BLG 506E – COMPUTER VISION

TERM PROJECT PROGRESS PRESENTATION

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

CANSU YANIK - 504201588

Table of Contents

01 Problem statement

02 Applied processes/methods

03 Intermediate results & discussions

04 Remaining work & timeline

05 Conclusion

Problem Statement

Goal of this Project

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

Rebuilt models with different hyperparameters

How results change?

Three exaple images and labels from train dataset

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

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

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

Chess Board Image Size for Test:
200 Board Image

Are they little amount?

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

1. Taking FEN notations and converting FEN to label

Label size = 13

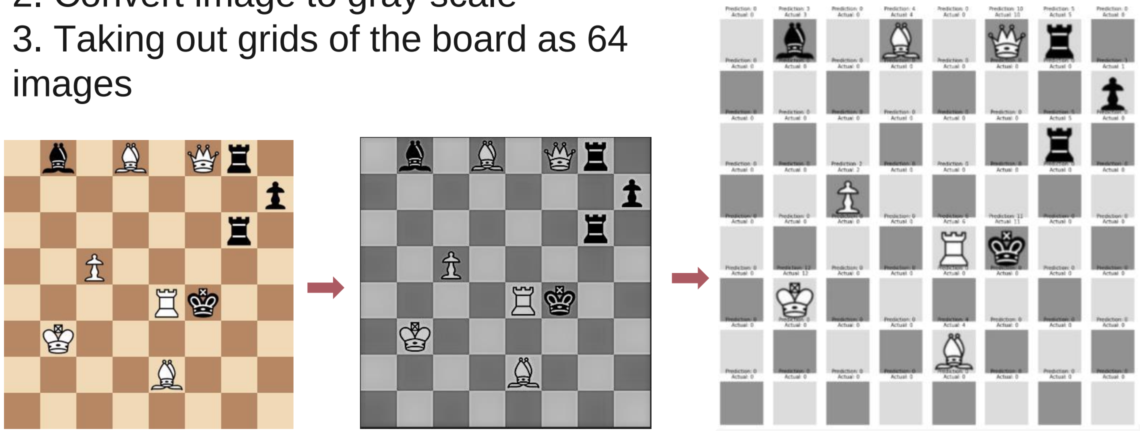
emptyGrid = 0
blackPawn = 1
whitePawn = 2
blackBishop = 3
whiteBishop = 4
blackRock = 5
whiteRock = 6
blackKnight = 7
whiteKnight = 8
blackQueen = 9
whiteQueen = 10
blackKing = 11
whiteKing = 12

```
FEN Notation:
./dataset/train\2K5-1N1PN3-8-3r2k1-8-1R1B1B1b-8-6R1.jpeg
Labels:
[[0, 0, 12, 0, 0, 0, 0, 0, 0, 8, 0, 2, 8, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 5, 0, 0, 11, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 6, 0, 4, 0, 4, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 6, 0]]
```

Applied Processes/Methods

1) Random image selectio
   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

2. Convert image to gray scale
3. Taking out grids of the board as 64 images



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

4. Resize Images to (224,224)

5. Convert gray Images to 3 channel Images

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

```
transform = transforms.Compose([
  transforms.ToTensor(),
  transforms.Normalize(
  mean=[0.485, 0.456, 0.406],
  std=[0.229, 0.224, 0.225]
)])

class CustomDataset(torch.utils.data.Dataset):
    def __init__(self, imageX, labelX):
        self.images = imageX
        self.labels = labelX
        self.transform = transform

    def __getitem__(self, index):

        img = self.transform(self.images[index])
        label = self.labels[index]
        return img, label

    def __len__(self):
        return len(self.images)
```

Convert to tensor format !

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

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

Create Data Loaders for train, validation and test datasets !

```
train_set, validation_set = torch.utils.data.random_split(custom_dataset,[int(train_split),int(validation_split)])
train_loader = torch.utils.data.DataLoader(dataset=train_set, batch_size=batch_size,shuffle=True)
validation_loader = torch.utils.data.DataLoader(dataset=validation_set, batch_size=batch_size,shuffle=True)
test_loader = torch.utils.data.DataLoader(dataset=test_custom_dataset, batch_size=batch_size,shuffle=False)
```

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

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)
)
```

Applied Processes/Methods

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

