|  |
| --- |
|  |
| **흑백사진 컬러화 알고리즘의 전반적인 분석**  **-Colourise SG를 중심으로-** |
|  |

**20184887 박지균**

**20183300 윤성빈**

**20186033 임채원**

**20184141 황수인**

**목차**

[1 서론 3](#_Toc27080717)

[1.1 흑백사진 컬러화 알고리즘 3](#_Toc27080718)

[1.2 종류 3](#_Toc27080719)

[2 프로젝트 진행 중 장애사항 3](#_Toc27080720)

[3 Colourise SG 원리 3](#_Toc27080721)

[3.1 알고리즘 분석 4](#_Toc27080722)

[4 Colourise SG 코드 분석 4](#_Toc27080723)

[4.1 모듈 불러오기 4](#_Toc27080724)

[4.2 초기 설정 5](#_Toc27080725)

[4.3 모델 사전학습 6](#_Toc27080726)

[4.4 GAN 사이클 반복시키기 7](#_Toc27080727)

[4.5 생성된 이미지 저장 7](#_Toc27080728)

[4.6 사전 학습된 Critic 7](#_Toc27080729)

[4.7 GAN 수정 8](#_Toc27080730)

[4.8 이미지 컬러화 단계 9](#_Toc27080731)

[5 Colourise SG 실제 사용 결과 10](#_Toc27080732)

[6 Colourise SG 의 문제점 및 개선 방향 12](#_Toc27080733)

[6.1 문제점 12](#_Toc27080734)

[6.2 개선 방향 12](#_Toc27080735)

[7 결론 13](#_Toc27080736)

[8 출처 14](#_Toc27080737)

[8.1 개발자 블로그 14](#_Toc27080738)

[8.2 그 외 14](#_Toc27080739)

# 서론

## 흑백사진 컬러화 알고리즘

흑백사진의 컬러화란, 말 그대로 흑백사진으로 찍힌 대상의 원래 색을 유추해 색을 입히는 것을 말한다. 과거에는 사람이 수동적으로 흑백사진을 컬러화하려면 많은 양의 역사적, 지리적, 문화적 문맥의 사진을 조사하고 상황에 맞는 적당한 색을 학습하고, 이후 포토샵 등을 사용해서 컬러화해야 했다. 하지만 지금은 인공지능이 그를 대체하기 시작했다. 위의 과정은 굉장히 많은 시간이 소요되는데, 이를 인공지능이 대체하면서 길어도 1분 안에 모든 과정이 끝나게 됐다.

## 종류

대표적으로 Algorithmia, Let There be Color, Colourise SG 등이 있다. 모든 프로젝트를 기반으로 흑백사진을 컬러화해본 결과, Colourise SG가 제일 본래 사진과 유사하게 컬러화를 했다. 이에 따라 본 보고서에는 Colourise SG를 중심으로 흑백사진의 컬러화 알고리즘을 분석했다.

# 프로젝트 진행 중 장애사항

초기에는 Algorithmia에 올라온 Colorization 모델을 이용하여 분석을 진행하려 했지만, 해당 모델의 핵심이 되는 부분이 caffe 프레임워크 기반으로 작성되어 있었는데, 이를 구현하기 위해 CUDA 및 caffe를 다운로드 하고 실행하려 했지만 많은 오류들이 발생했고, 구글링을 통해서 일부 오류는 잡아낼 수 있었으나 끝내 마지막 오류를 잡아내지 못해 실행을 하지 못했다.

이후 Algorithmia의 개선 버전인 Colourise SG 프로젝트를 찾아내어 분석을 진행하려 했으나, Colourise SG 는 입출력을 할 수 있는 사이트는 공개되어 있지만 실제 실행 코드는 비공개 되어 있어 개발자인 Preston Lim 이 개발과정에서 참고한 모델인 GAN 관련 프로젝트인 DeOldfiy에 대해 코드 분석을 진행하게 되었다.

