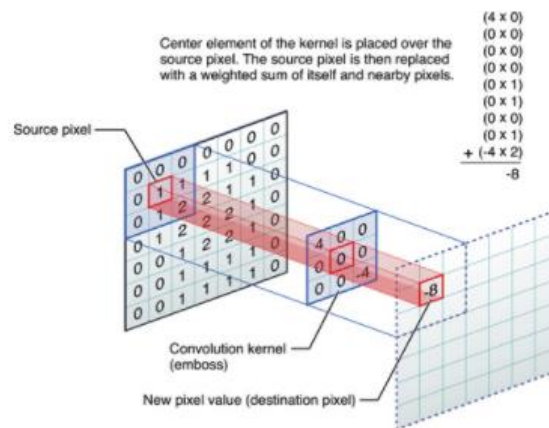


Non-local neural networks

- Reading the paper in terms of Non-local neural networks -

1. Introduction

Convolution은 엄밀하게 말하자면 local operator에 속한다. 연산시에 kernel의 크기만큼만 특징을 추출할 수 있기 때문이다.



[그림1] Convolution

따라서 CNN의 경우 넓은 receptive field를 확보하기 위해(이미지의 더 넓은 부분을 보기 위해) convolution 연산을 여러 차례 겹쳐 쌓거나, pooling을 통해 이미지 자체를 요약해서 특징을 추출하기도 합니다. 하지만 이러한 방식은 receptive field를 일정 수준 확보할 수 있지만, 연산량 대비 효율의 측면에서 좋은 방식은 아닙니다. 이를 극복하고자 본 논문에서는 기존의 convolution의 local한 특성을 보완하기 위해, 한번의 연산이 영상 전체 영역을 대상으로 하는 non-local한 연산을 제안합니다. Non-local block이라고 정의한 이 레이어는 기존의 CNN에 손쉽게 적용 가능하고 이미지나, 비디오 분류 문제에서 큰 성능 향상을 보였습니다. Non-local block에 대한 설명에 앞서, 본 논문 아이디어의 토대가 된 non-local means filter라는 노이즈 필터에 대하여 먼저 간단하게 알아보겠습니다.

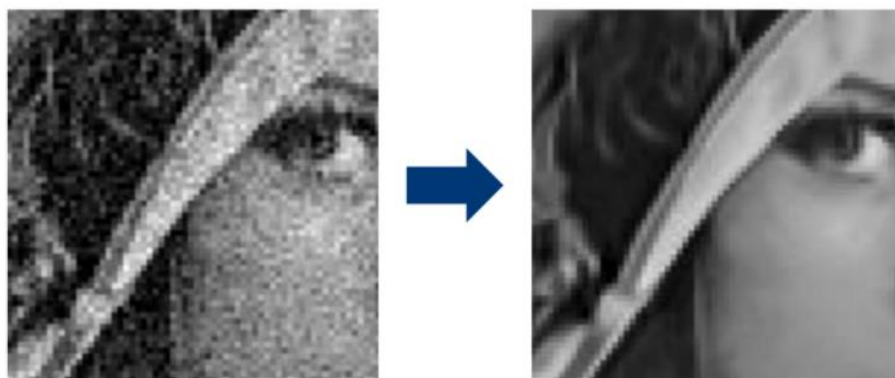
2. Background

Computer vision에서 이미지의 노이즈 제거 문제는 이미지 전처리 단계에서 굉장히 중요한 부분을 차지하고 있으며, 다양한 방법들이 제안되었습니다. 그 중에서도 non-local mean filter(NLM filter)는 노이즈 제거에서 큰 효과를 보였습니다. NLM filter는 노이즈를 제거하기 위해 한 장의 이미지 내에서 유사한 영역을 찾아 평균을 취해주는 방식을 이용해 denoising을 진행합니다.



[그림2] NLM Filter 작동 원리

위의 그림은 NLM Filter의 작동 방식을 직관적으로 보여줍니다. 이미지에서 p 지점의 픽셀을 denoising을 하고 싶다고 했을 때, 사각형 형태의 주변 픽셀을 포함하는 bounding box를 생성합니다. $q1, q2, q3$ 는 p 를 제외한 이미지 내의 픽셀로 정의됩니다. $q1, q2, q3$ 에도 마찬가지로 bounding box를 생성하고 p 의 bounding box와 $q1, q2, q3$ 의 bounding box와의 유사도를 계산합니다. 유사도는 Euclidean 거리를 사용하며, gaussian 커널에 mapping되어 최종 가중치를 산출하게 됩니다. 위 그림에서 보듯이 $p1, p2$ 의 bounding box는 p 의 bounding box와 유사하지만, $q3$ 의 bounding box는 유사하지 않습니다. 따라서 p 픽셀은 $0.05 * q1 + 0.05 * q2 + 0.00001 * q3$ 와 같은 방식으로 픽셀이 계산됩니다. 즉, p 픽셀의 denoising 값은 이미지 내에 존재하는 모든 픽셀의 가중치 선형결합을 통해 생성되게 됩니다. 만약 이미지의 픽셀 개수가 224×224 라면 50176번의 연산이 진행되어야 합니다.

