



BLG 506E – COMPUTER VISION



Project Proposal PRESENTATION



CANSU YANIK - 504201588



Table of Contents

01

Motivation

02

About the Project

03

Project Goals and
Impacts

04

Project Schedule

05

Conclusion



Motivation

C. Szegedy et al., "Going Deeper with Convolutions", 2014. Available: <https://arxiv.org/abs/1409.4842>.

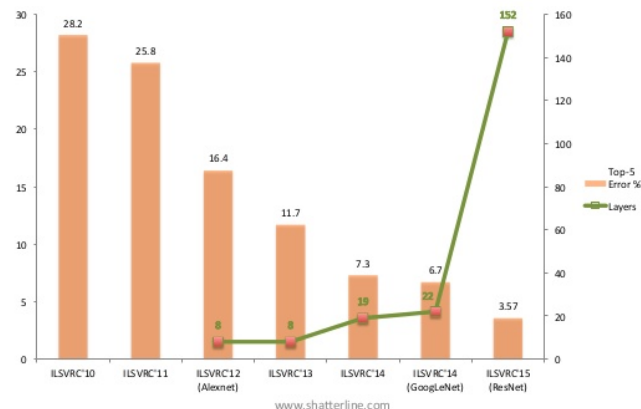
In the last three years, mainly due to the advances of deep learning, more concretely convolutional networks [10], the quality of image recognition and object detection has been progressing at a dramatic pace. One encouraging news is that most of this progress is not just the result of more powerful hardware, larger datasets and bigger models, but mainly a consequence of new ideas, algorithms and improved network architectures. No new data sources were used, for example, by the top entries in the ILSVRC 2014 competition besides the classification dataset of the same competition for detection purposes. Our GoogLeNet submission to ILSVRC 2014 actually uses $12\times$ fewer parameters than the winning architecture of Krizhevsky et al [9] from two years ago, while being significantly more accurate. The biggest gains in object-detection have not come from the utilization of deep networks alone or bigger models, but from the synergy of deep architectures and classical computer vision, like the R-CNN algorithm by Girshick et al [6].

Another notable factor is that with the ongoing traction of mobile and embedded computing, the efficiency of our algorithms – especially their power and memory use – gains importance. It is noteworthy that the considerations leading to the design of the deep architecture presented in this paper included this factor rather than having a sheer fixation on accuracy numbers. For most of the experiments, the models were designed to keep a computational budget of 1.5 billion multiply-adds at inference time, so that the they do not end up to be a purely academic curiosity, but could be put to real world use, even on large datasets, at a reasonable cost.

GoogleNet vs AlexNet

- 12x fewer parameters
- More accurate
- Lower memory use
- Much bigger network than the AlexNet

Year 2012 Marked The Inflection Point Reintroducing CNNs Led to Big Drop in Error for Image Classification. Since Then, Deeper Networks Continued to Reduce Error



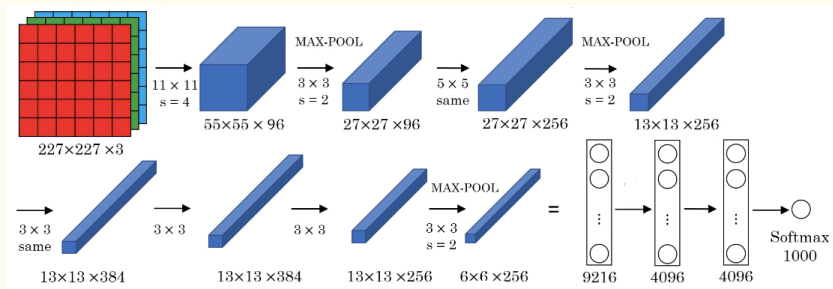


About the Project

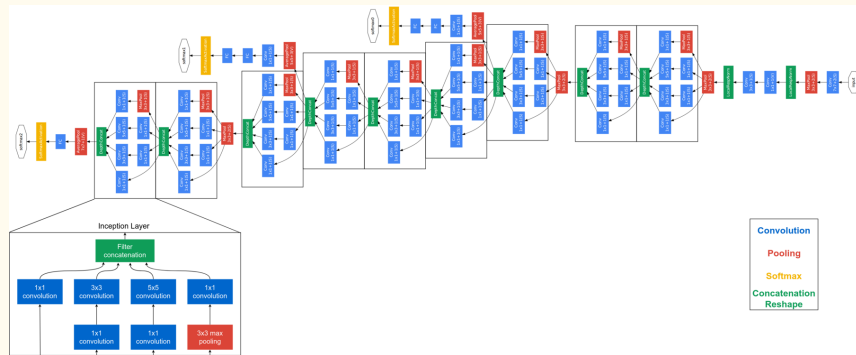
Generating FEN Descriptions of Chess Boards by Using Two Important Pre-trained CNN Architectures

Aimed => showing with a real dataset that GoogleNet is a better model than AlexNet.