# Colourise SG 원리

Colourise SG는 기존에 있던 컬러화 알고리즘인 Algorithmia의 문제점을 일부 개선해 등장한 컬러화 알고리즘이다. 이들은 Algorithmia가 학습에 사용하는 이미지 데이터 세트가 스탠포드, 프린스턴의 연구원들이 주로 사용하는, 즉 주로 서양 쪽에 치중되는 ImageNet의 이미지를 사용했기 때문에 동양 쪽의 사진의 컬러화엔 유용하지 않다는 결론을 내렸다. 이에 그들은 다양하고 포괄적인 이미지 데이터 세트를 사용하고 흑백 – 컬러화 이미지와 실제 컬러 사진 간의 유사성을 높이기 위해 GAN 모델을 사용하여 프로젝트를 구성했다.

## 알고리즘 분석

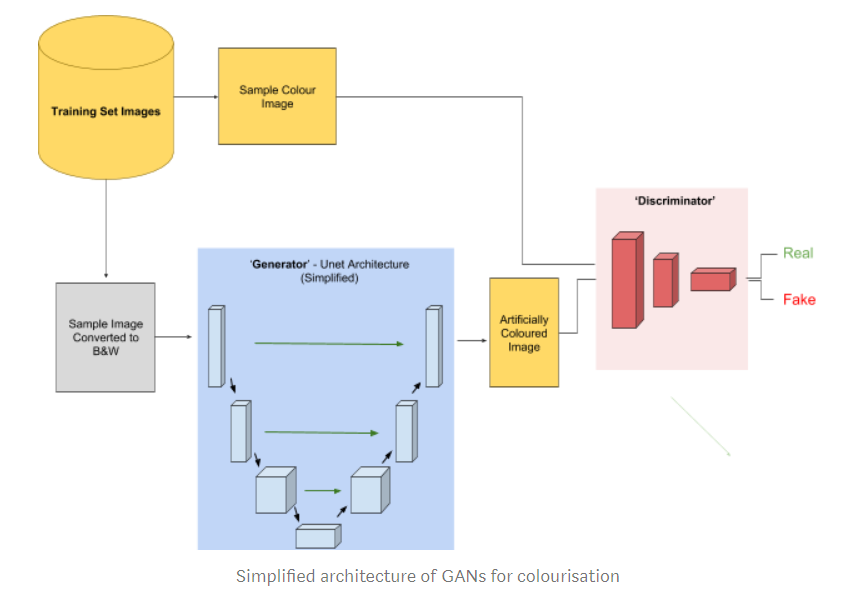


그림 ‑ 컬러화를 위한 GAN 구조

Colourise SG는 앞서 언급한 대로 GAN 모델을 사용하여 흑백사진을 컬러화한다. 아래 사진에서 확인할 수 있듯, 학습을 위해 샘플 이미지와, 흑백화한 샘플 이미지를 준비한다. 흑백 샘플 이미지를 U-net 구조의 생성자 모델에 넣어 색을 입힌 뒤, 구별자 모델에서 샘플 이미지와 컬러화한 흑백 샘플 이미지를 구별한다. 이 때 Colourise SG는 최대한 샘플 이미지에 가까운 색을 입히는 생성자 모델이 되도록 학습하고, 이를 위해 구별자 모델 또한 생성자 모델이 만들어 낸 컬러 이미지와 실제 이미지를 최대한 잘 구별할 수 있도록 학습한다. 그렇게 하면 결론적으로, 생성자 모델은 실제 이미지에 가까운 컬러화 이미지를 생성할 수 있기 때문이다.

# Colourise SG 코드 분석

## 모듈 불러오기



그림 ‑ os 모듈을 불러와서 GPU 사용을 위한 환경 구축



그림 ‑ import fastai

fastai.vision 모듈을 불러와서 fastai 라이브러리의 데이터셋을 정의하고 모델을 훈련시키는데 필요한 모든 기능을 사용할 수 있도록 한다. fastai.callbacks.tensorboard 모듈을 불러와서 tensorboard magic command를 로드하여 Jupyter Notebook에 내장된 tensorboard 를 표시한다. fastai.vision.gan 모듈을 통해 GAN을 만들어 내기 위해 필요한 기능들을 불러온다. 또한 이번 컬러화를 위해 따로 만들어 놓은 deoldify 폴더 내의 generator, critics, dataset, loss, save 모듈들을 불러온다. PIL 모듈 내에서 이미지를 표현하는 Image 클래스, 2D 그래픽을 보여주는 ImageDraw 클래스, bitmap font를 저장하는 ImageFont 클래스, 이미지를 열거나 저장하는 것을 도와주는 ImageFile 클래스를 불러온다.

