AutoAugment: Learning Augmentation Policies from Data

https://arxiv.org/abs/1805.09501v1

학과 : 데이터사이언스융합전공
이름 : 박민승 / 2019310065

Key insight : Create a search space of data augmentation policies
강화학습을 통해 어떠한 data augmentation 방법이 가장 좋을지에 대한 탐색이 이루어진다.
논문에는 총 16개의 operations들을 사용했다고 명시되어 있으며, 이들 중 랜덤하게 선택(mini batch마다)하고 최적의 policy를 찾는다.

논문에서는 시간이 너무 오래 걸려 데이터의 일부(Cifar10 : 4000장, SVHN : 1000장, ImageNet : 6000장)만을 활용하여 강화학습을 진행하였다.

로컬 환경이 좋지 않아 강화학습을 진행하기에 문제가 있어, 3개의 데이터를 토대로 나온 policy들을 우리가 가진 화재 데이터에 적용하는 방식으로 코드를 진행하였다.

Github에 구현된 operations들은 총 14개(ops.py)이며, 미구현된 것은 2가지이다.
The operations we searched over are ShearX/Y, TranslateX/Y,Rotate, AutoContrast, Invert, Equalize, Solarize, Posterize, Contrast, Color, Brightness, Sharpness,Cutout [25], Sample Pairing. -> 총 16개의 operations임을 확인할 수 있다.

<Ops.py>

[미구현] : Cuttoff, SampleParing

1. **Cutoff** : we first apply the baseline augmentation providedby the existing baseline methods, then apply the AutoAugment policy, then apply Cutout. We did not optimize the Cutout region size.

14개의 operation 중 하나를 수행한 후 cutoff를 진행한다. Region size를 optimize하지않고 random하게 진행하였다고 한다. 논문에서는 무작위로 선정된 operation이 cutoff인 경우 다른 operation을 고른 후 cutoff를 진행하였다고 하는데, cutoff의 영향인지 또는 선정된 operation이 큰 영향을 미친것인지 파악할 수 없다. 이러한 점을 어떻게 고려하였는지는 명시되어 있지 않다.

2. **SamplePairing** : Linearly add the image with another image.

무작위로 선택된 이미지를 다른 이미지와 합치고 원래 label로 유지하였다고 하는데 여러 label이 겹친 경우, 어떻게 해결하였는지 명시되어 있지 않다.

[구현] : 14개의 operations

공통점 :
모두 PIL library를 통해 구현하였다.
논문에서 magnitude에 대해 10가지 값으로 진행
We discretize the range of magnitudes into 10 values (uniform spacing) 라고 명시
구현에서는 -1, 1 고르고 1~10을 곱하는 식으로 진행한다.

[PIL.Image] : Shear X/Y, Translate X/Y, Rotate 적용

```python
class ShearX(object):
    def __init__(self, fillcolor=(128, 128, 128)):
        self.fillcolor = fillcolor

    def __call__(self, x, magnitude):
        return x.transform(
            x.size, Image.AFFINE, (1, magnitude * random.choice([-1, 1]), 0, 0, 1, 0),
            Image.BICUBIC, fillcolor=self.fillcolor)

class ShearY(object):
    def __init__(self, fillcolor=(128, 128, 128)):
        self.fillcolor = fillcolor

    def __call__(self, x, magnitude):
        return x.transform(
            x.size, Image.AFFINE, (1, 0, 0, magnitude * random.choice([-1, 1]), 1, 0),
            Image.BICUBIC, fillcolor=self.fillcolor)
```

```python
class TranslateX(object):
    def __init__(self, fillcolor=(128, 128, 128)):
        self.fillcolor = fillcolor

    def __call__(self, x, magnitude):
        return x.transform(
            x.size, Image.AFFINE, (1, 0, magnitude * x.size[0] * random.choice([-1, 1]), 0, 1, 0),
            fillcolor=self.fillcolor)


class TranslateY(object):
    def __init__(self, fillcolor=(128, 128, 128)):
        self.fillcolor = fillcolor

    def __call__(self, x, magnitude):
        return x.transform(
            x.size, Image.AFFINE, (1, 0, 0, 0, 1, magnitude * x.size[1] * random.choice([-1, 1])),
            fillcolor=self.fillcolor)
```

