📖

# [논문 리뷰] GoogLeNet [2014]

## 1. Introduction

🌟 Inception이라고 불리는 NeuralNet을 propose합니다. 신중하게 달성한 네트워크 깊이와 너비를 늘리면서, 정교한 디자인으로 일정함을 유지합니다. 22 layers deep network를 가지고 있고, Classification and Detection에서 접근하는 것이 가능합니다.

해당 논문은 Inception이라고 불리는 Computer Vision에서 효율적인 Deep Neural Netwrok 구조에 집중하고 있습니다.

해당 논문에서 “deep”은 두가지 의미를 가지고 있습니다.

- “Inception Module”의 형태를 가진 새로운 차원의 구조 도입
- Network의 depth Increased

## 2. Related Work

**Inception layer**들은 여러 번 반복되고,

GoogLeNet은 **leading to a 22-layer deep model**로 구성되어 있습니다.

Network-in-Network는 **Neural Network에서 표현능력을 제안하기 위해 표현**되었습니다. NIN이 CNN layer에 적용될 때, 해당 방법은 1x1 conv에 ReLU를 적용시키는것과 동일합니다.

해당 방법은 CNN piplines를 적용하기 쉽습니다.

**1X1 Convolutions의 두가지 목적**

- Computational bottlenecks를 제거하기 위해 **dimension reduction modules를 사용**합니다.
- Significant performace penatly없이 **NeuralNet의 depth, width를 증가**시킬 수 있습니다.

**R-CNN(Region with Convolutional Neural Networks) Detection 문제를 두 가지로 분류합니다.**

- Color과 SuperPixel의 Consistency로 Potential object proposal을 제안하는 것이다.
- 해당 위치에 카테고리를 분류하는 것입니다.

GoogLeNet은 R-CNN에서와 유사한 방법으로 similar pipeline을 채택합니다.

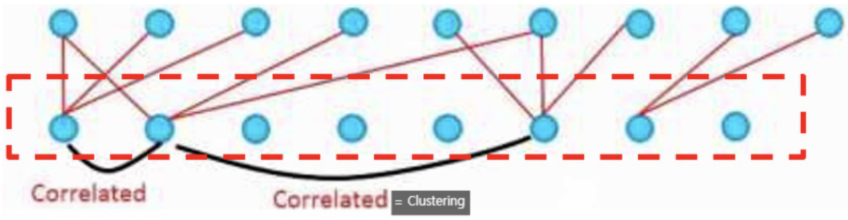
multi-box prediction으로부터 더 좋은 성능을 얻었으며, Ensemble접근법을 활용해 성능이 향상

## 3. Motivation and High Level Considerations

NeuralNetwork의 성능을 개선하는 방법은 depth, the number of levels, width, units을 증가시키는 것이다. 하지만 **두가지 중대한 문제점(two major drawbacks)**이 존재합니다.

- Size가 커지면, 많은 파라미터가 존재하게 되고, Overfitting을 야기하게 됩니다.
- Computational resources가 dramatically하게 증가한다는 것입니다.

이러한 문제를 해결하는 방식은, **CNN 내에 FC를 드물게 섞음으로서, 해결 가능**합니다.



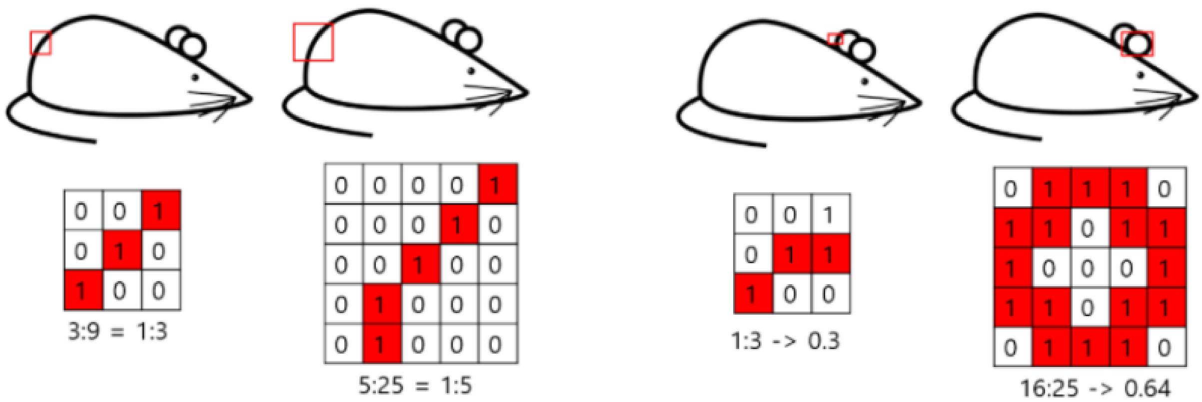

이때, Sparse 매트릭스 연산을 다룬 많은 문헌에서는 **Sparse 매트릭스를 클러스터링 하여 상대적으로 Dense 한 Submatrix를 만드는 것을 제안**하였고, 이는 좋은 성능을 보였다고 한다.