```python
AlexNetModel = model
batch_size = 64
learning_rate = 0.001
weight_decay = 0.0000001
num_epochs  = 10
#momentum = 0.9

criterion = nn.CrossEntropyLoss()
#optimizer = torch.optim.Adam(model.classifier.parameters(), lr=learning_rate)
optimizer = torch.optim.Adam(filter(lambda x: x.requires_grad, model.parameters()), lr=learning_rate)
#optimizer = optim.SGD(model.classifier.parameters(), lr=learning_rate, momentum=momentum)
```

- 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

```
Number of images: 128000, Number of batches: 1600
Epoch: [1/10] | Train Loss: 0.074 | Train Accuracy: 0.986 | Val Loss: 0.010 | Val Accuracy: 0.998|
Validation Loss Decreased: inf ---> 0.009990 ---> Saving Model
Epoch: [2/10] | Train Loss: 0.065 | Train Accuracy: 0.994 | Val Loss: 0.004 | Val Accuracy: 0.999|
Validation Loss Decreased: 0.009990 ---> 0.003745 ---> Saving Model
Epoch: [3/10] | Train Loss: 0.055 | Train Accuracy: 0.996 | Val Loss: 0.002 | Val Accuracy: 1.000|
Validation Loss Decreased: 0.003745 ---> 0.002439 ---> Saving Model
Epoch: [4/10] | Train Loss: 0.035 | Train Accuracy: 0.997 | Val Loss: 0.001 | Val Accuracy: 1.000|
Validation Loss Decreased: 0.002439 ---> 0.001436 ---> Saving Model
Epoch: [5/10] | Train Loss: 0.048 | Train Accuracy: 0.997 | Val Loss: 0.034 | Val Accuracy: 0.999|
Epoch: [6/10] | Train Loss: 0.035 | Train Accuracy: 0.998 | Val Loss: 0.007 | Val Accuracy: 0.999|
Epoch: [7/10] | Train Loss: 0.044 | Train Accuracy: 0.998 | Val Loss: 0.004 | Val Accuracy: 1.000|
Epoch: [8/10] | Train Loss: 0.054 | Train Accuracy: 0.998 | Val Loss: 0.001 | Val Accuracy: 1.000|
Validation Loss Decreased: 0.001436 ---> 0.001176 ---> Saving Model
Epoch: [9/10] | Train Loss: 0.040 | Train Accuracy: 0.998 | Val Loss: 0.004 | Val Accuracy: 1.000|
Epoch: [10/10] | Train Loss: 0.050 | Train Accuracy: 0.998 | Val Loss: 0.000 | Val Accuracy: 1.000|
Validation Loss Decreased: 0.001176 ---> 0.000061 ---> Saving Model
```

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

```python
with torch.no_grad():
    model.eval()
    correct = 0
    total = 0
    for images, labels in test_loader:
        images = images.to(device)
        labels = labels.to(device)
        #(no grad calculation)

        outputs = model(images)

        # Get predictions and calculate accuracy
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

    print('Accuracy of the network on the 12800 test images: {} %'.format(100 * correct / total))
```

AlexNet