## 초기 설정

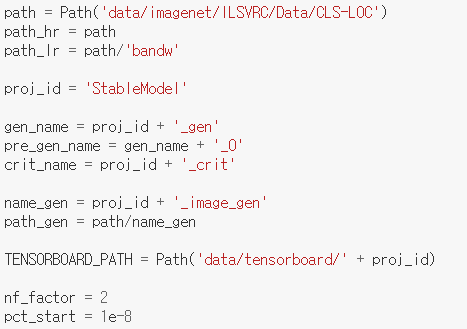
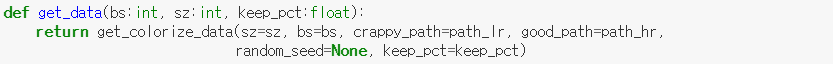


그림 ‑

ImageNet 데이터셋에서 가져올 이미지들의 경로 설정 과 쓰일 변수들의 이름을 설정한다.



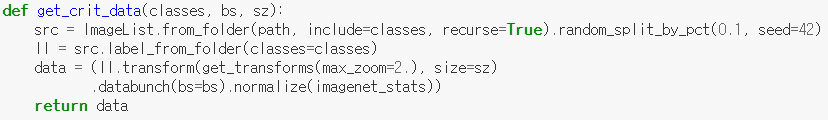
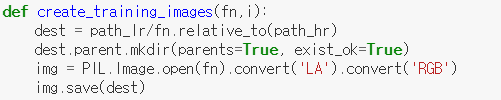


그림 ‑ 컬러화에 쓰일 데이터와 분별기에 쓰일 이미지를 가져온다



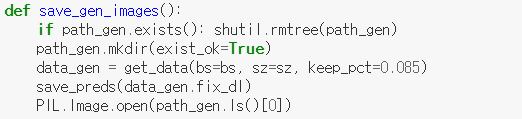


그림 ‑ 학습에 쓰일 이미지, 이미지를 저장하는 함수 생성

## 모델 사전학습

대부분의 학습은 이 단계에서 이루어진다. 기존 훈련을 통해 최대한 생성기를 학습시키려는 것이다.

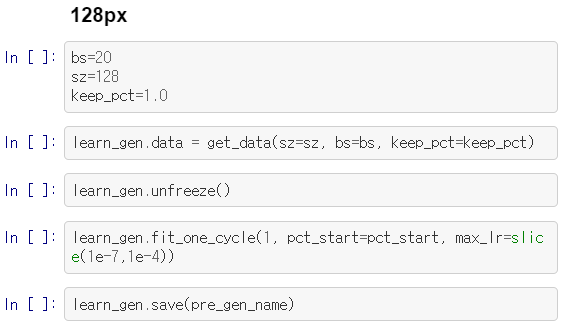


그림 ‑ 이미지의 사이즈를 128로 설정, 쓰일 이미지 데이터들을 불러온다

이 이미지들은 가중치를 수정하지 못하고 그레이디언트 값들은 레이어에 연산되어 있지 않은 상태들이다. 그래서 unfreeze() 함수를 통해 학습 가능한 상태로 만들어 준다.

learn\_gen.fit\_one\_cycle() 함수에서 가장 마지막 레이어만 학습시키는데 이 이유는 과적합에 빠질 수 있기 때문이다. 매개변수 max\_lr을 사용하여 차별화된 학습 속도를 사용할 수 있다. 초기 레이어에서는 조정의 필요성이 많이 없으므로 낮은 학습률을 유지하고, 후반 레이어에서는 조정의 필요성이 많으므로 학습률을 점차 증가시키는 역할을 한다. 여기서 초기 레이어란, 주로 곡선, 선, 모양, 패턴과 같은 저수준 기능을 학습하는 단계이고, 후반 레이어는 주로 현재 데이터 세트의 고급 기능들을 학습하는 단계이다.

마지막으로 learn\_gen.save() 함수를 통해 학습된 레이어들을 저장한다.