Inception 구조는, 유사 Sparas 구조를 시험하기 위해 시작되었습니다. 여러가지 tunning 과 learning rate를 조정한 결과, **Localization and object detection을 base로 둔 곳에서 Useful한 것**을 알 수 있었습니다.

# 4. Architecture Details

Inception Architectures는 2가지 main Idea로 형성되었다. optimal local sparse structure(최적의 희소 구조)에 approximated 하고, Dense Components로 변환을 하는 것이다.

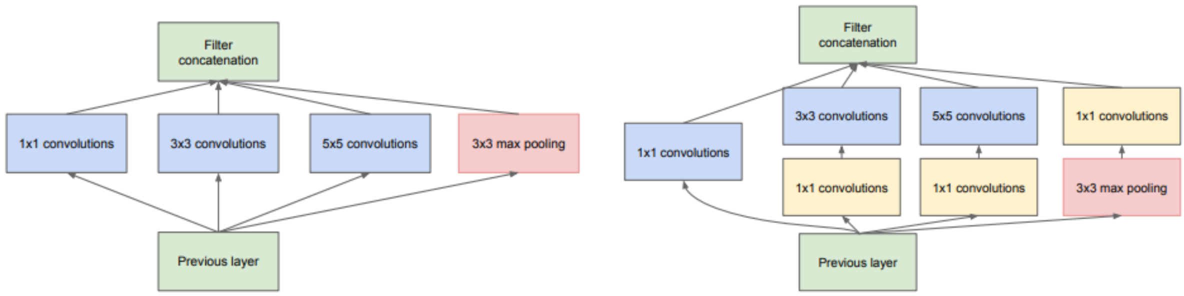
🌟 All we need is to find the optimal local construction and to repeat it spatially.




하지만, 몇몇 위치에서는 좀 더 넓은 영역의 Convolutional filter가 있어야 Coreelated unit의 비율을 높일 수 있다. 또한, Patch-alignment issues를 피하기 위해서는, **1x1, 3x3, 5x5 Convolutional 연산**을 시행해야 합니다.

🌟 Inception modules들이 위에서부터 쌓이기 때문에,  Output correlation statistics는 다양합니다.

As these "Inception modules" are stacked on top of each other, their output correlation statistics are bound to vary: as features of higher abstraction are captured by higher layers, their spatial concentration is expected to decrease suggesting that the ratio of $3\times3$ and $5\times5$ convolutions should increase as we move to higher layers.




(a) Inception module, naïve version (b) Inception module with dimension reductions

Figure 2: Inception module

💥여기서 문제가 발생한다.

**3x3 Convolutional filter 뿐만 아니라, 5x5 Convolutional fiilter를 사용**하는 경우, 연산량이 많아져,

**Computational이 매우 증가**하게 된다. 추가적으로, **ReLU라는 비선형적 특징을 추가**할 수 있습니다.

> In general, an Inception network is a network consisting of modules of the above type stacked upon each other, with occasional max-pooling layers with stride 2 to halve the resolution of the grid. For technical reasons (memory efficiency during training), it seemed beneficial to start using Inception modules only at higher layers while keeping the lower layers in traditional convolutional fashion. This is not strictly necessary, simply reflecting some infrastructural inefficiencies in our current implementation.

Inception module을 사용하면, 두 가지 효과를 볼 수 있습니다.

- 과도한 연산량 문제없이 각 단계에서 **유닛 수를 상당히 증가**시킨다. (layer의 input 조절)
- Visual 정보가 다양한 Scale로 처리되고, **다음 layer는 동시에 서로 다른 layer의 특징 추출**

# 5. GoogLeNet

🌟 LeNet으로부터 유래되었으며, incarnation of the Inception architecture입니다.

All the convolutions, including those inside the Inception modules, use rectified linear activation.