Image.AFFINE(a,b,tx, c,d,ty)
: (a, b) 및 (c, d)는 각각 X 및 Y 축에서의 이미지의 크기 조절을 의미, tx,ty는 각 축으로의 이동을 의미한다.
Image.BICUBIC
: Image interpolation 기법으로, 16개의 주변 픽셀을 활용하여 가중치를 계산한 후 새로운 픽셀에 도입한다.

```python
class Rotate(object):
    # from https://stackoverflow.com/questions/
    # 5252170/specify-image-filling-color-when-rotating-in-python-with-pil-and-setting-expand
    def __call__(self, x, magnitude):
        rot = x.convert("RGBA").rotate(magnitude * random.choice([-1, 1]))
        return Image.composite(rot, Image.new("RGBA", rot.size, (128,) * 4), rot).convert(x.mode)
```

x.convert('RGBA')로 r,g,b,a(투명)으로 변경 후 rotate시킨다. 이후 투명한 rotate전 사이즈에 rotate시킨 이미지를 합성한다.

[PIL.ImageEnhance] : Color, Contrast, Sharpness, Brightness적용

```python
class Color(object):
    def __call__(self, x, magnitude):
        return ImageEnhance.Color(x).enhance(1 + magnitude * random.choice([-1, 1]))

class Contrast(object):
    def __call__(self, x, magnitude):
        return ImageEnhance.Contrast(x).enhance(1 + magnitude * random.choice([-1, 1]))


class Sharpness(object):
    def __call__(self, x, magnitude):
        return ImageEnhance.Sharpness(x).enhance(1 + magnitude * random.choice([-1, 1]))


class Brightness(object):
    def __call__(self, x, magnitude):
        return ImageEnhance.Brightness(x).enhance(1 + magnitude * random.choice([-1, 1]))
```

<Color> 0: 흑백, 1: 원본 , 1이 넘어가면 색상이 더해진다.
<Contrast> 0 에 가까울 수록 대비가 없는 회색 이미지에 가깝게 되고 1 이 원본 값이되고 1이 넘어가면 대비가 강해지는 결과
<Brightness> 0 에 가까울 수록 그냥 검정 이미지에 가깝게 되고 1 이 원본 값이되고 1이 넘어가면 밝기가 강해지는 결과
<Sharpness> 0 에 가까울 수록 이미지는 흐릿한 이미지에 가깝게 되고 1 이 원본 값이고 1이 넘어가면 원본에 비해 선명도가 강해지는 결과

[PIL.ImageOps] : Invert, Posterize, Solarize, Autocontrast, Equalize 활용

```python
class Posterize(object):
    def __call__(self, x, magnitude):
        return ImageOps.posterize(x, magnitude)


class Solarize(object):
    def __call__(self, x, magnitude):
        return ImageOps.solarize(x, magnitude)


class AutoContrast(object):
    def __call__(self, x, magnitude):
        return ImageOps.autocontrast(x)


class Equalize(object):
    def __call__(self, x, magnitude):
        return ImageOps.equalize(x)

class Invert(object):
    def __call__(self, x, magnitude):
        return ImageOps.invert(x)
```

<Invert> 색상 반전
<Posterize> 색상 제한 magnitude만큼의 색상으로 제한됨
<Solarize> magnitude이상(threshold)의 모든 픽셀값을 반전
<Autocontrast> 이미지 contrast 자동으로 설정
<Equalize> This function applies a non-linear mapping to the input image,
in order to create a uniform distribution of grayscale values in the output image

<DataAugment.py>

ops.py에서 각 operations들을 import 하여 가져온다
이후 각 operation마다 magnitude가 다르기에 range를 설정해준다.
이후 cifar10, SVHN, ImageNet에 위의 함수들을 활용하여 policy를 설정한다.

```python
class SubPolicy(object):
    def __init__(self, p1, operation1, magnitude_idx1, p2, operation2, magnitude_idx2, fillcolor=(128, 128, 128)):
        ranges = {
            "shearX": np.linspace(0, 0.3, 10),
            "shearY": np.linspace(0, 0.3, 10),
            "translateX": np.linspace(0, 150 / 331, 10),
            "translateY": np.linspace(0, 150 / 331, 10),
            "rotate": np.linspace(0, 30, 10),
            "color": np.linspace(0.0, 0.9, 10),
            "posterize": np.round(np.linspace(8, 4, 10), 0).astype(int),
            "solarize": np.linspace(256, 0, 10),
            "contrast": np.linspace(0.0, 0.9, 10),
            "sharpness": np.linspace(0.0, 0.9, 10),
            "brightness": np.linspace(0.0, 0.9, 10),
            "autocontrast": [0] * 10,
            "equalize": [0] * 10,
            "invert": [0] * 10
        }
```

