BLG 506E – COMPUTER VISION

TERM PROJECT FINAL PRESENTATION

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

CANSU YANIK - 504201588

Table of contents

Problem Statement

Build Models

Building two different models able to learn how to generate FEN labels

+

GoogleNet better ?

Showing that GoogleNet is a better model than AlexNet by using a real dataset

+

Different Hyperparameters

Rebuilt models and showing how results change with different hyperparameters

Methodology

Applied Processes/Methods

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

Repeat these processes with different hyper parameters

Hyperparameters			
Model	Learning Rate	Optimizer	Batch Size
AlexNet	0.1	Adam	32
GoogleNet	0.01	SGD	64
	0.001		128

Methodology

	Model	Learning Rate	Optimizer	Batch Size
1. Case	AlexNet	0.1	Adam	64
2. Case	AlexNet	0.1	Adam	32
3. Case	AlexNet	0.1	Adam	128
4. Case	AlexNet	0.1	SGD	64
5. Case	AlexNet	0.1	SGD	32
6. Case	AlexNet	0.1	SGD	128
7. Case	AlexNet	0.01	Adam	64
8. Case	AlexNet	0.01	Adam	32
9. Case	AlexNet	0.01	Adam	128
10. Case	AlexNet	0.01	SGD	64
11. Case	AlexNet	0.01	SGD	32
12. Case	AlexNet	0.01	SGD	128
13. Case	AlexNet	0.001	Adam	64
14. Case	AlexNet	0.001	Adam	32
15. Case	AlexNet	0.001	Adam	128
16. Case	AlexNet	0.001	SGD	64
17. Case	AlexNet	0.001	SGD	32
18. Case	AlexNet	0.001	SGD	128
19. Case	GoogleNet	0.1	Adam	64
20. Case	GoogleNet	0.1	Adam	32
21. Case	GoogleNet	0.1	Adam	128
22. Case	GoogleNet	0.1	SGD	64
23. Case	GoogleNet	0.1	SGD	32
24. Case	GoogleNet	0.1	SGD	128
25. Case	GoogleNet	0.01	Adam	64
26. Case	GoogleNet	0.01	Adam	32
27. Case	GoogleNet	0.01	Adam	128
28. Case	GoogleNet	0.01	SGD	64
29. Case	GoogleNet	0.01	SGD	32
30. Case	GoogleNet	0.01	SGD	128
31. Case	GoogleNet	0.001	Adam	64
32. Case	GoogleNet	0.001	Adam	32
33. Case	GoogleNet	0.001	Adam	128
34. Case	GoogleNet	0.001	SGD	64
35. Case	GoogleNet	0.001	SGD	32
36. Case	GoogleNet	0.001	SGD	128

For each case, repeat

Applied Processes/Methods

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

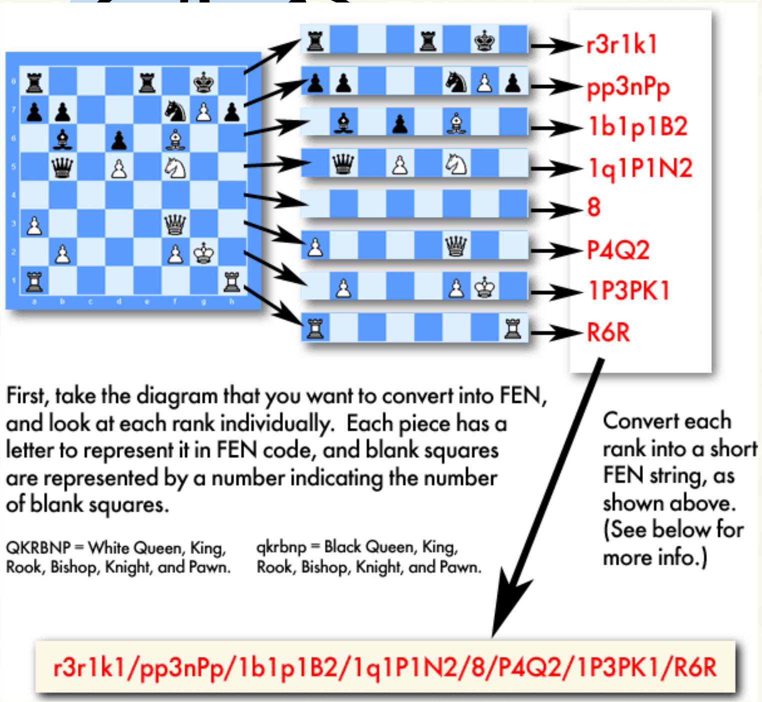
Used Datasets & Used Evaluation metrics

Chess Positions dataset put by Pavel Koryakin on the Kaggle website
https://www.kaggle.com/koryakinp/chess-positions.

Three exaple images and labels from train dataset

- 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.

Used Datasets & Used Evaluation metrics



Forsyth–Edwards Notation (FEN)

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

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.

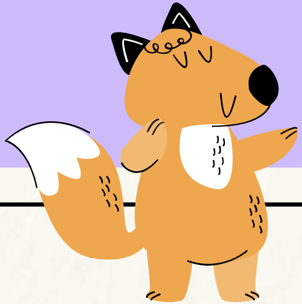
*Used Datasets & Used Evaluation metrics

1

Accuracy

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

```
Accuracy = (TP+TN)/(TP+FP+FN+TN)
```



2

Cross-Entropy Loss

In binary classification, where the number of classes M equals 2, cross-entropy can be calculated as:

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

If $M > 2$ (i.e. multiclass classification), we calculate a separate loss for each class label per observation and sum the result.