(**모든 Convolutions과 Inception modules 내부를 포함하여, ReLU를 사용합니다.**)

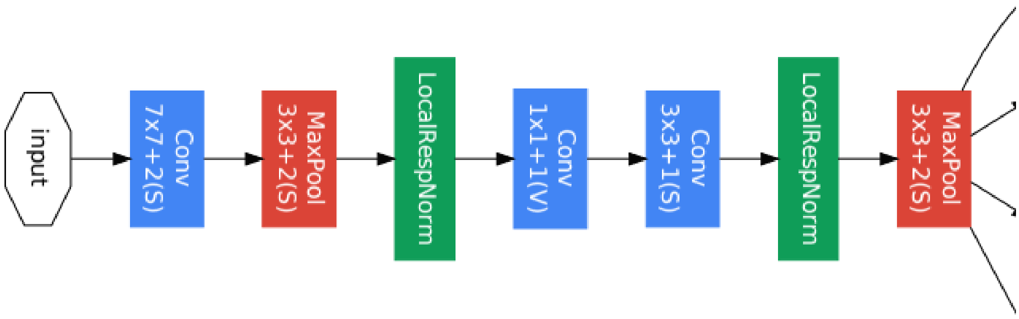
| type | patch size/ stride | output size | depth | #1×1 | #3×3 reduce | #3×3 | #5×5 reduce | #5×5 | pool proj | params | ops |
|---|---|---|---|---|---|---|---|---|---|---|---|
| convolution | 7×7/2 | 112×112×64 | 1 | | | | | | | 2.7K | 34M |
| max pool | 3×3/2 | 56×56×64 | 0 | | | | | | | | |
| convolution | 3×3/1 | 56×56×192 | 2 | | 64 | 192 | | | | 112K | 360M |
| max pool | 3×3/2 | 28×28×192 | 0 | | | | | | | | |
| inception (3a) | | 28×28×256 | 2 | 64 | 96 | 128 | 16 | 32 | 32 | 159K | 128M |
| inception (3b) | | 28×28×480 | 2 | 128 | 128 | 192 | 32 | 96 | 64 | 380K | 304M |
| max pool | 3×3/2 | 14×14×480 | 0 | | | | | | | | |
| inception (4a) | | 14×14×512 | 2 | 192 | 96 | 208 | 16 | 48 | 64 | 364K | 73M |
| inception (4b) | | 14×14×512 | 2 | 160 | 112 | 224 | 24 | 64 | 64 | 437K | 88M |
| inception (4c) | | 14×14×512 | 2 | 128 | 128 | 256 | 24 | 64 | 64 | 463K | 100M |
| inception (4d) | | 14×14×528 | 2 | 112 | 144 | 288 | 32 | 64 | 64 | 580K | 119M |
| inception (4e) | | 14×14×832 | 2 | 256 | 160 | 320 | 32 | 128 | 128 | 840K | 170M |
| max pool | 3×3/2 | 7×7×832 | 0 | | | | | | | | |
| inception (5a) | | 7×7×832 | 2 | 256 | 160 | 320 | 32 | 128 | 128 | 1072K | 54M |
| inception (5b) | | 7×7×1024 | 2 | 384 | 192 | 384 | 48 | 128 | 128 | 1388K | 71M |
| avg pool | 7×7/1 | 1×1×1024 | 0 | | | | | | | | |
| dropout (40%) | | 1×1×1024 | 0 | | | | | | | | |
| linear | | 1×1×1000 | 1 | | | | | | | 1000K | 1M |
| softmax | | 1×1×1000 | 0 | | | | | | | | |

Table 1: GoogLeNet incarnation of the Inception architecture

# 6. 4 Section of GoogLeNet

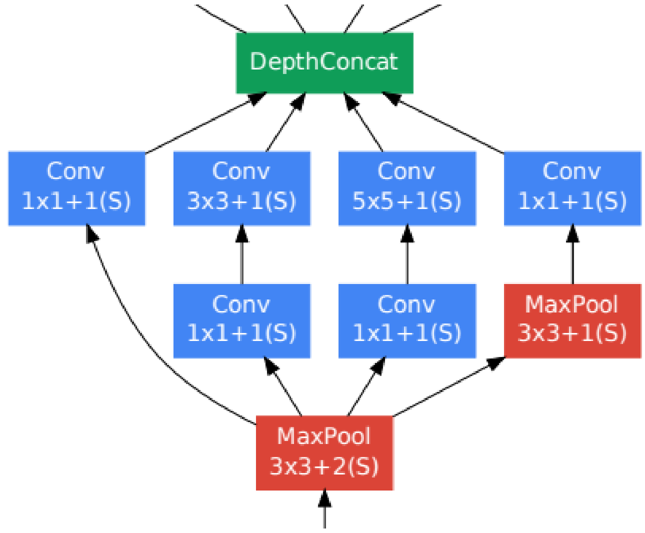
## PART 1

- 효율적인 메모리 사용을 위해 낮은 layer에서는 전통적인 CNN을 사용합니다.



