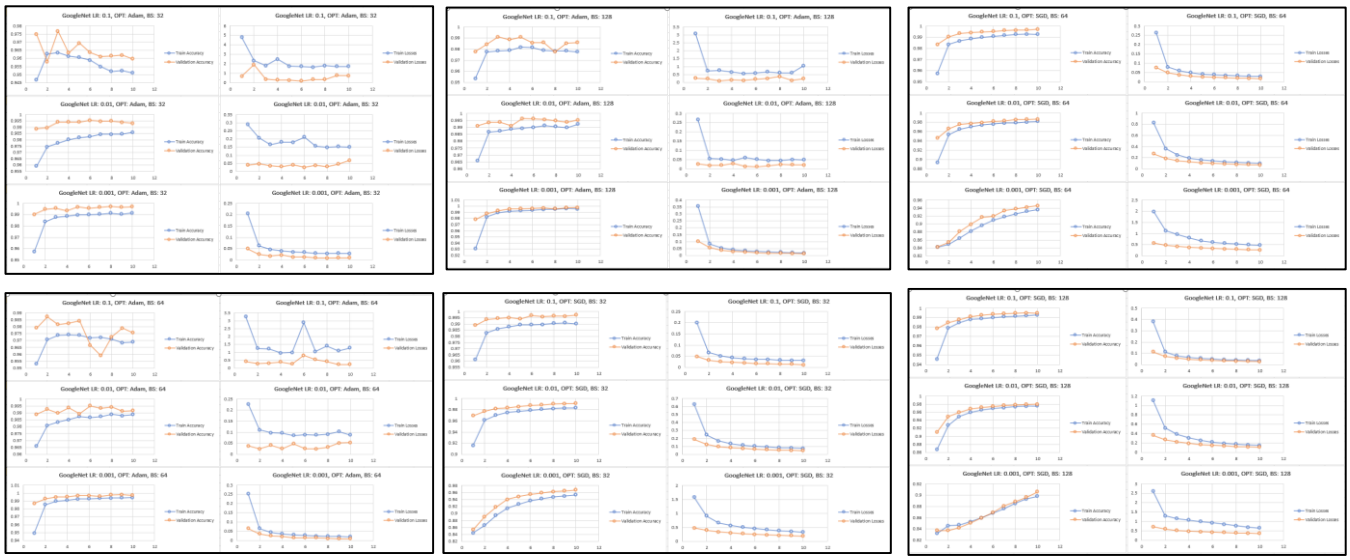

Fig 11. Plots of the Loss and Accuracy Values of the GoogleNet Model for Different Hyperparameters

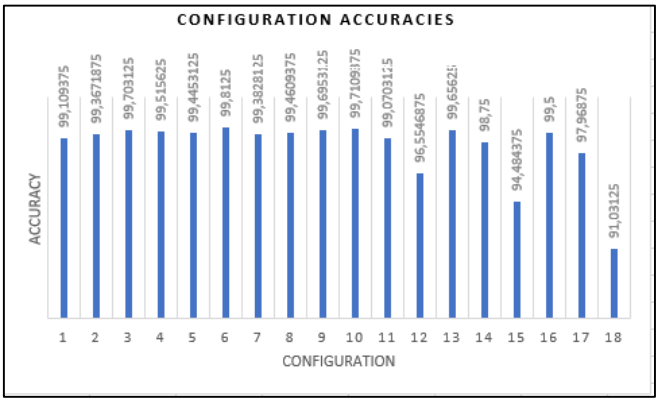

Fig. 12. GoogleNet Configuration Accuracies

REFERENCES

[1] C. Szegedy et al., "Going Deeper with Convolutions", arXiv.org, 2014. [Online]. Available: https://arxiv.org/abs/1409.4842.)

[2] [Online]. Available: http://www.image-net.org/ http://www.image-net.org/challenges/LSVRC/.