AlexNet



GoogleNet



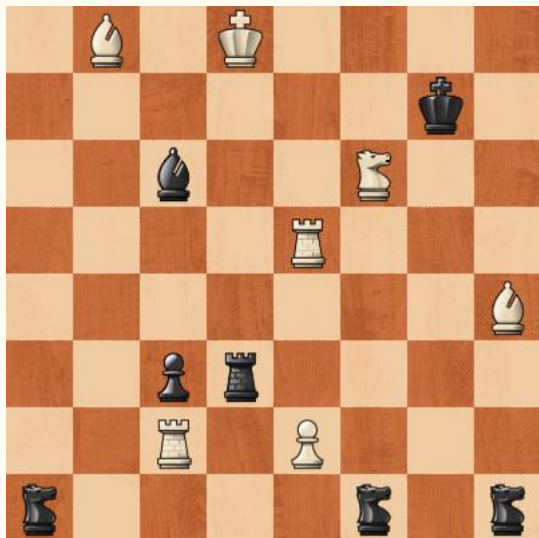
Generating FEN Descriptions of Chess Boards by Using Two Important Pre-trained CNN Architectures

DATASET

Chess Positions dataset put by Pavel Koryakin on the Kaggle website

<https://www.kaggle.com/koryakin/chess-positions>.

A sample of dataset and its label

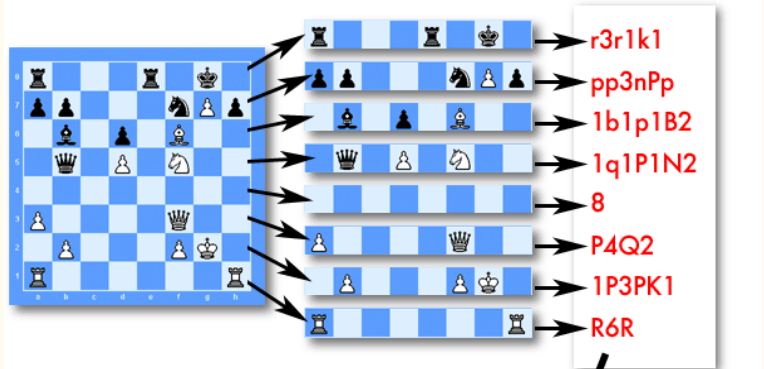


1B1K4-6k1-2b2N2-4R3-7B-2pr4-2R1P3-n4n1n.jpeg

- 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.

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

Forsyth-Edwards Notation (FEN)



First, take the diagram that you want to convert into FEN, and look at each rank individually. Each piece has a letter to represent it in FEN code, and blank squares are represented by a number indicating the number of blank squares.

QKRBNP = White Queen, King, Rook, Bishop, Knight, and Pawn. qkrbnp = Black Queen, King, Rook, Bishop, Knight, and Pawn.

Convert each rank into a short FEN string, as shown above. (See below for more info.)

r3r1k1/pp3nPp/1b1p1B2/1q1P1N2/8/P4Q2/1P3PK1/R6R

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.

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

FINETUNING

Layer (type)	Output Shape	Param #
Conv2d-1	[-1, 64, 55, 55]	23,296
ReLU-2	[-1, 64, 55, 55]	0
MaxPool2d-3	[-1, 64, 27, 27]	0
Conv2d-4	[-1, 192, 27, 27]	307,392
ReLU-5	[-1, 192, 27, 27]	0
MaxPool2d-6	[-1, 192, 13, 13]	0
Conv2d-7	[-1, 384, 13, 13]	663,936
ReLU-8	[-1, 384, 13, 13]	0
Conv2d-9	[-1, 256, 13, 13]	884,992
ReLU-10	[-1, 256, 13, 13]	0
Conv2d-11	[-1, 256, 13, 13]	590,080
ReLU-12	[-1, 256, 13, 13]	0
MaxPool2d-13	[-1, 256, 6, 6]	0
AdaptiveAvgPool2d-14	[-1, 256, 6, 6]	0
Dropout-15	[-1, 9216]	0
Linear-16	[-1, 4096]	37,752,832
ReLU-17	[-1, 4096]	0
Dropout-18	[-1, 4096]	0
Linear-19	[-1, 4096]	16,781,312
ReLU-20	[-1, 4096]	0
Linear-21	[-1, 1000]	4,097,000

Freeze these layers

Train these layers

Total params: 61,100,840
 Trainable params: 61,100,840
 Non-trainable params: 0

Input size (MB): 0.57
 Forward/backward pass size (MB): 8.38
 Params size (MB): 233.08
 Estimated Total Size (MB): 242.03