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

Note

- M - number of classes (dog, cat, fish)
- log - the natural log
- y - binary indicator (0 or 1) if class label c is the correct classification for observation o
- p - predicted probability observation o is of class c

https://ml-cheatsheet.readthedocs.io/en/latest/loss_functions.html

Experimental setup

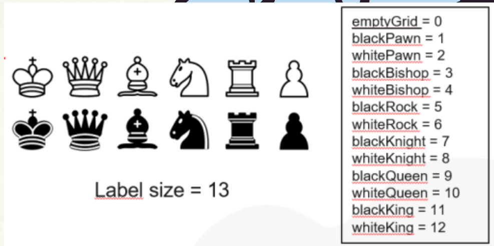
1) Random image selection from dataset
2) Image Preprocessing
3) Creating Custom Dataset
4) Train and validation dataset separation
5) Model Finetuning
6) Parameters Selection
7) Training
8) Evaluation of Model using Test dataset
9) Doing some predictions
10) Comparison of Models

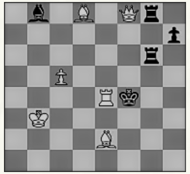
Chess Board Image Size for Training & Validation :
1000 Board Image
Chess Board Image Size for Test:
200 Board Image

Dataset for Training & Validation :
1000 x 64 = 64000 Image
Dataset for Test:
200 x 64 = 12800 Image

Image Preprocessing:
1. Taking FEN notations and converting FEN to label
2. Convert image to gray scale
3. Taking out grids of the board as 64 images
4. Resize Images to (224,224)
5. Convert gray Images to 3 channel Images

Train and Validation Split = 0.2
Train Dataset = 80%
Validation Dataset = 20%





Experimental setup

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

Freeze initial layers, train classifiers for both models!

Freeze

```
AlexNet(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(11, 11), stride=(4, 4), padding=(2, 2))
    (1): ReLU(inplace=True)
    (2): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
    (3): Conv2d(64, 192, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
    (4): ReLU(inplace=True)
    (5): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
    (6): Conv2d(192, 384, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (7): ReLU(inplace=True)
    (8): Conv2d(384, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (9): ReLU(inplace=True)
    (10): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (11): ReLU(inplace=True)
    (12): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
```

```
GoogLeNet(
  (conv1): BasicConv2d(
    (conv): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)
    (bn): BatchNorm2d(64, eps=0.001, momentum=0.1, affine=True, track_running_stats=True)
  )
  (maxpool1): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=True)
  (conv2): BasicConv2d(
    (conv): Conv2d(64, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn): BatchNorm2d(64, eps=0.001, momentum=0.1, affine=True, track_running_stats=True)
  )
  (conv3): BasicConv2d(
    (conv): Conv2d(64, 192, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (bn): BatchNorm2d(192, eps=0.001, momentum=0.1, affine=True, track_running_stats=True)
  )
  (maxpool2): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=True)
  (inception3a): Inception(
    (branch1): BasicConv2d(
      (conv): Conv2d(192, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn): BatchNorm2d(64, eps=0.001, momentum=0.1, affine=True, track_running_stats=True)
    )
    (branch2): Sequential(
      (0): BasicConv2d(
        (conv): Conv2d(192, 96, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn): BatchNorm2d(96, eps=0.001, momentum=0.1, affine=True, track_running_stats=True)
      )
      (1): BasicConv2d(
        (conv): Conv2d(96, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (bn): BatchNorm2d(128, eps=0.001, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
    (branch3): Sequential(
      (0): BasicConv2d(
        (conv): Conv2d(192, 16, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn): BatchNorm2d(16, eps=0.001, momentum=0.1, affine=True, track_running_stats=True)
      )
      (1): BasicConv2d(
        (conv): Conv2d(16, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (bn): BatchNorm2d(32, eps=0.001, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
```

Train

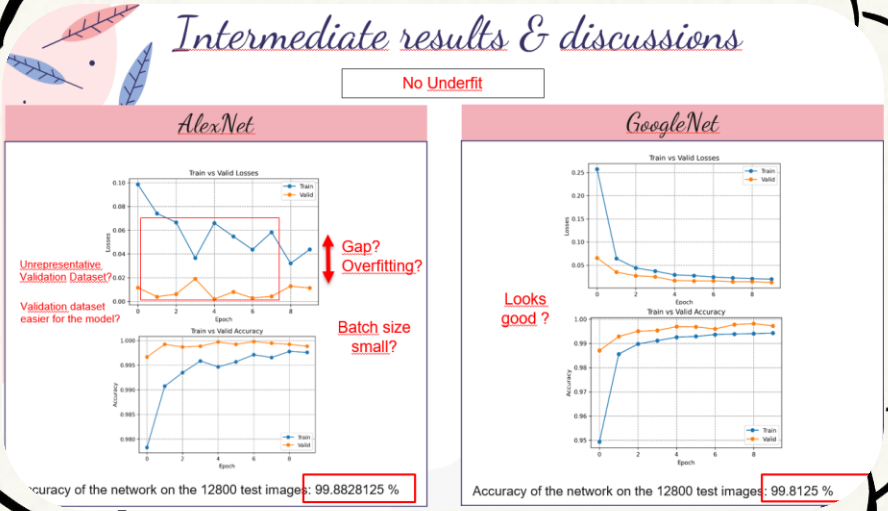
```
  (avgpool): AdaptiveAvgPool2d(output_size=(6, 6))
  (classifier): Sequential(
    (0): Dropout(p=0.5, inplace=False)
    (1): Linear(in_features=9216, out_features=4096, bias=True)
    (2): ReLU(inplace=True)
    (3): Dropout(p=0.5, inplace=False)
    (4): Linear(in_features=4096, out_features=4096, bias=True)
    (5): ReLU(inplace=True)
    (6): Linear(in_features=4096, out_features=13, bias=True)
  )
)
```

```
  (aux1): InceptionAux(
    (conv): BasicConv2d(
      (conv): Conv2d(512, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn): BatchNorm2d(128, eps=0.001, momentum=0.1, affine=True, track_running_stats=True)
    )
    (fc1): Linear(in_features=2048, out_features=1024, bias=True)
    (fc2): Linear(in_features=1024, out_features=1000, bias=True)
    (bn): BatchNorm2d(128, eps=0.001, momentum=0.1, affine=True, track_running_stats=True)
  )
  (aux2): InceptionAux(
    (conv): BasicConv2d(
      (conv): Conv2d(528, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn): BatchNorm2d(128, eps=0.001, momentum=0.1, affine=True, track_running_stats=True)
    )
    (fc1): Linear(in_features=2048, out_features=1024, bias=True)
    (fc2): Linear(in_features=1024, out_features=1000, bias=True)
    (bn): BatchNorm2d(128, eps=0.001, momentum=0.1, affine=True, track_running_stats=True)
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(1, 1))
  (dropout): Dropout(p=0.2, inplace=False)
  (fc): Linear(in_features=1024, out_features=13, bias=True)
)
```