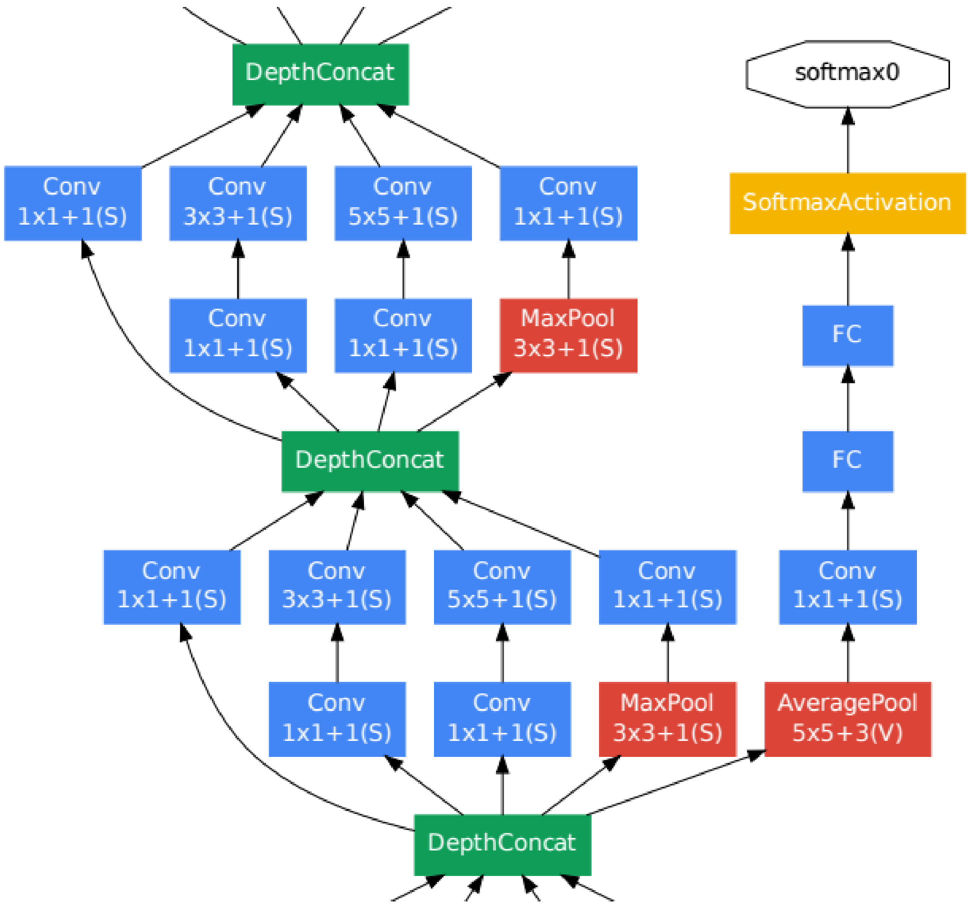

## PART 2

- Inception module로서, **Various Feature extract 하기 위해, 1x1, 3x3, 5x5, Convolutional layer가 병렬적으로 연산**을 수행하고 있으며, **차원을 축소하여 연산량을 줄이기 위해 1x1 Convolutional layer**를 적용하였습니다.



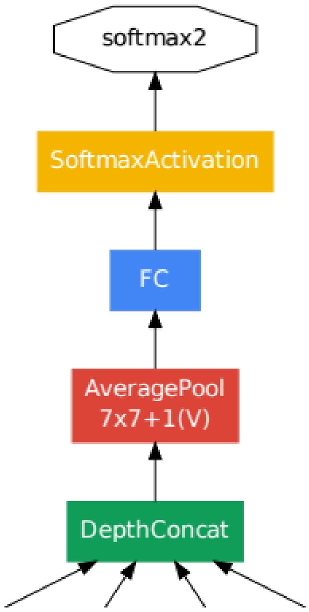

## PART 3

- Model's depth가 있을수록, 기울기가 0으로 수렴하는 **Gradient Vanishing 문제**가 발생하는 것을 알 수 있습니다. 그러므로, 중간 layer에 **auxiliary classifier를 추가**하여, 중간결과를 출력해, gradient가 전달될 수 있게 하여, **Normalize효과**가 나타나게 합니다.