예를 들어 rotate의 경우 논문에서 -30도~30도라고 명시되어있으며, ranges에서 범위를 설정해주고, ops.py에서 (-1~1)과 magnitude를 곱해주는 식으로 진행하였다.
이후 Cifrar10, SVHN, ImageNet class에서 subpolicy class를 활용하여 총 25개의 policies 들을 생성하게 된다.

<Result>

원본 이미지는 다음과 같다.

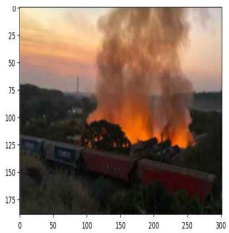

그림 8 ImageNet 활용

그림 9 Cifar10 활용

그림 10 SVHN 활용

코드에는 총 25개의 operations들이 하나의 리스트 안에 존재한다.
ImageNet, Cifar10, SVHN의 클래스마다 각기 다른 operations들이 존재하는데 이들을 활용하여 약간의 변형을 주고, data loader에 추가한다.
-> 논문에 Best 5 policies to form a single policy with 25 sub-policies라고 명시.

이미지는 25개의 operations들 중 랜덤하게 선택하여 이미지를 보이도록 하였다.

```
self.policies = [
    SubPolicy(0.9, "shearX", 4, 0.2, "invert", 3, fillcolor),
    SubPolicy(0.9, "shearY", 8, 0.7, "invert", 5, fillcolor),
    SubPolicy(0.6, "equalize", 5, 0.6, "solarize", 6, fillcolor),
    SubPolicy(0.9, "invert", 3, 0.6, "equalize", 3, fillcolor),
    SubPolicy(0.6, "equalize", 1, 0.9, "rotate", 3, fillcolor),

    SubPolicy(0.9, "shearX", 4, 0.8, "autocontrast", 3, fillcolor),
    SubPolicy(0.9, "shearY", 8, 0.4, "invert", 5, fillcolor),
    SubPolicy(0.9, "shearY", 5, 0.2, "solarize", 6, fillcolor),
    SubPolicy(0.9, "invert", 6, 0.8, "autocontrast", 1, fillcolor),
    SubPolicy(0.6, "equalize", 3, 0.9, "rotate", 3, fillcolor),

    SubPolicy(0.9, "shearX", 4, 0.3, "solarize", 3, fillcolor),
    SubPolicy(0.8, "shearY", 8, 0.7, "invert", 4, fillcolor),
    SubPolicy(0.9, "equalize", 5, 0.6, "translateY", 6, fillcolor),
    SubPolicy(0.9, "invert", 4, 0.6, "equalize", 7, fillcolor),
    SubPolicy(0.3, "contrast", 3, 0.8, "rotate", 4, fillcolor),

    SubPolicy(0.8, "invert", 5, 0.0, "translateY", 2, fillcolor),
    SubPolicy(0.7, "shearY", 6, 0.4, "solarize", 8, fillcolor),
    SubPolicy(0.6, "invert", 4, 0.8, "rotate", 4, fillcolor),
    SubPolicy(0.3, "shearY", 7, 0.9, "translateX", 3, fillcolor),
    SubPolicy(0.1, "shearX", 6, 0.6, "invert", 5, fillcolor),

    SubPolicy(0.7, "solarize", 2, 0.6, "translateY", 7, fillcolor),
    SubPolicy(0.8, "shearY", 4, 0.8, "invert", 8, fillcolor),
    SubPolicy(0.7, "shearX", 9, 0.8, "translateY", 3, fillcolor),
    SubPolicy(0.8, "shearY", 5, 0.7, "autocontrast", 3, fillcolor),
    SubPolicy(0.7, "shearX", 2, 0.1, "invert", 5, fillcolor)
]
```

5x5씩 총 25개의 policies들이 모여있는 상황

최종적으로 선택된 policy를 우리가 가진 이미지에 적용하여 새로운 이미지들을 생성하고, 이 이미지들을 data loader에 넣어줌으로 더 많은 데이터를 확보할 수 있게 된다.