- 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

AlexNet

- ✓ Better Accuracy than GoogleNet
- ✗ GAP between losses and Overfit
- ✓ No Underfit
- ✗ Validation dataset is easier to predict
- ✗ Unrepresentative Validation Dataset
- ✗ More parameters to learn



GoogleNet

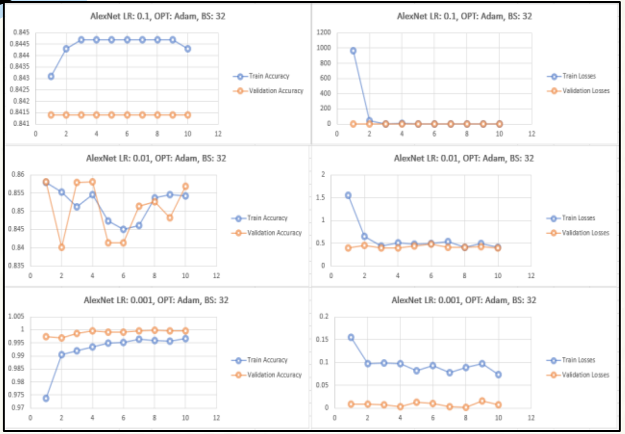
- ✗ AlexNet has better Accuracy
- ✓ No gap and Overfit
- ✓ No Underfit
- ✓ More smooth (no noisy movements)
- ✓ Curves looks good!
- ✓ Less parameters to learn

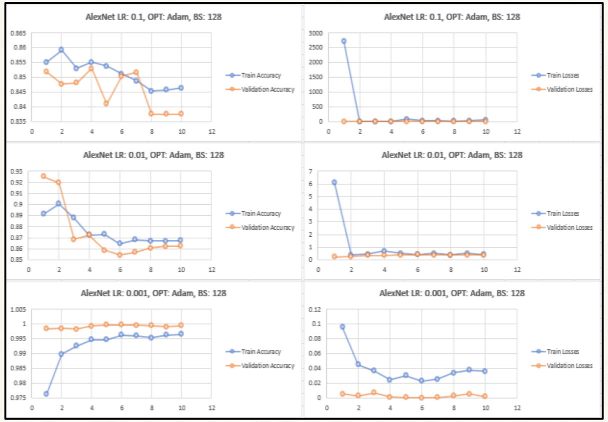
Parameter Name	Value
Learning Rate	0.001
Optimizer	Adam
Batch Size	64

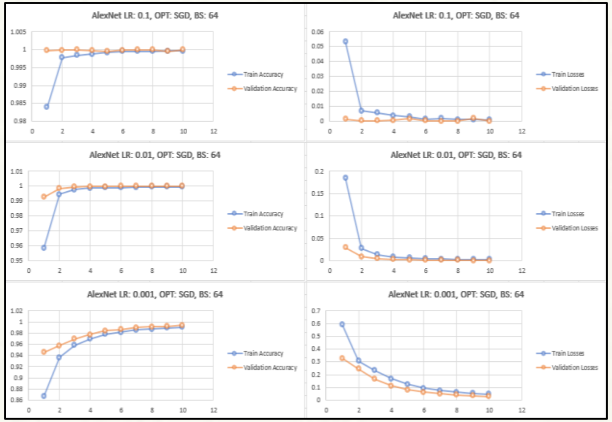
Parameter Name	Value
Cross-Entropy Loss	-
Epoch Number	10
Momentum	0.9
Weight Decay	0.0000001