## GAN 사이클 반복시키기



그림 ‑

지금껏 가장 좋은 방법은 감소된 결과가 발생할 때까지 몇 번에 걸쳐 아래의 사이클을 반복하는 것이었다. 주기를 반복할 때마다 새 체크포인트가 이전 체크포인트를 덮어쓰지 않도록 해당 old\_checkpoint\_num을 1씩 증가시켜야 한다.

## 생성된 이미지 저장

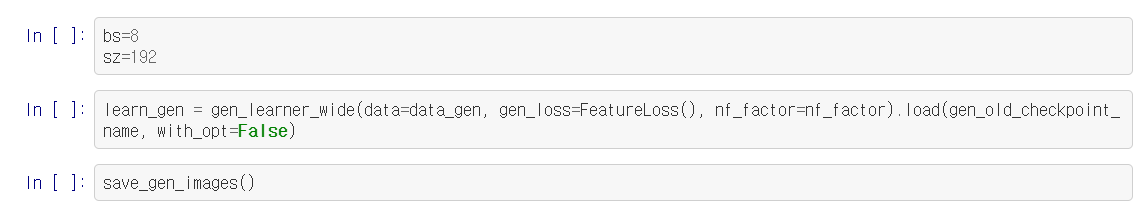


그림 ‑

bs는 8 사이즈는 192로 맞춰놓고 data와 arch를 가지고 새로운 Unet learner를 생성한 후 생성된 이미지를 저장한다.

## 사전 학습된 Critic

처음부터 시작할 때만 Critic의 완전한 사전 학습이 필요하기 때문에 그렇지 않으면, 마무리를 해야 한다.

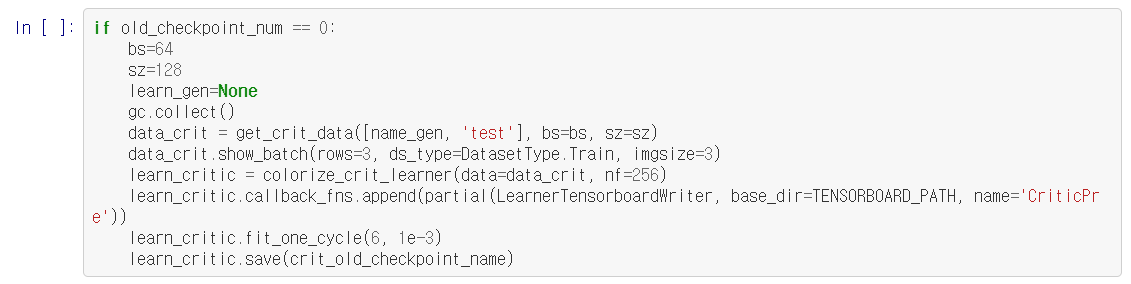


그림 ‑

만약 체크포인트가 0이라면 bs는 64 sz는 128로 맞추고 learner를 생산하지 않는다. 그 후 가비지컬렉션을 수행한다. Data\_crit에 crit 데이터를 받고, 행, dstype, 이미지 사이즈를 표시한다. .Colorize\_crit\_learner 함수를 통해 학습자의 데이터를 받고 critic의 nf를 256으로, metrics에 정확도 값을, loss\_func에 critic손실 값을 넣고 learn\_critic에 return해준다. 그 후 old 체크포인트 name을 저장한다.