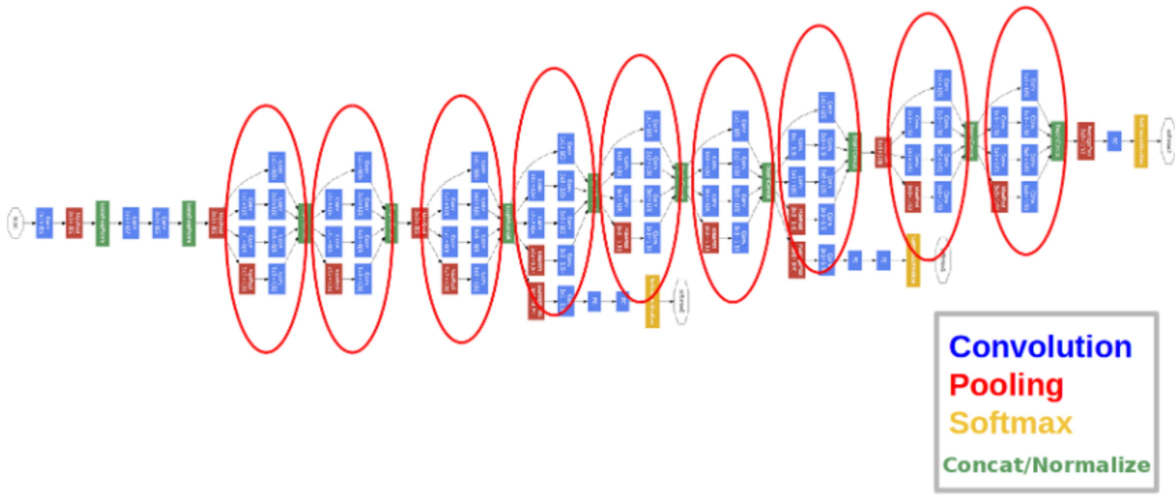



## PART 4

- Average Pooling layer를 사용하고, GAP가 적용된 것으로, 이전 layer에서 추출된 **feature map을 평균 내어 1차원 vector**로 만든다. 이후, **softmax layer와 연결**을 합니다.
- Averaging하여 **1차원 벡터로 만들면 가중치의 개수를 상당히 많이 줄여주며**, **GAP을 사용**하면 단 1개의 가중치도 필요하지 않다. 또한 **GAP 적용 시, fine tuning을 하기 매우 편리하다**.




FINAL MODEL




# 7. Training Methodology

0.9 Momentum의 Stochastic Gradient Descent를 이용하였으며,

learning rate는 8epochs마다 4%씩 감소를 시켰습니다.

또한, 이미지의 가로, 세로 비율을 3:4 와 4:3 사이로 유지하면서, 본래 사이즈의 8% ~ 100%가 포함되도록 다양한 크기의 patch를 사용하였습니다.

그리고 **photometric distortions를 통해 학습 데이터를 늘렸다.**

# 8. Conclusions

**Inception 구조는 Sparas 구조를 근사화하여, 성능을 개선**하였습니다. 이는 기존에 CNN 성능을 높이기 위한 새로운 방법이었으며,

**성능은 대폭 상승하지만, 연산량은 약간만 증가한다는 장점이 존재**합니다.

| Team | Year | Place | Error (top-5) | Uses external data |
|---|---|---|---|---|
| SuperVision | 2012 | 1st | 16.4% | no |
| SuperVision | 2012 | 1st | 15.3% | Imagenet 22k |
| Clarifai | 2013 | 1st | 11.7% | no |
| Clarifai | 2013 | 1st | 11.2% | Imagenet 22k |
| MSRA | 2014 | 3rd | 7.35% | no |
| VGG | 2014 | 2nd | 7.32% | no |
| GoogLeNet | 2014 | 1st | 6.67% | no |

Table 2: Classification performance

▼ **Inception [Pytorch]**

```
import torch
import torch.nn as nn

class Inception_block(nn.Module):
    def __init__(self, in_channels, out_1x1, red_3x3, out_3x3, red_5x5, out_5x5, out_1x1pool):
        super(Inception_block, self).__init__()

        self.branch1 = conv_block(in_channels, out_1x1, kernel_size = 1)
        self.branch2 = nn.Sequential(
            conv_block(in_channels, red_3x3, kernel_size = 1),
            conv_block(red_3x3, out_3x3, kernel_size = 3, padding = 1)
        )

        self.branch3 = nn.Sequential(
            conv_block(in_channels, red_5x5, kernel_size = 1),
            conv_block(red_5x5, out_5x5, kernel_size = 5,  padding = 2)
        )

        self.branch4 = nn.Sequential(
            nn.MaxPool2d(kernel_size=3, stride = 1, padding = 1),
            conv_block(in_channels, out_1x1pool, kernel_size = 1)
        )
```