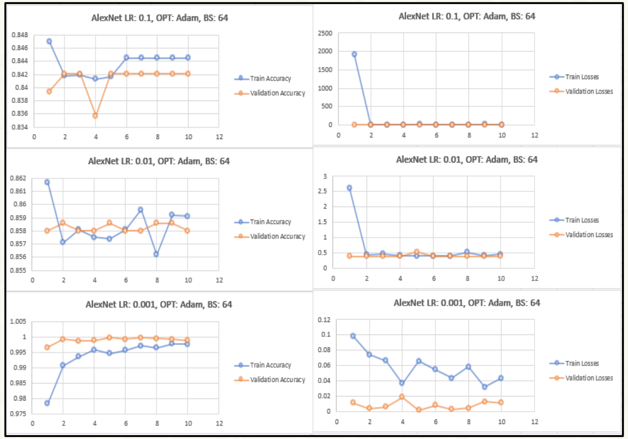
Results & discussions

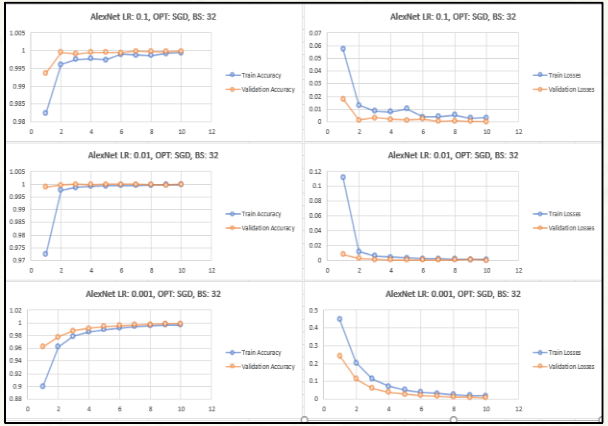
AlexNet

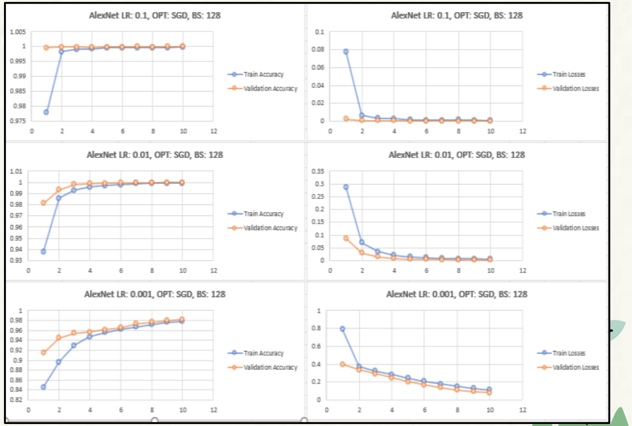






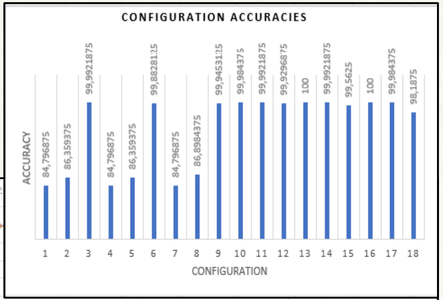


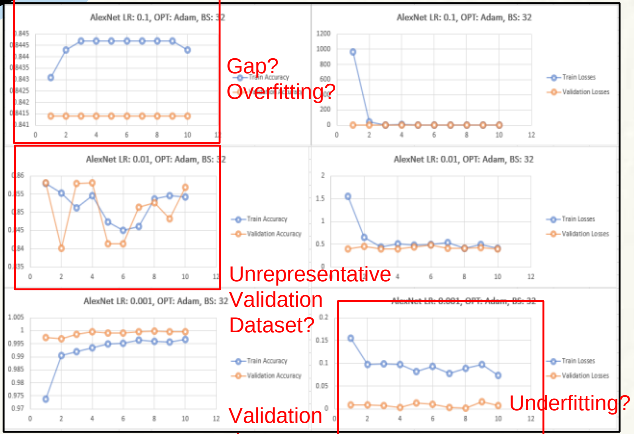




Results & discussions

AlexNet



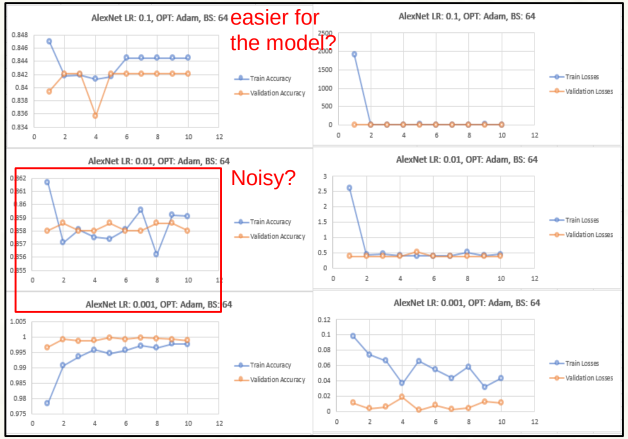


Gap?
Overfitting?

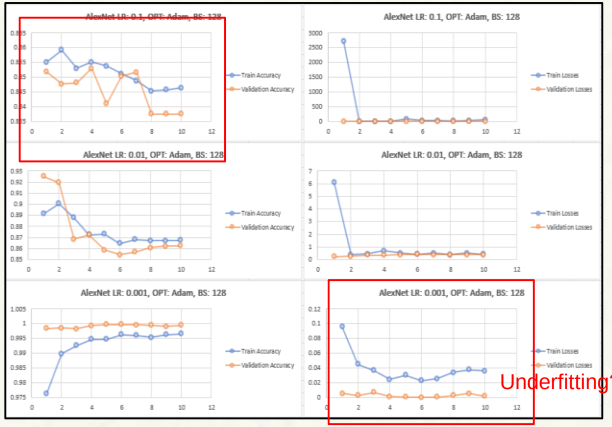
Unrepresentative Validation Dataset?

Validation dataset easier for the model?

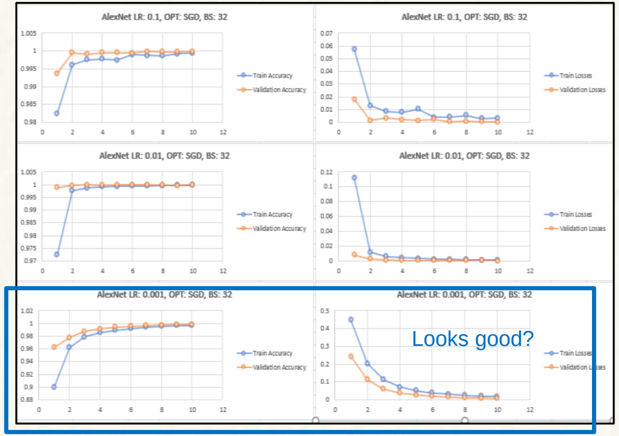
Underfitting?