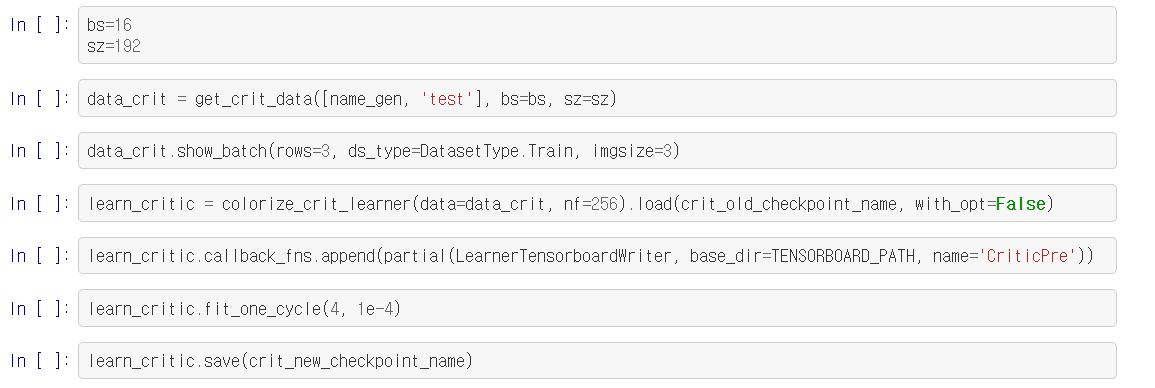


그림 ‑

bs는 16, sz는 192로 맞추고 date\_crit에 ‘test’란 이름의 crit데이터를 가져오고 행, dstype, 이미지 사이즈를 표시한다. learn\_critic에 학습자의 old 체크 포인트를 로드 받는다. Callback 부분을 append 함수를 통해 추가한 다음 가장 마지막 레이어를 학습시키고 새로운 체크포인트를 저장한다.

## GAN 수정



그림 ‑

학습 crit과 학습 생산자를 멈추고 가비지 컬렉션을 수행한다. lr, sz, bs를 각각 2e-5, 192, 5 값으로 정해준다. Data\_crit에 ‘test’라는 crit 데이터를 넣어주고 learn\_crit에 crit 학습자의 새로운 체크 포인트를 저장한다. Learn\_gen에 data\_gen과 gen\_loss를 이용해 Unet 학습자를 만들고 old 체크포인트를 할당한다.

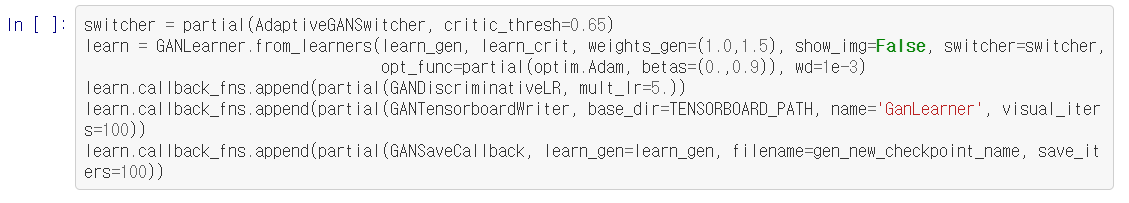


그림 ‑

Switcher에 AdaptiveGANSwitcher와 critic\_thresh의 부분을 할당하고 learn에 GANLearner의 분류된 데이터들을 해석해서 할당한다. GANDiscriminativeLR, GANTensorboardWriter, GANSaveCallback의 Callback 부분을 append 함수를 통해 추가한다.

## 이미지 컬러화 단계

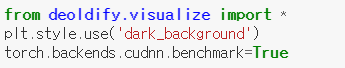
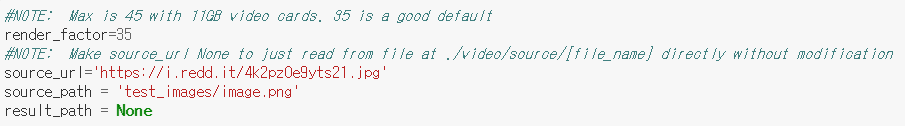


그림 ‑

Deoldify.visulize를 가져온다. 실행화면을 어둡게 바꿔준다. (이 부분은 사람의 기호에 따라 바꿔주면 된다.) 그 다음 cudnn에서 벤치 마크 모드를 활성화한다. 벤치 마크 모드는 네트워크의 입력 크기가 다를 때마다 좋다. cudnn은 특정 구성에 대해 최적의 알고리즘 세트를 찾기 때문에 일반적으로 런타임이 빨라진다.



그림 ‑ Colorizer 실행



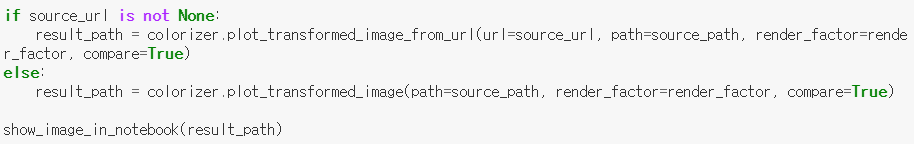


그림 ‑

Render\_factor를 35로 맞춰 놓는다. 35가 가장 Render\_factor란 이미지의 색상 부분이 렌더링 되는 해상도를 결정하는 역할을 한다. Source\_url을 설정해서 이미지의 링크를 연결한다. 자신의 이미지를 사용하고 싶으면 None으로 설정하여도 된다. 그리고 source\_path 이름을 임의로 정하고, result\_path를 None으로 설정해 놓는다.

