BLG 506E – COMPUTER VISION

Project Proposal PRESENTATION

CANSU YANIK - 504201588

Table of Contents

01 Motivation

02 About the Project

03 Related Works

04 Project Goals and Impacts

05 Project Schedule

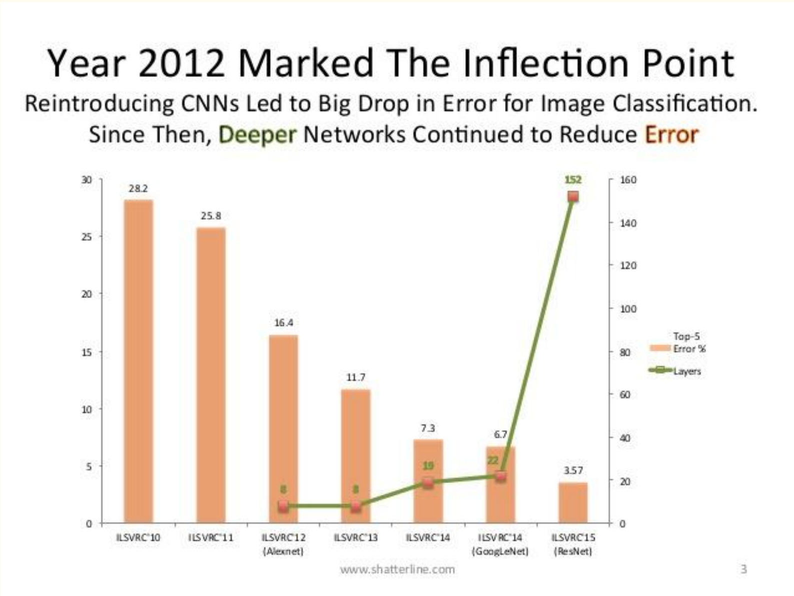
06 Conclusion

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× 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

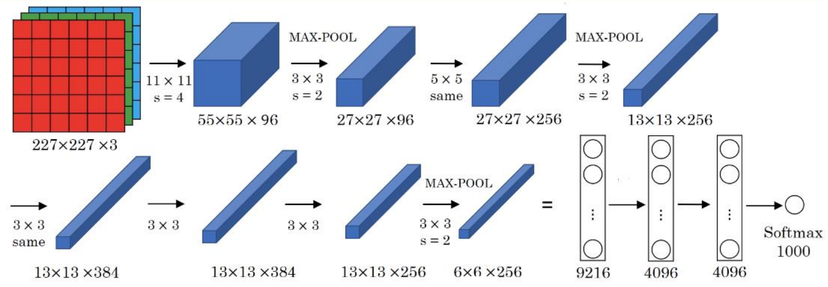
- 12x fewer parameters
- More accurate
- Lower memory use
- Much bigger network than the AlexNet

About the Project

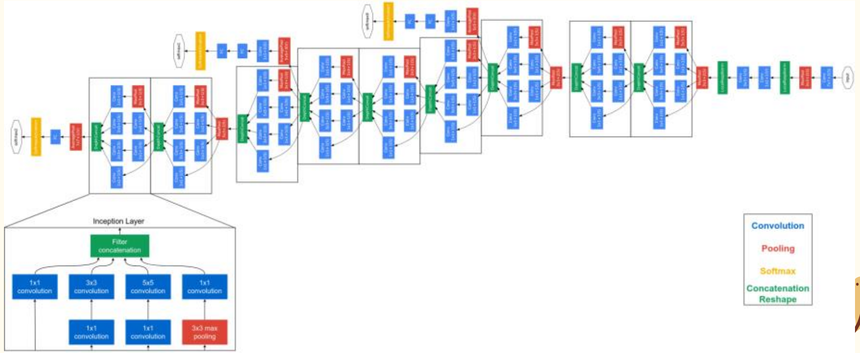
Generating FEN Descriptions of Chess Boards by Using Two Important Pre-trained CNN Architectures

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

AlexNet



GoogleNet

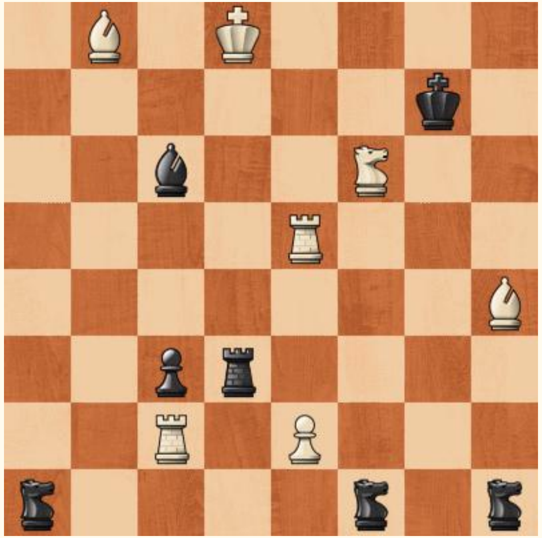


Generating FEN Descriptions of Chess Boards by Using Two Important Pre-trained CNN Architectures