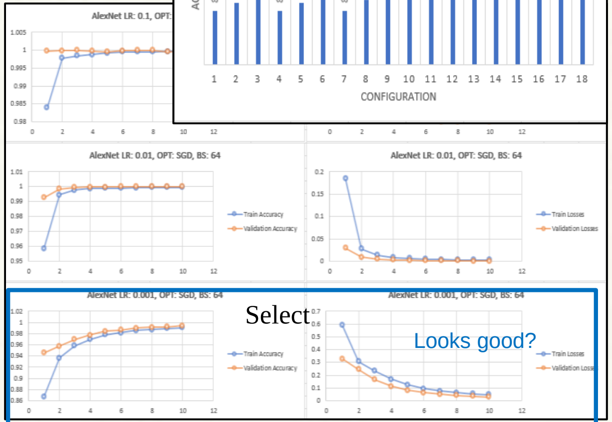
Noisy?



Underfitting?

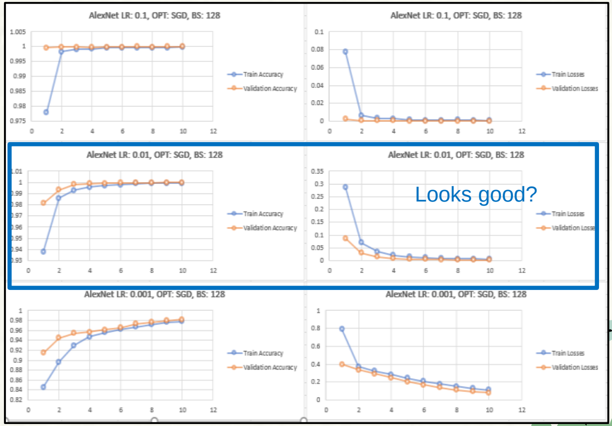


Looks good?



Select

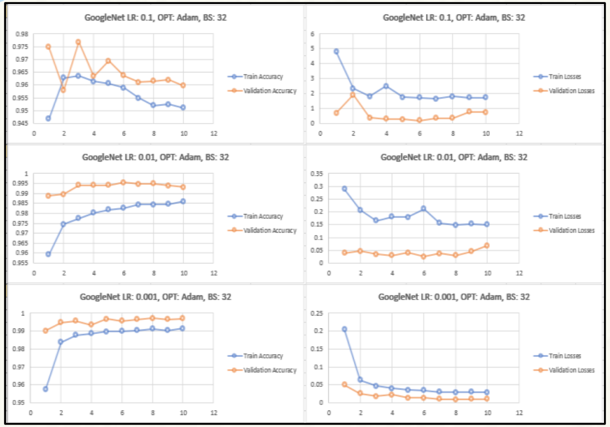
Looks good?

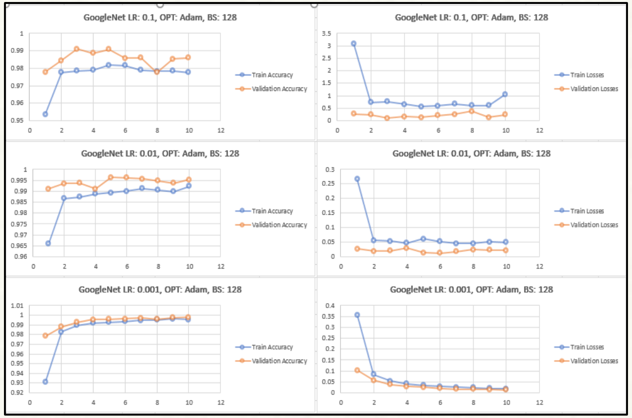


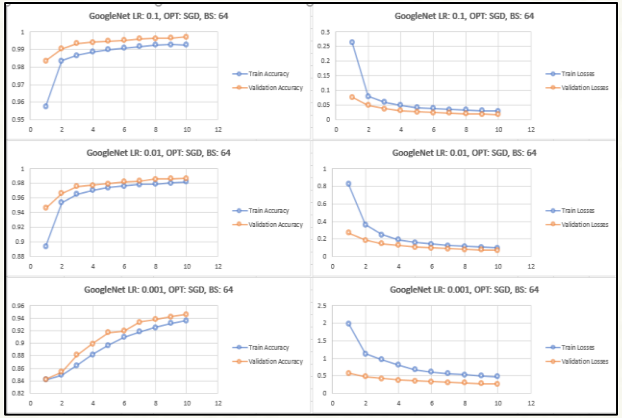
Looks good?