[3] Krizhevsky, Alex & Sutskever, Ilya & Hinton, Geoffrey. (2012). ImageNet Classification with Deep Convolutional Neural Networks. Neural Information Processing Systems. 25. 10.1145/3065386.

[4] Pan, S.J. and Yang, Q., 2010. A survey on transfer learning. IEEE Transactions on knowledge and data engineering, 22(10), pp.1345–1359.

[5] Rawat, W. and Wang, Z., 2017. Deep convolutional neural networks for image classification: A comprehensive review. Neural computation, 29(9), pp.2352–2449.

[6] Yosinski, J., Clune, J., Bengio, Y. and Lipson, H., 2014. How transferable are features in deep neural networks?. In Advances in neural information processing systems (pp. 3320–3328).

[7] Zabir, M. & Fazira, N. & Ibrahim, Zaidah & Sabri, Nurbaity. (2018). Evaluation of Pre-Trained Convolutional Neural Network Models for Object Recognition. International Journal of Engineering and Technology(UAE). 7. 95-98. 10.14419/ijet.v7i3.15.17509.

[8] A. P. Rahmathunneesa and K. V. Ahammed Muneer, "Performance Analysis of Pre-trained Deep Learning Networks for Brain Tumor Categorization," 2019 9th International Conference on Advances in Computing and Communication (ICACC), 2019, pp. 253-257, doi: 10.1109/ICACC48162.2019.8986151.

[9] Baykal, E., Dogan, H., Ercin, M.E. et al. Transfer learning with pre-trained deep convolutional neural networks for serous cell classification. Multimed Tools Appl 79, 15593–15611 (2020). https://doi.org/10.1007/s11042-019-07821-9.

[10] "Chess Positions", Kaggle.com. [Online]. Available: https://www.kaggle.com/koryakinp/chess-positions. [Accessed: 27-Apr- 2021].

[11] "Forsyth–Edwards Notation - Wikipedia", En.wikipedia.org. [Online]. Available: https://en.wikipedia.org/wiki/Forsyth%E2%80%93Edwards_Notation#cite_note-pgn_spec-1.

[12] "Forsyth-Edwards Notation F.A.Q.", Chessgames.com. [Online]. Available: https://www.chessgames.com/fenhelp.html.

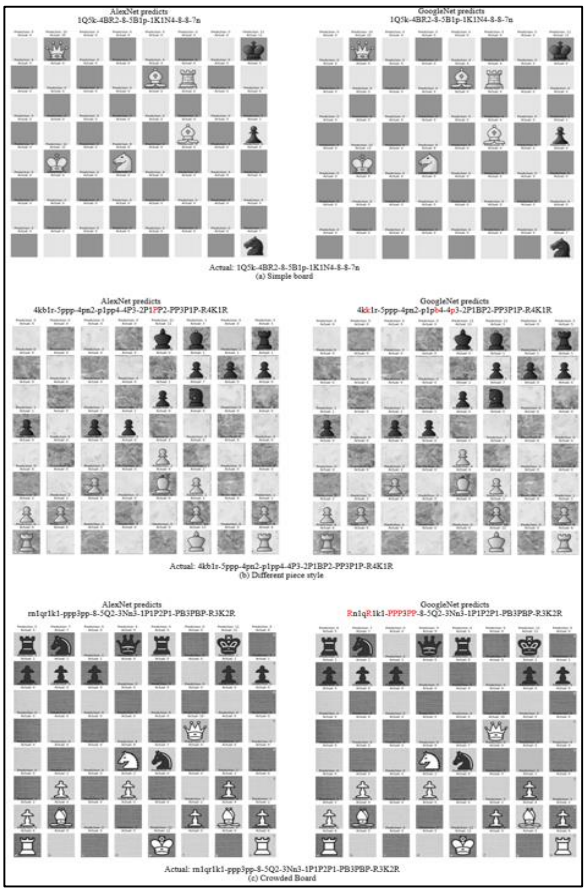

Fig. 13. Sample Predictions