▼ **Model [Pytroch]**

```
import torch
import torch.nn as nn

class GoogLeNet(nn.Module):
    def __init__(self, in_channels = 3, num_classes = 1000):
        super(GoogLeNet, self).__init__()

        self.conv1 = conv_block(in_channels = in_channels, out_channels = 64,
                                kernel_size = (7, 7), stride = (2, 2), padding = (3, 3))
        self.maxpool1 = nn.MaxPool2d(kernel_size= 3, stride = 2, padding = 1)
```

```python
        self.conv2 = conv_block(64, 192, kernel_size = 3, padding = 1, stride = 1)
        self.maxpool2 = nn.MaxPool2d(kernel_size=3, stride=2, padding=1)

        self.inception3a = Inception_block(192, 64, 96, 128, 16, 32, 32)
        self.inception3b = Inception_block(256, 128, 128, 192, 32, 96, 64)
        self.maxpool3 = nn.MaxPool2d(kernel_size=3, stride = 2, padding = 1)

        self.inception4a = Inception_block(480, 192, 96, 208, 16, 48, 64)
        self.inception4b = Inception_block(512, 160, 112, 224, 24, 64, 64)
        self.inception4c = Inception_block(512, 128, 128, 256, 24, 64, 64)
        self.inception4d = Inception_block(512, 112, 144, 288, 32, 64, 64)
        self.inception4e = Inception_block(528, 256, 160, 320, 32, 128, 128)
        self.maxpool4 = nn.MaxPool2d(kernel_size=3, stride = 2, padding = 1)

        self.inception5a = Inception_block(832, 256, 160, 320, 32, 128, 128)
        self.inception5b = Inception_block(832, 384, 192, 384, 48, 128, 128)

        self.avgpool = nn.AvgPool2d(7, 1)
        self.dropout = nn.Dropout(p = 0.4)
        self.fc1 = nn.Linear(1024, num_classes)

    def forward(self, x):
        x = self.conv1(x)
        x = self.maxpool1(x)
        x = self.conv2(x)
        x = self.maxpool2(x)

        x = self.inception3a(x)
        x = self.inception3b(x)
        x = self.maxpool3(x)

        x = self.inception4a(x)
        x = self.inception4b(x)
        x = self.inception4c(x)
        x = self.inception4d(x)
        x = self.inception4e(x)
        x = self.maxpool4(x)

        x = self.inception5a(x)
        x = self.inception5b(x)
        x = self.avgpool(x)
        x = x.reshape(x.shape[0], -1)
        x = self.dropout(x)
        x = self.fc1(x)

        return x

class Inception_block(nn.Module):
    def __init__(self, in_channels, out_1x1, red_3x3, out_3x3, red_5x5, out_5x5, out_1x1pool):
        super(Inception_block, self).__init__()

        self.branch1 = conv_block(in_channels, out_1x1, kernel_size = 1)
        self.branch2 = nn.Sequential(
            conv_block(in_channels, red_3x3, kernel_size = 1),
            conv_block(red_3x3, out_3x3, kernel_size = 3, padding = 1)
        )

        self.branch3 = nn.Sequential(
            conv_block(in_channels, red_5x5, kernel_size = 1),
            conv_block(red_5x5, out_5x5, kernel_size = 5,  padding = 2)
        )

        self.branch4 = nn.Sequential(
            nn.MaxPool2d(kernel_size=3, stride = 1, padding = 1),
            conv_block(in_channels, out_1x1pool, kernel_size = 1)
        )

    def forward(self, x):
        # N x filters x 28 x 28
        return torch.cat([self.branch1(x), self.branch2(x), self.branch3(x), self.branch4(x)], 1)
```

```
class conv_block(nn.Module):
    def __init__(self, in_channels, out_channels, **kwargs):
        super(conv_block, self).__init__()
        self.relu = nn.ReLU()
        self.conv = nn.Conv2d(in_channels = in_channels, out_channels = out_channels, **kwargs)
                            # kernel_size = (1, 1), (3, 3), (5,5 )
        self.batchnorm = nn.BatchNorm2d(out_channels)

    def forward(self, x):
        return self.relu(self.batchnorm(self.conv(x)))



x = torch.randn(3 , 3, 224, 224)
model = GoogLeNet()
print(model(x).shape)  # 3, 1000
```