Finetuning AlexNet Model

Layer (type)	Output Shape	Param #
Conv2d-1	[-1, 64, 112, 112]	9,408
BatchNorm2d-2	[-1, 64, 112, 112]	128
BasicConv2d-3	[-1, 64, 112, 112]	0
MaxPool2d-4	[-1, 64, 56, 56]	0
Conv2d-5	[-1, 64, 56, 56]	4,096
BatchNorm2d-6	[-1, 64, 56, 56]	128
BasicConv2d-7	[-1, 64, 56, 56]	0
Conv2d-8	[-1, 192, 56, 56]	110,592
BatchNorm2d-9	[-1, 192, 56, 56]	304
BasicConv2d-10	[-1, 192, 56, 56]	0
MaxPool2d-11	[-1, 192, 28, 28]	0
Conv2d-12	[-1, 64, 28, 28]	12,208
BatchNorm2d-13	[-1, 64, 28, 28]	128
BasicConv2d-14	[-1, 64, 28, 28]	0
Conv2d-15	[-1, 96, 28, 28]	18,432
BatchNorm2d-16	[-1, 96, 28, 28]	192
BasicConv2d-17	[-1, 96, 28, 28]	0
Conv2d-18	[-1, 128, 28, 28]	110,592
BatchNorm2d-19	[-1, 128, 28, 28]	256
BasicConv2d-20	[-1, 128, 28, 28]	0
Conv2d-21	[-1, 16, 28, 28]	3,072
BatchNorm2d-22	[-1, 16, 28, 28]	32
BasicConv2d-23	[-1, 16, 28, 28]	0
Conv2d-24	[-1, 32, 28, 28]	4,608
BatchNorm2d-25	[-1, 32, 28, 28]	64
BasicConv2d-26	[-1, 32, 28, 28]	0
MaxPool2d-27	[-1, 32, 28, 28]	0
Conv2d-28	[-1, 32, 28, 28]	6,144
BatchNorm2d-29	[-1, 32, 28, 28]	64
BasicConv2d-30	[-1, 32, 28, 28]	0
Inception-31	[-1, 256, 28, 28]	0
Conv2d-32	[-1, 128, 28, 28]	32,768
BatchNorm2d-33	[-1, 128, 28, 28]	256
BasicConv2d-34	[-1, 128, 28, 28]	0
Conv2d-35	[-1, 128, 28, 28]	32,768
BatchNorm2d-36	[-1, 128, 28, 28]	256
BasicConv2d-37	[-1, 128, 28, 28]	0
Conv2d-38	[-1, 192, 28, 28]	221,184
BatchNorm2d-39	[-1, 192, 28, 28]	384
BasicConv2d-40	[-1, 192, 28, 28]	0
... More Layers		
Inception-171	[-1, 832, 7, 7]	0
Conv2d-174	[-1, 384, 7, 7]	319,488
BatchNorm2d-175	[-1, 384, 7, 7]	768
BasicConv2d-176	[-1, 384, 7, 7]	0
Conv2d-177	[-1, 192, 7, 7]	159,744
BatchNorm2d-178	[-1, 192, 7, 7]	384
BasicConv2d-179	[-1, 192, 7, 7]	0
Conv2d-180	[-1, 384, 7, 7]	663,552
BatchNorm2d-181	[-1, 384, 7, 7]	768
BasicConv2d-182	[-1, 384, 7, 7]	0
Conv2d-183	[-1, 48, 7, 7]	39,936
BatchNorm2d-184	[-1, 48, 7, 7]	96
BasicConv2d-185	[-1, 48, 7, 7]	0
Conv2d-186	[-1, 128, 7, 7]	55,296
BatchNorm2d-187	[-1, 128, 7, 7]	256
BasicConv2d-188	[-1, 128, 7, 7]	0
MaxPool2d-189	[-1, 832, 7, 7]	0
Conv2d-190	[-1, 128, 7, 7]	106,496
BatchNorm2d-191	[-1, 128, 7, 7]	256
BasicConv2d-192	[-1, 128, 7, 7]	0
Inception-193	[-1, 1024, 7, 7]	0
AdaptiveAvgPool2d-194	[-1, 1024, 1, 1]	0
Dropout-195	[-1, 1024]	0
Linear-196	[-1, 1000]	1,025,000

Freeze these layers

Train these layers

... More Layers

Total params: 6,624,904
 Trainable params: 6,624,904
 Non-trainable params: 0

Input size (MB): 0.57
 Forward/backward pass size (MB): 94.11
 Params size (MB): 25.27
 Estimated Total Size (MB): 119.95

Finetuning GoogleNet Model

Project Goals and Impacts



Build Models



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





GoogleNet is better

Showing GoogleNet has a better performance than AlexNet

Different Hyperparameters



Showing how results change with different hyper parameters (eg learning rate, regularization, dropout etc.)