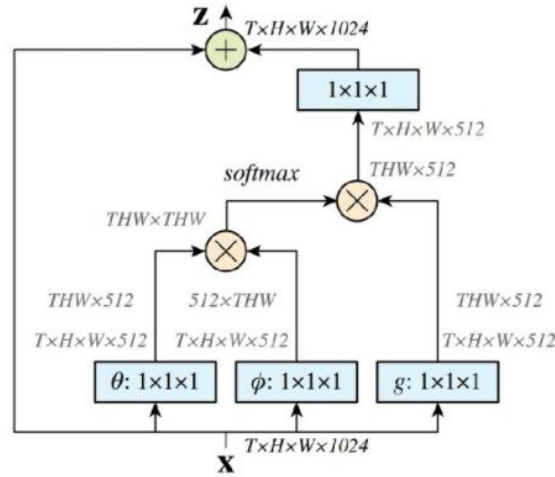


[그림3] NLM Filter 적용 결과

NLM filter는 연산량이 많지만, 좋은 성능을 가지고 있기 때문에 많이 응용되고 있습니다. NLM이라는 이름에서 드러나듯이 다른 필터들과 가장 큰 차이점은 영상 주변부(local)만을 이용한 것이 아니라 영상 전체 영역(non-local)을 활용하는 필터라는 점입니다.

3. Proposed Method

NLM Filter의 핵심은 이미지 전체 영역, 즉 non-local한 영역에서, 서로 간의 유사도를 측정하고 이를 활용하는데 있습니다. 우리가 많이 사용하는 기존의 CNN의 경우는 엄밀히 보면 local한 특성만을 볼 수 있는 구조입니다. 본 연구의 시작은 non-local operator를 네트워크에 추가하면, 이전에는 인식하지 못하던 영상의 새로운 특성을 네트워크가 학습할 수 있을 것이라는 기대에서 출발합니다. 논문의 저자는 NLM Filter의 아이디어를 차용해서 non-local block이라는 구조를 제안했습니다.



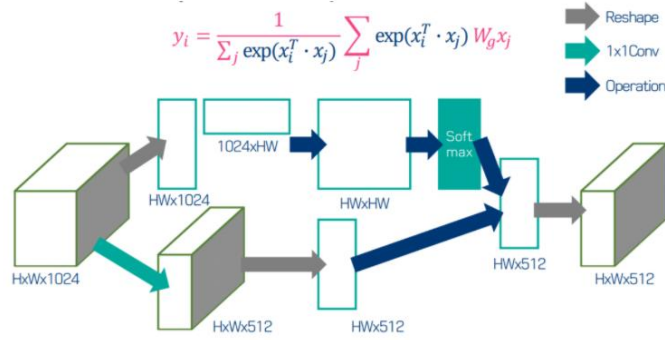
[그림4] Non local block

$$y_i = \frac{1}{\mathcal{C}(\mathbf{x})} \sum_{\forall j} f(\mathbf{x}_i, \mathbf{x}_j) g(\mathbf{x}_j).$$

$$f(\mathbf{x}_i, \mathbf{x}_j) = e^{\theta(\mathbf{x}_i)^T \phi(\mathbf{x}_j)}$$

본 논문에서는 비디오 데이터를 사용하였기 때문에 T라는 시간 축이 포함되어 있지만, 이미지로 생각할 경우, 단순히 T가 없다고 생각하면 됩니다. 수식과 그림을 통해 전체적인 구조에 대해 살펴보자면 Non-local block의 수식에서 x_i 의 위치에 대한 결과를 만들기 위해 x_i 와 그 외 각 모든 영역 x_j 와의 관계를 $f(x_i, x_j)$ 를 통해 계산합니다. θ, ϕ 는 채널을 줄이기 위한 1x1 convolution을 수행합니다. 1x1 convolution을 통과한 값들은 관계를 계산하기 위해서 $HW \times C$ 와 $C \times HW$ 의 dot product 연산을 수행합니다. 함수 f 는 NLM filter에서 유사도를 계산하는 것과 같은 개념으로 생각할 수 있습니다. 유사도는 embedded gaussian 커널에 mapping되어 유사도를 산출하게 됩니다. $g(x_j)$ 는 NLM filter로 생각하면 x_i 와 비교할 pixel에 해당하는 값인데, 네트워크 연산에 적합하게 만들기 위하여 W_g 라는 weight matrix에 곱하여 생성하게 됩니다.