이 실험이 임의로 정해 놓은 이미지를 사용하지 않을 때와 사용할 때를 구분해서 설정해 놓은 다음 컬러화를 진행한다.



그림 ‑

위의 코드를 사용하여 각 render\_factor에서 이미지의 색깔이 얼마나 잘 설정되는지 평가할 수 있다.

# Colourise SG 실제 사용 결과

왼쪽부터 차례대로 원본 – 흑백화한 원본 – 컬러화한 원본 순이다.



그림 ‑ 사물 사진 컬러화



그림 ‑ 음식 사진 컬러화



그림 ‑ 인물 사진 컬러화



그림 ‑ 풍경 사진 컬러화

# Colourise SG 의 문제점 및 개선 방향

## 문제점

위 사례들을 봤을 때, 가장 큰 문제점으로 두 가지를 제시할 수 있다.

먼저 사물과 음식 사진에서 두드러지게 나타나듯이, 붉은색 계열은 거의 무시되거나 왜곡이 발생한다. 음식도 보면 실제로는 붉은 계열의 음식들인데, 컬러화 알고리즘은 그를 적용하지 못하고 있다.

두번째로 서양 풍경 데이터와 동양 풍경 데이터가 모두 학습 데이터로 쓰이기는 하지만, 서양 풍경 사진의 컬러화 품질이 떨어지는 것을 볼 수 있다. 풍경 예시뿐만 아니라 위의 인물 사진에서도 볼 수 있듯, 서양 풍경의 배경이 거의 컬러화되지 못한 것을 확인할 수 있다

## 개선 방향

붉은색 관련 문제점의 개선 방향으로는, 많은 양의 붉은색 계열의 이미지를 통한 모델의 학습이 있을 것이다. 붉은색들 위주로 이루어진 이미지, 붉은색과 다른 색들이 적당히 섞인 이미지, 붉은색보다 다른 색들이 더 많은 이미지로 세 부류로 나누어 생성자 모델이 만든 컬러화 이미지를 구별자 모델이 실제 이미지라고 판단할 정도로 더 학습이 이루어져야 할 것이다. 또한 이 부분에서, 붉은색과 다른 색들이 적당히 섞인 이미지 부류와 붉은색보다 다른 색들이 더 많은 이미지 부류의 학습 정도가 상당한 정도로 이루어져야, 붉은색 관련 문제점을 개선할 수 있을 것이다.

두번째로 서양 풍경 사진 문제점의 개선 방향으로는, 어쩔 수 없이 서양 풍경 데이터의 학습을 늘리는 것뿐이다. Colourise SG가 싱가폴 풍경 데이터 위주로 학습을 한 프로젝트인 만큼, 서양 풍경 데이터가 적을 것이기 때문에 모델 학습 시 서양 풍경 데이터의 양을 늘리되 동양 풍경 데이터의 학습을 방해하지 않을 정도로 늘려야 할 것이다.

# 결론

흑백 사진의 컬러화는 역사적인 사진을 컬러화함으로써 역사의 한 순간을 생생히 느껴보거나, 누군가의 추억을 그린 흑백 사진을 생생한 기억으로 바꿔줄 수 있는 프로젝트이다. Colourise SG는 이미 잘 학습된 프로젝트이고 여타 다른 컬러화 프로젝트들보다 성공적인 사례들을 만들고 있지만, 본 보고서에서 제시한 문제점들을 잘 개선한다면 보다 더 나은 컬러화 결과를 제공할 수 있을 것이다

# 출처

## 개발자 블로그

Preston Lim, “Bringing black and white photos to life using Colourise.sg — a deep learning colouriser trained with old Singaporean photos, 2019.02 (Colourise SG 개발자 프로젝트 소개)

<https://blog.data.gov.sg/bringing-black-and-white-photos-to-life-using-colourise-sg-435ae5cc5036>

## 그 외

Jantic, “DeOldify”, 2018

<https://github.com/jantic/DeOldify>