Project Goals and Impacts

Build Models



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



GoogleNet is better

Showing GoogleNet has a better performance than AlexNet

Different Hyperparameters



Showing how results change with different hyper parameters (eg learning rate, regularization, dropout etc.)

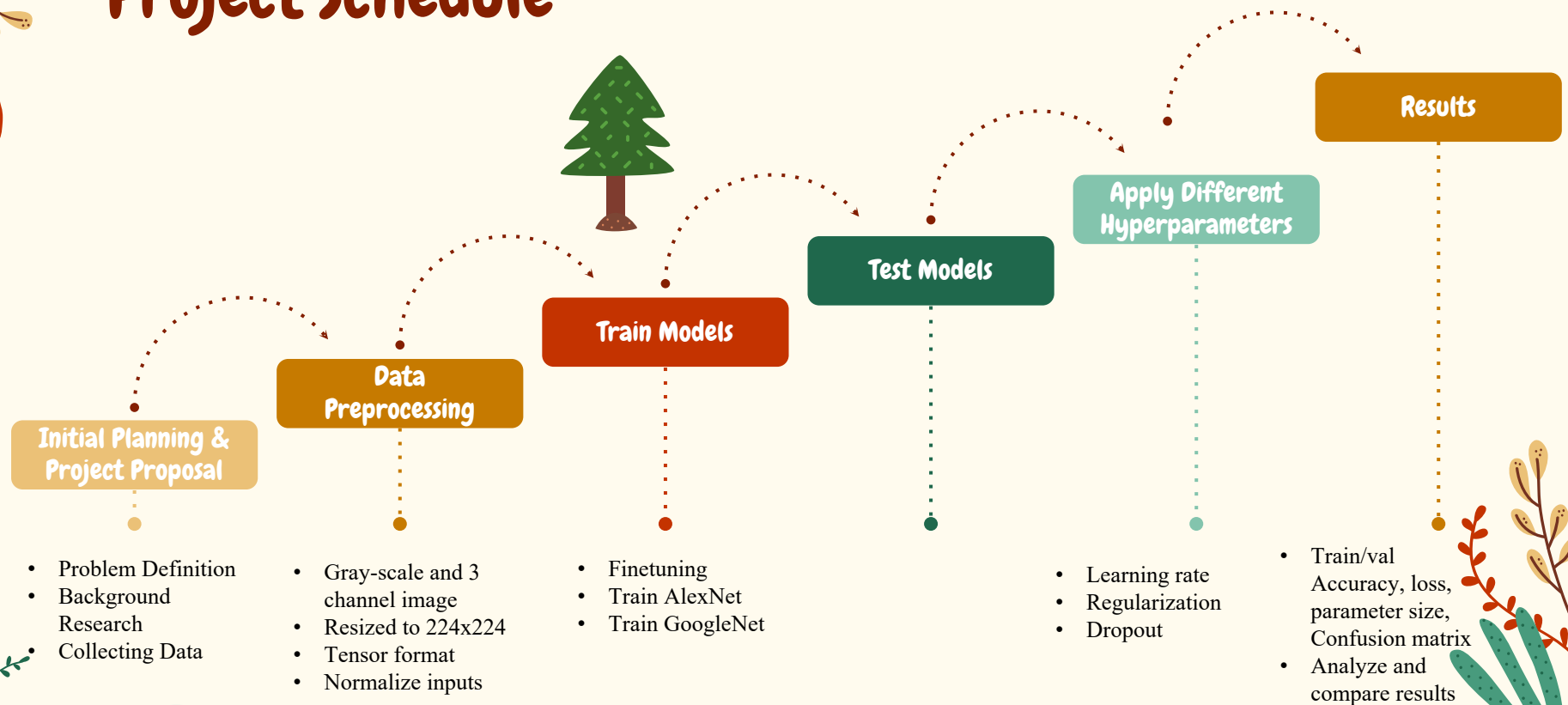


Bonus

(if enough time)

Reducing the sizes of filters used or applying a 1x1 bottleneck layer to AlexNet

Project Schedule



Conclusion

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

- Use two important CNN architectures in the classification problem
- Evaluate in terms of performance, speed and confusion.
- Discuss the strengths and weaknesses of the models
- Finetuning methods
- Retrain using different hyper parameters (learning rate, regularization etc.)
- Compare results
- Observe whether there will be an improvement in the performance of the models when different parameters are used
- (If there will be enough time), Add a 1x1 bottleneck layer to the AlexNet and examine how its performance changes





**Thank you for
listening**