```
Number of images: 64000, Number of batches: 800
Epoch: [1/10] | Train Loss: 0.120 | Train Accuracy: 0.974 | Val Loss: 0.011 | Val Accuracy: 0.997 |
Validation Loss Decreased: inf ---> 0.010798 ---> Saving Model
Epoch: [2/10] | Train Loss: 0.046 | Train Accuracy: 0.991 | Val Loss: 0.008 | Val Accuracy: 0.998 |
Validation Loss Decreased: 0.010798 ---> 0.007895 ---> Saving Model
Epoch: [3/10] | Train Loss: 0.054 | Train Accuracy: 0.992 | Val Loss: 0.008 | Val Accuracy: 0.998 |
Validation Loss Decreased: 0.007895 ---> 0.007894 ---> Saving Model
Epoch: [4/10] | Train Loss: 0.054 | Train Accuracy: 0.994 | Val Loss: 0.003 | Val Accuracy: 0.999 |
Validation Loss Decreased: 0.007894 ---> 0.002764 ---> Saving Model
Epoch: [5/10] | Train Loss: 0.042 | Train Accuracy: 0.995 | Val Loss: 0.002 | Val Accuracy: 0.999 |
Validation Loss Decreased: 0.002764 ---> 0.001925 ---> Saving Model
Epoch: [6/10] | Train Loss: 0.037 | Train Accuracy: 0.996 | Val Loss: 0.012 | Val Accuracy: 0.998 |
Epoch: [7/10] | Train Loss: 0.050 | Train Accuracy: 0.996 | Val Loss: 0.003 | Val Accuracy: 1.000 |
Epoch: [8/10] | Train Loss: 0.059 | Train Accuracy: 0.996 | Val Loss: 0.004 | Val Accuracy: 0.999 |
Epoch: [9/10] | Train Loss: 0.047 | Train Accuracy: 0.997 | Val Loss: 0.003 | Val Accuracy: 1.000 |
Epoch: [10/10] | Train Loss: 0.052 | Train Accuracy: 0.997 | Val Loss: 0.001 | Val Accuracy: 1.000
Validation Loss Decreased: 0.001925 ---> [illegible] ---> Saving Model
Accuracy of the network on the 12800 test images: 99.984375 %
```

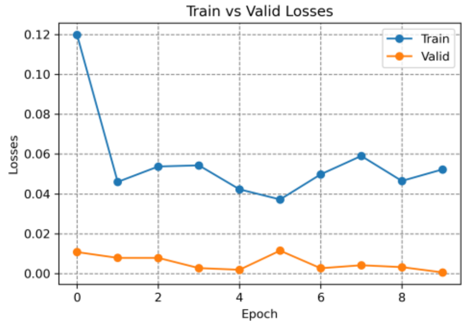
GoogleNet

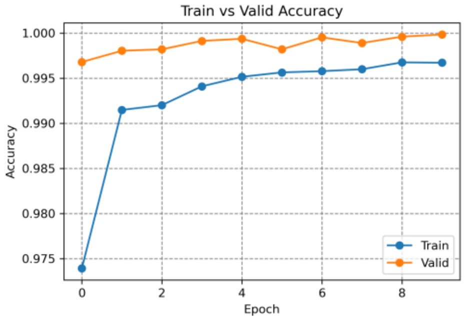
```
Number of images: 64000, Number of batches: 800
Epoch: [1/10] | Train Loss: 0.257 | Train Accuracy: 0.949 | Val Loss: 0.066 | Val Accuracy: 0.986|
Validation Loss Decreased: inf ---> 0.065633 ---> Saving Model
Epoch: [2/10] | Train Loss: 0.064 | Train Accuracy: 0.986 | Val Loss: 0.035 | Val Accuracy: 0.994|
Validation Loss Decreased: 0.065633 ---> 0.035024 ---> Saving Model
Epoch: [3/10] | Train Loss: 0.044 | Train Accuracy: 0.990 | Val Loss: 0.027 | Val Accuracy: 0.994|
Validation Loss Decreased: 0.035024 ---> 0.027179 ---> Saving Model
Epoch: [4/10] | Train Loss: 0.037 | Train Accuracy: 0.991 | Val Loss: 0.025 | Val Accuracy: 0.993|
Validation Loss Decreased: 0.027179 ---> 0.024920 ---> Saving Model
Epoch: [5/10] | Train Loss: 0.029 | Train Accuracy: 0.993 | Val Loss: 0.017 | Val Accuracy: 0.996|
Validation Loss Decreased: 0.024920 ---> 0.016835 ---> Saving Model
Epoch: [6/10] | Train Loss: 0.027 | Train Accuracy: 0.993 | Val Loss: 0.016 | Val Accuracy: 0.996|
Validation Loss Decreased: 0.016835 ---> 0.015948 ---> Saving Model
Epoch: [7/10] | Train Loss: 0.025 | Train Accuracy: 0.993 | Val Loss: 0.016 | Val Accuracy: 0.996|
Epoch: [8/10] | Train Loss: 0.023 | Train Accuracy: 0.994 | Val Loss: 0.014 | Val Accuracy: 0.996|
Validation Loss Decreased: 0.015948 ---> 0.014224 ---> Saving Model
Epoch: [9/10] | Train Loss: 0.021 | Train Accuracy: 0.994 | Val Loss: 0.015 | Val Accuracy: 0.995|
Epoch: [10/10] | Train Loss: 0.020 | Train Accuracy: 0.995 | Val Loss: 0.013 | Val Accuracy: 0.996|
Validation Loss Decreased: 0.014224 ---> 0.012909 ---> Saving Model
Accuracy of the network on the 12800 test images: 99.578125 %
```

Intermediate results & discussions

8) Evaluation of Model using Test dataset

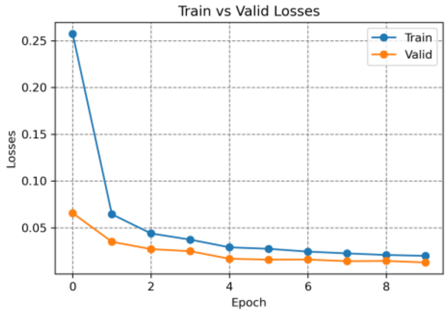
AlexNet

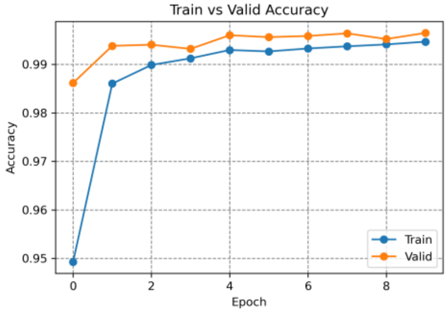




Accuracy of the network on the 12800 test images: 99.984375 %

GoogleNet





Accuracy of the network on the 12800 test images: 99.578125 %

Intermediate results & discussions

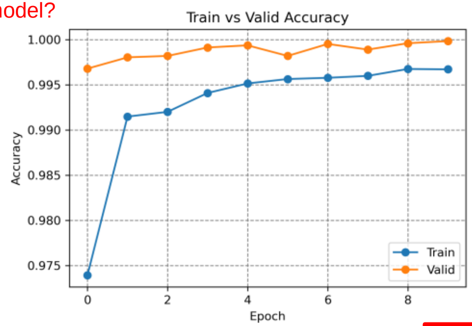
No Underfit

AlexNet

Unrepresentative Validation Dataset?

Gap? Overfitting?

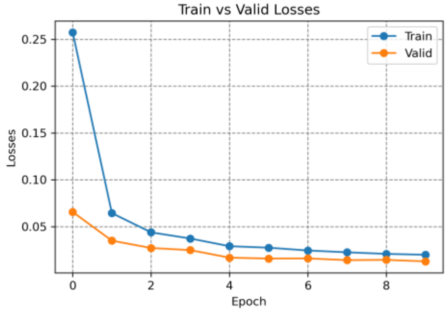
Validation dataset easier for the model?



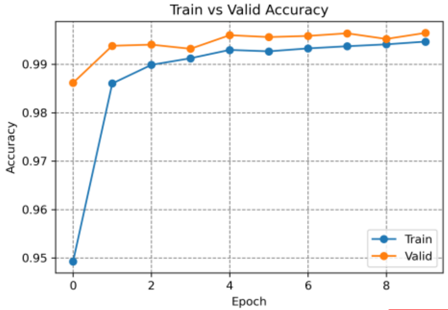
Batch size small?

Accuracy of the network on the 12800 test images: 99.984375 %

GoogleNet



Looks good ?



Accuracy of the network on the 12800 test images: 99.578125 %

Intermediate results & discussions

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

Intermediate results & discussions

Better!

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 ne overfit
✓	No Underfit
✓	More smooth (no noisy movements)
✓	Curves looks good!
✓	Less parameters to learn