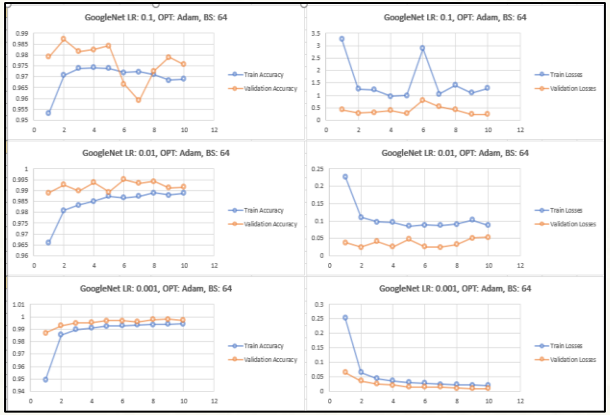
Results & discussions

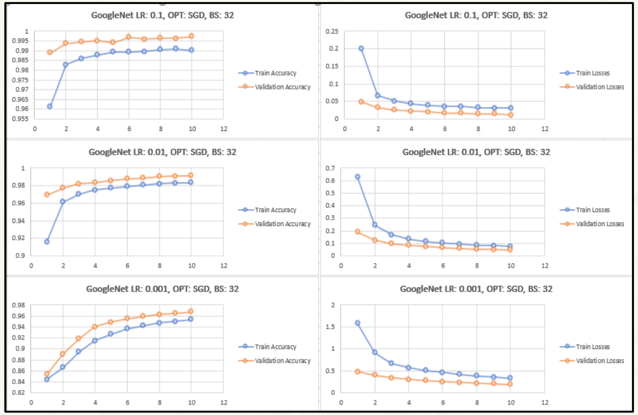
GoogleNet

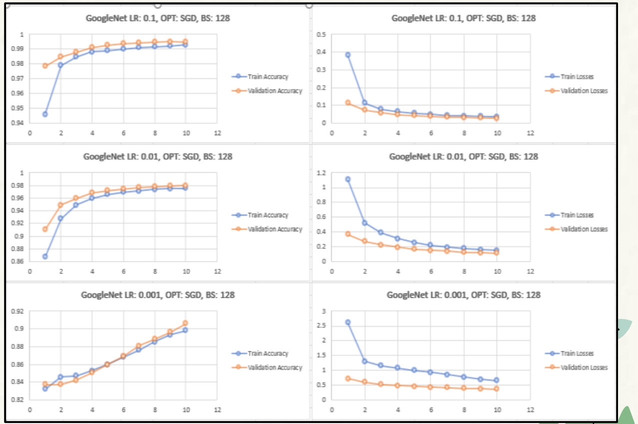




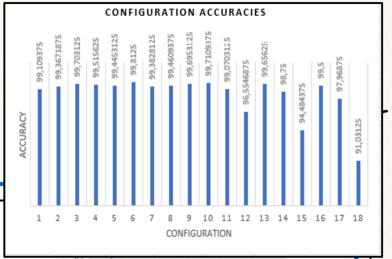








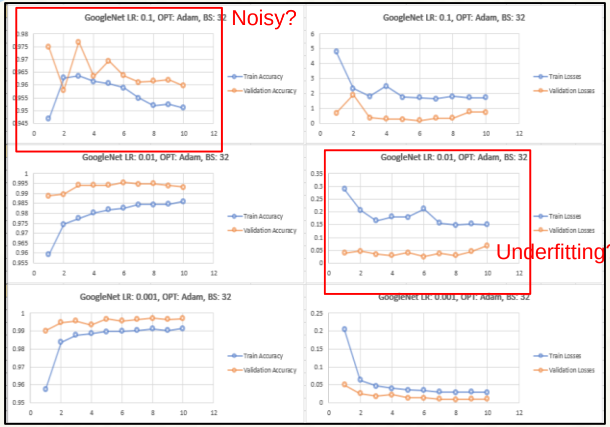
Results & discussions

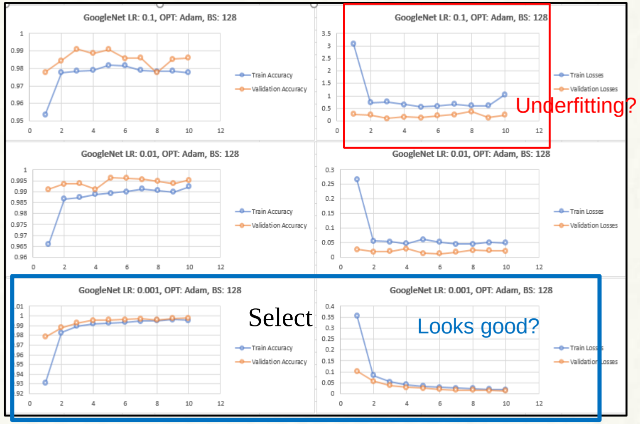


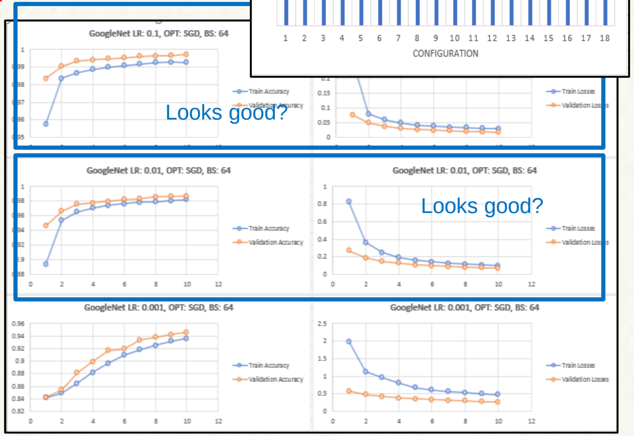
GoogleNet

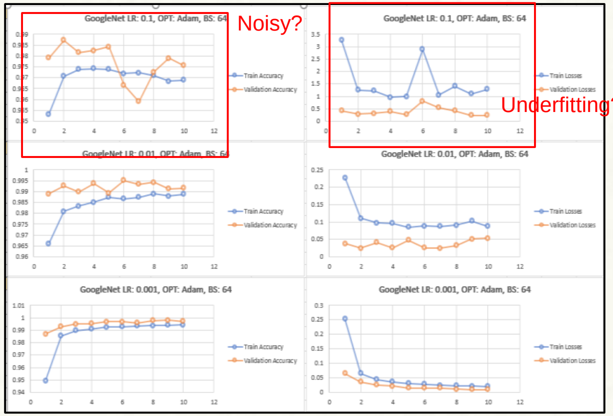
No overfitting?