DATASET

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

A sample of dataset and its label

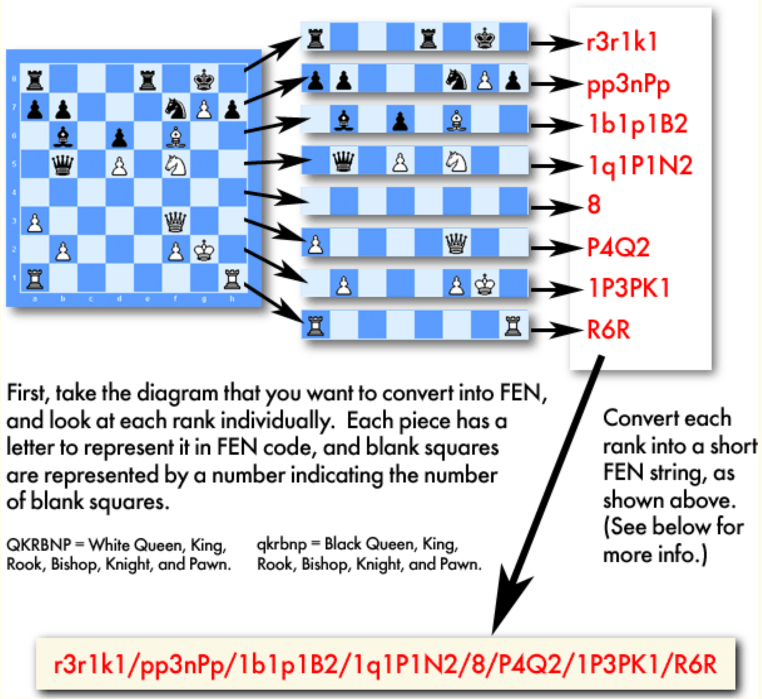


1B1K4-6k1-2b2N2-4R3-7B-2pr4-2R1P3-n4n1n.jpeg

- 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)



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.

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

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
================================================================
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
----------------------------------------------------------------
```

Freeze these layers

Train these layers

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]             384
      BasicConv2d-10          [-1, 192, 56, 56]               0
        MaxPool2d-11          [-1, 192, 28, 28]               0
           Conv2d-12           [-1, 64, 28, 28]          12,288
      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, 192, 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-173            [-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
================================================================
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
----------------------------------------------------------------
```

Freeze these layers

Train these layers

Finetuning GoogleNet Model

Related Works

2012

Model	Top-1 (val)	Top-5 (val)	Top-5 (test)
SIFT + FVs [7]	—	—	*26.2%*
1 CNN	40.7%	18.2%	—
5 CNNs	38.1%	16.4%	**16.4%**
1 CNN*	39.0%	16.6%	—
7 CNNs*	36.7%	15.4%	**15.3%**

Krizhevsky, Alex & Sutskever, Ilya & Hinton, Geoffrey. (2012). ImageNet Classification with Deep Convolutional Neural Networks.

2014

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× 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].

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

2018

GoogLeNet achieves better result compared to AlexNet and custom CNN models. Even though AlexNet and GoogLeNet yield similar accuracy which is 99.65%, GoogLeNet achieves its consistency at an earlier rate. The error function drops early below

Zabir, M. & Fazira, N. & Ibrahim, Zaidah & Sabri, Nurbaity. (2018). Evaluation of Pre-Trained Convolutional Neural Network Models for Object Recognition.

2019

PERFORMANCE ANALYSIS

Architecture	Accuracy	Precision	Recall	F1-score	Time
AlexNet	**0.9298**	0.9906	0.9325	0.9606	**19 m 41 s**
GoogLeNet	**0.8596**	0.9634	0.8830	0.9171	**36 m 40 s**
Inception V3	**0.8684**	0.9698	0.8705	0.9162	**128 m 29 s**
ResNet	**0.9605**	0.9167	0.9569	0.9776	**100 m 15 s**

A. P. Rahmathunneesa and K. V. Ahammed Muneer, 2019, "Performance Analysis of Pre-trained Deep Learning Networks for Brain Tumor Categorization.

2020

Model	Accuracy(%)
AlexNet-TL	62.94
GoogleNet-TL	63.79
ResNet-TL	**66.84**
DenseNet-TL	**68.15**

Baykal, E., Dogan, H., Ercin, M.E. et al. 2020. Transfer learning with pre-trained deep convolutional neural networks for serous cell classification.

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, data augmentation, regularization 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, data augmentation, regularization etc.)

Bonus

(if enough time)

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

Project Schedule

Initial Planning & Project Proposal

- Problem Definition
- Background Research
- Collecting Data

Data Preprocessing

- Gray-scale and 3 channel image
- Resized to 224x224
- Tensor format
- Normalize inputs

Train Models

- Finetuning
- Train AlexNet
- Train GoogleNet

Test Models

Apply Different Hyperparameters

- Learning rate
- Regularization
- Dropout

Results

- Train/val Accuracy, loss, parameter size, Confusion matrix
- Analyze and compare results

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.
- Discusse the strengths and weaknesses of the models
- Finetuning methods
- Retraine 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