### ▼ Baseline [Pytroch]

```
from datetime import datetime
import numpy as np
import matplotlib.pyplot as plt

import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.utils.data import DataLoader
from torchvision import transforms, datasets
```

```
random_seed = 42
lr = 1e-3
batch_size = 32
n_epochs = 15
img_size = 224
n_classes = 10
```

```
def get_accuracy(model, data_loader, device):
    correct_pred = 0
    n = 0

    with torch.no_grad():
        model.eval()
        for X, y_true in data_loader:
            X, y_true = X.to(device), y_true.to(device)

            y_pred = model(X)
            _, predicted_labels = torch.max(y_pred, 1)

            n += y_true.size(0)
            correct_pred += (predicted_labels == y_true).sum()

    return correct_pred.float() / n
```

```
def train(train_loader, model, criterion, optimizer, device):
    model.train()
    running_loss = 0
    for X, y_true in train_loader:
        optimizer.zero_grad()

        X, y_true = X.to(device), y_true.to(device)
        y_hat = model(X)
        loss = criterion(y_hat, y_true)
```

```
        running_loss += loss.item() * X.size(0)
        loss.backward()
        optimizer.step()

    epoch_loss = running_loss / len(train_loader.dataset)
    return model, optimizer, epoch_loss
```

```
def validation(valid_loader, model, criterion, device):
    model.eval()
    running_loss = 0
    for X, y_true in valid_loader:
        X, y_true = X.to(device), y_true.to(device)
        y_hat = model(X)
        loss = criterion(y_hat, y_true)
        running_loss += loss.item() * X.size(0)

    epoch_loss = running_loss / len(valid_loader.dataset)

    return model, epoch_loss
```

```
def training_loop(model, criterion, optimizer, train_loader, valid_loader, epochs, device, print_every = 1):
    best_loss = 1e10
    train_losses = []
    valid_losses = []

    for epoch in range(epochs):
        model, optimizer, train_loss = train(train_loader, model, criterion, optimizer, device)
        train_losses.append(train_loss)

        with torch.no_grad():
            model, valid_loss = validation(valid_loader, model, criterion, device)
            valid_losses.append(valid_loss)

        if epoch % print_every == (print_every - 1):
            train_acc = get_accuracy(model, train_loader, device)
            valid_acc = get_accuracy(model, valid_loader, device)

            print(f'{datetime.now().time().replace(microsecond=0)} --- '
                  f'Epoch: {epoch}\t'
                  f'Train loss: {train_loss:.4f}\t'
                  f'Valid loss: {valid_loss:.4f}\t'
                  f'Train accuracy: {100 * train_acc:.2f}\t'
                  f'Valid accuracy: {100 * valid_acc:.2f}')

    return model, optimizer, (train_losses, valid_losses)
```

```
common_transforms = transforms.Compose([
                transforms.ToTensor(),
                transforms.Resize(224),
                transforms.Normalize([meanR, meanG, meanB],
                                     [stdR, stdG, stdB])
])
```

```
train_dataset = datasets.CIFAR10(root = 'cifar10',
                                 train = True,
                                 transform = common_transforms,
                                 download = True)

test_dataset = datasets.CIFAR10(root = 'cifar10',
                                train = False,
                                transform = common_transforms,
                                download = True)
```

```
import numpy as np
meanRGB = [np.mean(x.numpy(), axis = (1, 2)) for x, _ in train_dataset]
stdRGB = [np.std(x.numpy(), axis = (1, 2)) for x, _ in train_dataset]

meanR = np.mean([m[0] for m in meanRGB])
meanG = np.mean([m[1] for m in meanRGB])
meanB = np.mean([m[2] for m in meanRGB])

stdR = np.mean([m[0] for m in stdRGB])
stdG = np.mean([m[1] for m in stdRGB])
stdB = np.mean([m[2] for m in stdRGB])

common_transforms = transforms.Compose([
                transforms.ToTensor(),
                transforms.Resize(224),
                transforms.Normalize([meanR, meanG, meanB],
                                     [stdR, stdG, stdB])
])
```

```
from sklearn.model_selection import StratifiedShuffleSplit
sss = StratifiedShuffleSplit(n_splits = 1, test_size = 0.2, random_state = 0)
indices = list(range(len(test_dataset)))
y_test0 = [y for _, y in test_dataset]

for test_index, val_index in sss.split(indices, y_test0):
    print('test :', len(test_index) , 'val :', len(val_index))

from torch.utils.data import Subset

valid_dataset = Subset(test_dataset, val_index)
test_dataset = Subset(test_dataset, test_index)

train_loader = DataLoader(train_dataset, batch_size = batch_size, shuffle = True)
valid_loader = DataLoader(valid_dataset, batch_size = batch_size, shuffle = False)
test_loader = DataLoader(test_dataset, batch_size = batch_size, shuffle = False)
```

```
import torch
import torch.nn as nn

class GoogLeNet(nn.Module):
    def __init__(self, in_channels = 3, num_classes = 1000):
        super(GoogLeNet, self).__init__()

        self.conv1 = conv_block(in_channels = in_channels, out_channels = 64,
                                kernel_size = (7, 7), stride = (2, 2), padding = (3, 3))
        self.maxpool1 = nn.MaxPool2d(kernel_size= 3, stride = 2, padding = 1)

        self.conv2 = conv_block(64, 192, kernel_size = 3, padding = 1, stride = 1)
        self.maxpool2 = nn.MaxPool2d(kernel_size=3, stride=2, padding=1)

        self.inception3a = Inception_block(192, 64, 96, 128, 16, 32, 32)
        self.inception3b = Inception_block(256, 128, 128, 192, 32, 96, 64)
        self.maxpool3 = nn.MaxPool2d(kernel_size=3, stride = 2, padding = 1)

        self.inception4a = Inception_block(480, 192, 96, 208, 16, 48, 64)
        self.inception4b = Inception_block(512, 160, 112, 224, 24, 64, 64)
        self.inception4c = Inception_block(512, 128, 128, 256, 24, 64, 64)
        self.inception4d = Inception_block(512, 112, 144, 288, 32, 64, 64)
        self.inception4e = Inception_block(528, 256, 160, 320, 32, 128, 128)
        self.maxpool4 = nn.MaxPool2d(kernel_size=3, stride = 2, padding = 1)

        self.inception5a = Inception_block(832, 256, 160, 320, 32, 128, 128)
        self.inception5b = Inception_block(832, 384, 192, 384, 48, 128, 128)

        self.avgpool = nn.AvgPool2d(7, 1)
        self.dropout = nn.Dropout(p = 0.4)
        self.fc1 = nn.Linear(1024, num_classes)
```

```python
    def forward(self, x):
        x = self.conv1(x)
        x = self.maxpool1(x)
        x = self.conv2(x)
        x = self.maxpool2(x)

        x = self.inception3a(x)
        x = self.inception3b(x)
        x = self.maxpool3(x)

        x = self.inception4a(x)
        x = self.inception4b(x)
        x = self.inception4c(x)
        x = self.inception4d(x)
        x = self.inception4e(x)
        x = self.maxpool4(x)

        x = self.inception5a(x)
        x = self.inception5b(x)
        x = self.avgpool(x)
        x = x.reshape(x.shape[0], -1)
        x = self.dropout(x)
        x = self.fc1(x)

        return x

class Inception_block(nn.Module):
    def __init__(self, in_channels, out_1x1, red_3x3, out_3x3, red_5x5, out_5x5, out_1x1pool):
        super(Inception_block, self).__init__()

        self.branch1 = conv_block(in_channels, out_1x1, kernel_size = 1)
        self.branch2 = nn.Sequential(
            conv_block(in_channels, red_3x3, kernel_size = 1),
            conv_block(red_3x3, out_3x3, kernel_size = 3, padding = 1)
        )

        self.branch3 = nn.Sequential(
            conv_block(in_channels, red_5x5, kernel_size = 1),
            conv_block(red_5x5, out_5x5, kernel_size = 5,  padding = 2)
        )

        self.branch4 = nn.Sequential(
            nn.MaxPool2d(kernel_size=3, stride = 1, padding = 1),
            conv_block(in_channels, out_1x1pool, kernel_size = 1)
        )

    def forward(self, x):
        # N x filters x 28 x 28
        return torch.cat([self.branch1(x), self.branch2(x), self.branch3(x), self.branch4(x)], 1)



class conv_block(nn.Module):
    def __init__(self, in_channels, out_channels, **kwargs):
        super(conv_block, self).__init__()
        self.relu = nn.ReLU()
        self.conv = nn.Conv2d(in_channels = in_channels, out_channels = out_channels, **kwargs)
                            # kernel_size = (1, 1), (3, 3), (5,5 )
        self.batchnorm = nn.BatchNorm2d(out_channels)

    def forward(self, x):
        return self.relu(self.batchnorm(self.conv(x)))
```

```python
device = "cuda:0" if torch.cuda.is_available() else "cpu"
model = GoogLeNet(num_classes= 10).to(device)
optimizer = torch.optim.Adam(model.parameters(), lr = 1e-3)
criterion = nn.CrossEntropyLoss()

model, optimizer, _ = training_loop(model, criterion, optimizer, train_loader, valid_loader, 15, device)
```