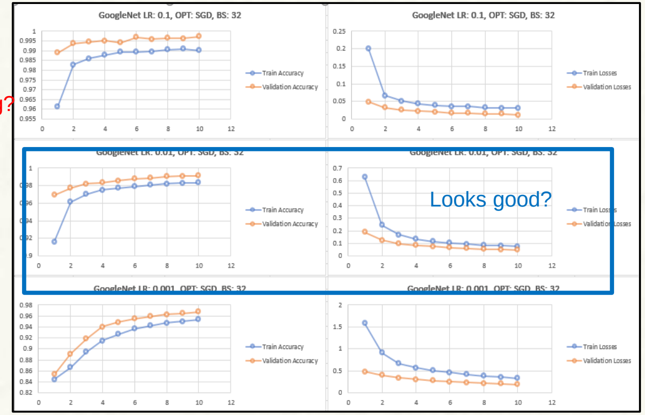
More smoother curves?

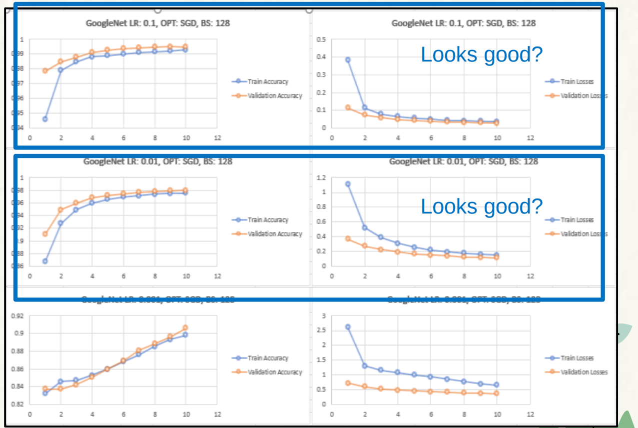






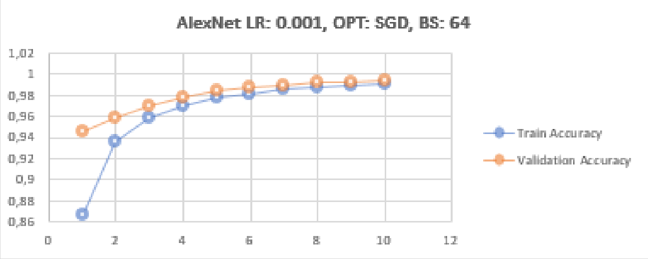


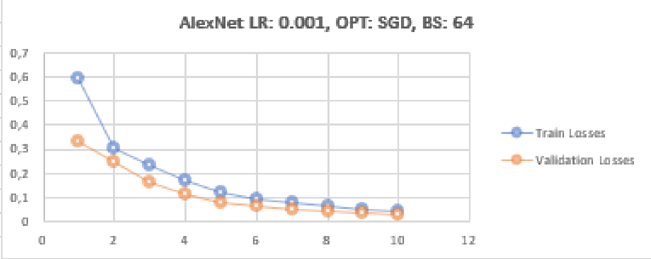




Selected Hyperparameters

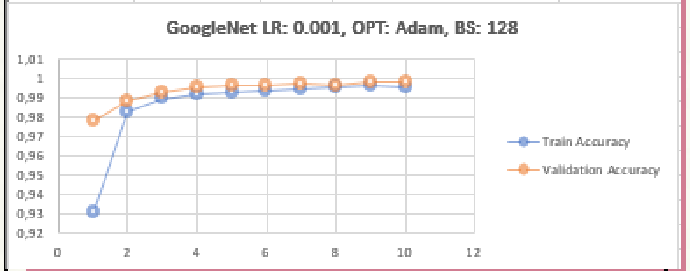
AlexNet

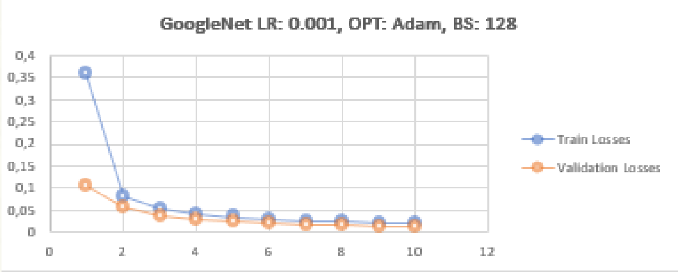




Accuracy of the model is 99,5625 %

GoogleNet





Accuracy of the model is 99,6953125 %

Some Predictions

```
fenToLabel_Dict = {'p' : blackPawn, 'P' : whitePawn,
                   'b' : blackBishop, 'B' : whiteBishop,
                   'r' : blackRock, 'R' : whiteRock,
                   'n' : blackKnight, 'N' : whiteKnight,
                   'q' : blackQueen, 'Q' : whiteQueen,
                   'k' : blackKing, 'K' : whiteKing,
                   '0' : emptyGrid}
```

```
labelToFen_Dict = {0:'0',
                   1:'p',2:'P',
                   3:'b',4:'B',
                   5:'r',6:'R',
                   7:'n',8:'N',
                   9:'q',10:'Q',
                   11:'k',12:'K'}
```

AlexNet predicts
1Q5k-4BR2-8-5B1p-1K1N4-8-8-7n

GoogleNet predicts
1Q5k-4BR2-8-5B1p-1K1N4-8-8-7n

Actual: 1Q5k-4BR2-8-5B1p-1K1N4-8-8-7n
(a) Simple board

Some Predictions

```
fenToLabel_Dict = {'p' : blackPawn, 'P' : whitePawn,
                   'b' : blackBishop, 'B' : whiteBishop,
                   'r' : blackRock, 'R' : whiteRock,
                   'n' : blackKnight, 'N' : whiteKnight,
                   'q' : blackQueen, 'Q' : whiteQueen,
                   'k' : blackKing, 'K' : whiteKing,
                   '0' : emptyGrid}
```

```
labelToFen_Dict = {0:'0',
                   1:'p',2:'P',
                   3:'b',4:'B',
                   5:'r',6:'R',
                   7:'n',8:'N',
                   9:'q',10:'Q',
                   11:'k',12:'K'}
```