```
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'}
```

Some Predictions

Actual FEN

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

AlexNet FEN

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

GoogleNet FEN

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

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'}
```

Actual FEN

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

AlexNet FEN

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

GoogleNet FEN

Rn1qR1k1-ppp3pp-8-5Q2-3Nn3-1P1P2P1-PB3PBP-R3K2R

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'}
```

Actual FEN

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

AlexNet FEN

4kb1r-5ppp-4pn2-p1pp4-4P3-2P1pP2-PP3p1P-R4K1R

GoogleNet FEN

4kk1r-5pbp-4pn2-p1pp4-4p3-2p1BP2-pP3P1P-R4K1R

Remaining work & timeline

Apply Different Hyperparameters

- Learning rate
- Optimizer
- Batch size
- Regularization
- Dropout

Analyizing Models

- Train & Test Models with different hyperparameters
- Get Results

Compare Results

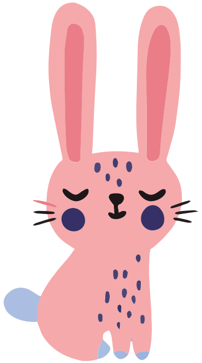
- Train & val accuracy
- Train & val loss
- PR & ROC curves can be used!
- Different performans metrics?

Final Work

- Final report
- Final presentation

Conclusion

- 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.
- However AlexNet has higher accuracy.
- And AlexNet predicts better.
- AlexNet has overfit and unrepresentative validation dataset problem.
- Models may predict false when chess pieces are so different than dataset.



Thank you for listening!