Similarity를 측정하는 함수 f 에 어떤 커널을 사용할 것인가는 본 논문에서도 중요하게 다루고 있는 문제인데 결론적으로 커널의 종류보다는 similarity를 측정하는 행위 자체가 중요한 것으로 이야기하고 있습니다.



[그림5] Non local block 행렬 연산 예시

4. Experiments(Code Review)

(1) Non-local block

Non-local block은 행렬곱 연산과 point-wise plus 연산으로 구성되기 때문에 __init__과 forward만으로도 좌측 그림의 흐름대로 구성할 수 있습니다.

```
import torch
from torch import nn
from torch.nn import functional as F

class NonLocalBlockND(nn.Module):
    def __init__(self, in_channels, inter_channels=None, dimension=3, sub_sample=True, bn_layer=True):
        super(NonLocalBlockND, self).__init__()
        assert dimension in [1, 2, 3]
        self.dimension = dimension
        self.sub_sample = sub_sample
        self.in_channels = in_channels
        self.inter_channels = inter_channels

        if self.inter_channels is None:
            self.inter_channels = in_channels // 2
            if self.inter_channels == 0:
                self.inter_channels = 1

        if dimension == 3:
            conv_nd = nn.Conv3d
            max_pool_layer = nn.MaxPool3d(kernel_size=(1, 2, 2))
            bn = nn.BatchNorm3d
        elif dimension == 2:
            conv_nd = nn.Conv2d
            max_pool_layer = nn.MaxPool2d(kernel_size=(2, 2))
            bn = nn.BatchNorm2d
        else:
            conv_nd = nn.Conv1d
            max_pool_layer = nn.MaxPool1d(kernel_size=(2))
            bn = nn.BatchNorm1d

        self.g = conv_nd(in_channels=self.in_channels, out_channels=self.inter_channels,
                        kernel_size=1, stride=1, padding=0)
        if bn_layer:
            self.W = nn.Sequential(
                conv_nd(in_channels=self.in_channels, out_channels=self.in_channels,
                    kernel_size=1, stride=1, padding=0),
                bn(self.in_channels)
            )
            nn.init.constant_(self.W[1].weight, 0)
            nn.init.constant_(self.W[1].bias, 0)
        else:
            self.W = conv_nd(in_channels=self.in_channels, out_channels=self.in_channels,
                kernel_size=1, stride=1, padding=0)
            nn.init.constant_(self.W.weight, 0)
            nn.init.constant_(self.W.bias, 0)

        self.theta = conv_nd(in_channels=self.in_channels, out_channels=self.inter_channels,
            kernel_size=1, stride=1, padding=0)
        self.phi = conv_nd(in_channels=self.in_channels, out_channels=self.inter_channels,
            kernel_size=1, stride=1, padding=0)

        if sub_sample:
            self.g = nn.Sequential(self.g, max_pool_layer)
            self.phi = nn.Sequential(self.phi, max_pool_layer)

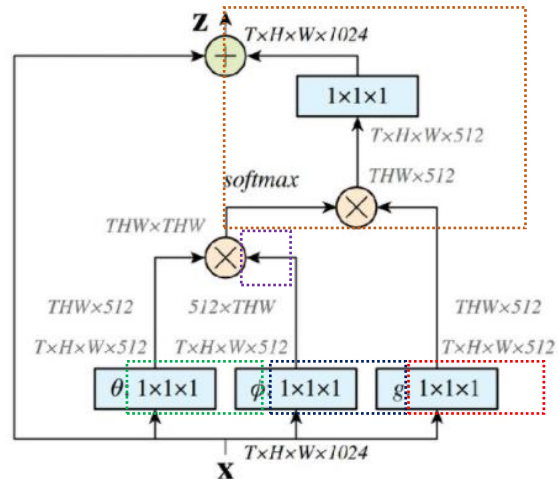
    def forward(self, x, return_nl_map=False):
        batch_size = x.size(0)

        g_x = self.g(x).view(batch_size, self.inter_channels, -1)
        g_x = g_x.permute(0, 2, 1)

        theta_x = self.theta(x).view(batch_size, self.inter_channels, -1)
        theta_x = theta_x.permute(0, 2, 1)
        phi_x = self.phi(x).view(batch_size, self.inter_channels, -1)
        f = torch.matmul(theta_x, phi_x)
        f_div_C = F.softmax(f, dim=-1)

        y = torch.matmul(g_x, f_div_C)
        y = y.permute(0, 2, 1).contiguous()
        y = y.view(batch_size, self.inter_channels, *x.size()[2:])
        W_y = self.W(y)
        z = W_y + x

        if return_nl_map:
            return z, f_div_C
        return z
```