AlexNet predicts

4kb1r-5ppp-4pn2-p1pp4-4P3-2P1PP2-PP3P1P-R4K1R

GoogleNet predicts

4kk1r-5ppp-4pn2-p1pb4-4p3-2P1BP2-PP3P1P-R4K1R

Actual: 4kb1r-5ppp-4pn2-p1pp4-4P3-2P1BP2-PP3P1P-R4K1R

(b) Different piece style

From progress presentation

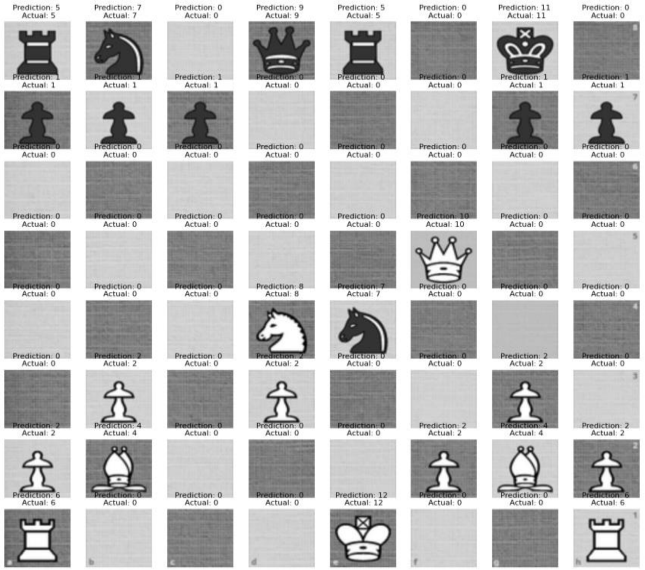
Some Predictions

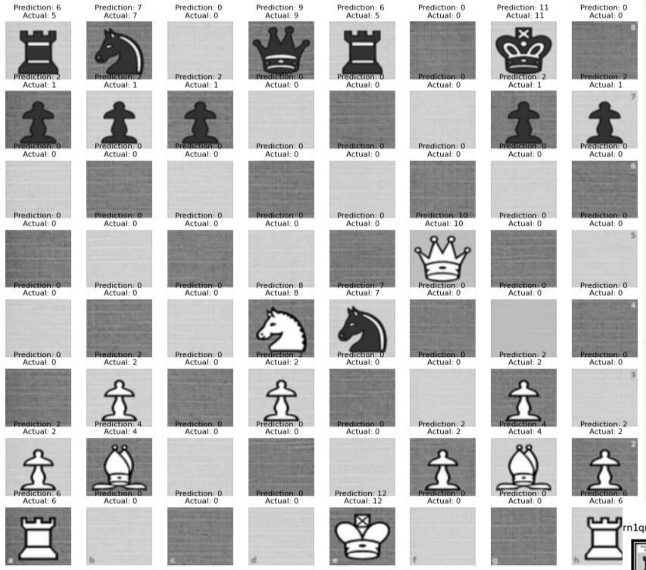
```
fenToLabel_Dict = {'p' : blackPawn, 'P' : whitePawn,
                   'b' : blackBishop, 'B' : whiteBishop,
                   'r' : blackRock, 'R' : whiteRock,
                   'n' : blackKnight, 'N' : whiteKnight,
                   'q' : blackQueen, 'Q' : whiteQueen,
                   'k' : blackKing, 'K' : whiteKing,
                   '0' : emptyGrid}
```

```
labelToFen_Dict = {0:'0',
                   1:'p',2:'P',
                   3:'b',4:'B',
                   5:'r',6:'R',
                   7:'n',8:'N',
                   9:'q',10:'Q',
                   11:'k',12:'K'}
```

AlexNet predicts
rn1qr1k1-ppp3pp-8-5Q2-3Nn3-1P1P2P1-PB3PBP-R3K2R

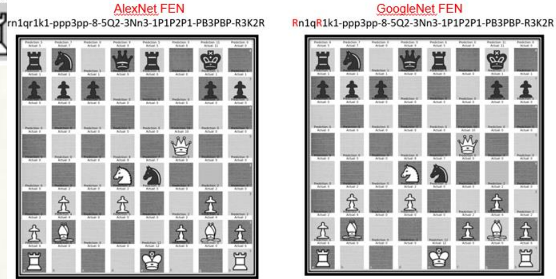
GoogleNet predicts
Rn1qR1k1-PPP3PP-8-5Q2-3Nn3-1P1P2P1-PB3PBP-R3K2R





Actual: rn1qr1k1-ppp3pp-8-5Q2-3Nn3-1P1P2P1-PB3PBP-R3K2R
(c) Crowded Board

From progress presentation



Conclusion

- One goal is to build two different models able to learn how to generate FEN labels.
- Second goal is to show that as it is said in “Going Deeper with Convolutions” paper, GoogleNet has a better performance than AlexNet by using a real dataset.
- It is also aimed to show how this situation changes with different hyper parameters (learning rate, optimizer and batch size).
- Finetuning methods were used. Initial layers were freezed and classifier layers were trained.
- Both models gives high accuracies.
- After analyzing results and performans metrics, It is shown that GoogleNet model looks more stable and suitable for the problem.
- AlexNet predicts better.
- Models may predict false when chess pieces are so different than dataset.

Thank you
for listening!