# Reference

Going Deeper with Convolutions

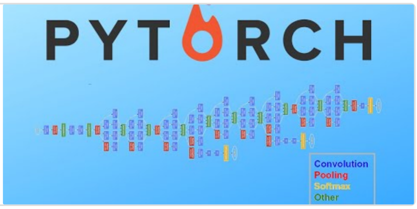
We propose a deep convolutional neural network architecture codenamed "Inception", which was responsible for setting the new state of the art for classification and detection in the ImageNet Large-Scale Visual Recognition Challenge 2014 (ILSVRC 2014). The

https://arxiv.org/abs/1409.4842

Pytorch GoogLeNet / InceptionNet implementation from scratch

In this video we go through how to code the GoogLeNet or InceptionNet from the original paper in Pytorch. I explain how the network works in the first couple...

https://youtu.be/uQc4Fs7yx5I




GoogLeNet (Going deeper with convolutions) 논문 리뷰

이번에는 ILSVRC 2014에서 VGGNet을 제치고 1등을 차지한 GoogLeNet을 다뤄보려 한다. 연구팀 대부분이 Google 직원이어서 아마 이름을 GoogLeNet으로 하지 않았나 싶다. 그럼 대체 왜 Google 팀에서는 이 구조의 이름을 인셉션이라 지었는지, VGGNet과는 어떤 점에서 다

https://phil-baek.tistory.com/entry/3-GoogLeNet-Going-deeper-with-convolutions-%EB%85%BC%EB%AC%B8-%EB%A6%AC%EB%B7%B0