(2) Network

네트워크는 총 3개의 convolution과 2개의 non-local block으로 구성되어 있습니다. CNN이 해당 이미지를 분류할 때 추론한 히트맵과 Non local block이 이미지에서 가장 중요하다고 생각한 부분을 나타내는 구역의 차이를 시각화하기 위해 forward_with_nl_map을 활용하였습니다.

```
class Network(nn.Module):
    def __init__(self):
        super(Network, self).__init__()

        self.conv_1 = nn.Sequential(
            nn.Conv2d(in_channels=1, out_channels=32, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(32),
            nn.ReLU(),
            nn.MaxPool2d(2),
        )

        self.nl_1 = NONLocalBlock2D(in_channels=32)
        self.conv_2 = nn.Sequential(
            nn.Conv2d(in_channels=32, out_channels=64, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(64),
            nn.ReLU(),
            nn.MaxPool2d(2),
        )

        self.nl_2 = NONLocalBlock2D(in_channels=64)
        self.conv_3 = nn.Sequential(
            nn.Conv2d(in_channels=64, out_channels=128, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(128),
            nn.ReLU(),
            nn.MaxPool2d(2),
        )

        self.fc = nn.Sequential(
            nn.Linear(in_features=128*3*3, out_features=256),
            nn.ReLU(),
            nn.Dropout(0.5),
            nn.Linear(in_features=256, out_features=10)
        )

    def forward_with_nl_map(self, x):
        batch_size = x.size(0)

        feature_1 = self.conv_1(x)
        nl_feature_1, nl_map_1 = self.nl_1(feature_1, return_nl_map=True)

        feature_2 = self.conv_2(nl_feature_1)
        nl_feature_2, nl_map_2 = self.nl_2(feature_2, return_nl_map=True)

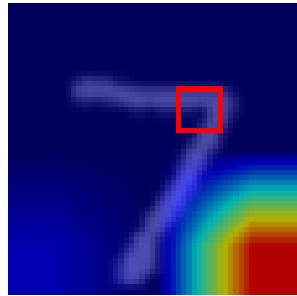
        output = self.conv_3(nl_feature_2).view(batch_size, -1)
        output = self.fc(output)

        return output, [nl_map_1, nl_map_2]

if __name__ == '__main__':
    import torch

    img = torch.randn(3, 1, 28, 28)
    net = Network()
    out = net(img)
    print(out.size())
```

(3) 결과 확인



[그림6]두 번째 Non-local layer에 대하여 오분류한 필기체 7 이미지의 시각화 결과

MNIST 데이터에 대해서 네트워크의 epoch을 약 100번정도 실행하였습니다. Test 정확도는 99.39%를 기록하였으며, training 정확도는 99.91%를 기록하였습니다. 중요한 점은 실험에서 epoch 1번만으로도 Test 정확도 96%, training 정확도 98.36%를 기록했다는 점입니다. Non-local block이 CNN의 이미지 분류에 큰 도움을 준다는 것을 알 수 있는 점입니다.

위의 그림은 오분류한 7의 이미지에 대해서 Non-local block이 이미지에서 가장 중요하다고 판단한 부분(빨간색 사각형 box)과 CNN이 해당 이미지를 7이라고 추론한 영역(히트맵)에 대해 시각화한 결과입니다. CNN은 위의 이미지를 판단할 때 전혀 다른 부분을 보고 판단하였기에 7이라는 클래스로 분류를 해내지 못하였지만, non-local block의 결과에서는 7의 꺾이는 부분을 확인한 것을 볼 수 있습니다. 이 결과는 convolution의 local하게 이미지를 본 영역과 다른 시각을 non-local block은 보여준다는 것을 알 수 있습니다.