이 모듈들을 다음과 같이 임포트하여 사용할 수 있습니다

```
from my_package import math_operations, string_operations

print(math_operations.add(5, 3))
```

```
8
```

my_package 패키지에서 math_operations 및 string_operations 모듈을 임포트합니다. 그러면 my_pacakage.math_operations.function과 같은 방식으로 사용할 필요 없이 해당 함수에 직접 액세스할 수 있습니다.

모듈과 패키지는 파이썬 코드를 논리적이고 관리하기 쉬운 방식으로 구성하고 구조화하는 데 기본이 됩니다. 모듈과 패키지는 코드 재사용성과 필요 없는 부분에 대한 구분을 용이하게 하여 복잡한 파이썬 프로그램을 더 쉽게 개발, 유지 관리 및 이해할 수 있게 해줍니다. 코드가 복잡하고 프로젝트가 거대해질수록 이러한 작업은 필수적인 작업이 됩니다.

다음으로는 OpenCV 및 텐서플로와 같은 강력한 라이브러리를 사용하여 이미지를 조작하고 분석하는 방법을 살펴보면서 파이썬으로 이미지 처리의 세계를 탐구해보겠습니다.

1.2.2 OpenCV

OpenCV는 컴퓨터로 이미지나 영상을 읽고, 이미지의 사이즈 변환이나 회전, 선분 및 도형 그리기, 채널 분리 등의 연산을 처리할 수 있도록 만들어진 오픈 소스 라이브러리로, 이미지 처리 분야에서 가장 많이 사용됩니다. 인텔에서 개발을 시작하여 2006년 10월 19일에 버전 1.0이 출시되었습니다. 초기에는 C 언어만 지원하였지만, 버전 2.0부터 C++를 중심적으로 지원하고, 또 이후에 파이썬 라이브러리로도 추가되었습니다.

안면 인식과 지문 인식, 객체 검출, 이상 탐지 등 다양한 이미지 처리에서 사용되며, 이미지 처리 분야에서 가장 널리 사용되는 OpenCV에 대해 알아보겠습니다.

OpenCV의 강점

현재 정말 다양한 언어 및 분야에서 OpenCV를 사용하고 있습니다. 어떻게 한 가지 툴이 다양한 언어와 분야를 아우르게 되었을지 살펴봅시다.

1. **다양한 이미지 및 비디오 처리**

   OpenCV는 이미지와 비디오 데이터를 다루는 다양한 기능을 제공합니다. 이미지 사이즈 변환, 회전, 필터링, 색상 공간 변환 등 다양한 처리를 간단하게 수행할 수 있습니다.

2. **컴퓨터 비전 알고리즘**

   컴퓨터 비전 알고리즘을 구현하기 위한 다양한 함수와 클래스를 제공합니다. 얼굴 인식, 객체 검출, 이미지 분할, 모션 추적 등의 작업을 쉽게 구현할 수 있습니다.

3. **머신 러닝 통합**

   머신 러닝 알고리즘을 구현하기 위한 툴도 제공하며, 다양한 머신 러닝 프레임워크와의 통합을 지원합니다. 텐서플로, 파이토치 등의 머신 러닝 프레임워크와 연계하여 사용할 수 있습니다.

4. **크로스 플랫폼 지원**

   다양한 플랫폼에서 동작하도록 설계되어 있으며, 윈도우, 리눅스, macOS, 안드로이드, iOS 등에서 사용할 수 있습니다.

5. **오픈 소스 라이선스**

   아파치 라이선스(Apache License 2.0)를 따르므로 무료로 사용할 수 있고, 소스 코드에 접근하여 필요에 따라 변경할 수 있습니다. 또 OpenCV를 사용한 툴을 상업용으로 배포하는 것 역시 가능합니다.

6. **높은 성능**

   C++로 작성되었기 때문에 빠른 속도와 효율적인 메모리 관리를 지원하며, 병렬 처리와 GPU 가속화를 활용하여 높은 성능을 제공합니다.

7. **커뮤니티 지원**

   활발한 개발자 및 사용자 커뮤니티가 존재하여 지속적으로 개발과 지원이 이루어지고 있습니다. 새로운 기능과 업데이트가 지속적으로 이루어지며, 문제를 해결하기 위한 다양한 자료와 지원이 제공됩니다.

8. **다양한 언어 지원**

   C++, 파이썬, 자바, C# 등 다양한 언어를 지원하여 개발자들이 자신에게 편한 언어를 선택하여 사용할 수 있습니다.

9. **광범위한 응용 분야**

   컴퓨터 비전과 이미지 처리 분야뿐만 아니라 로봇, 자율 주행, 의료, 보안, 산업 등 다양한 분야에서 사용됩니다.

이렇게 여러 면에서 장점이 많기 때문에 이미지 처리를 할 때는 늘 OpenCV를 사용하며, 우리도 이에 대해서 좀 더 자세히 알아볼 것입니다. 먼저 OpenCV를 사용한 이미지의 입출력, 이어서 이미지 변환 및 보강 방법에 대해 간단히 살펴보겠습니다.

이미지 입출력

OpenCV의 입력과 출력에 대해 살펴보기 전에 먼저 알아야 할 사실은, OpenCV 라이브러리는 구글 코랩이 아닌 로컬 컴퓨터를 기준으로 만들어졌다는 것입니다. 즉, 온라인 서버에서 구동하는 구글 코랩에서는 이미지, 영상 출력 기능에 해당하는 기능을 온전히 모두 사용해볼 수 없습니다.

그림 1-13 like_lenna.png 이미지

이 이미지는 우리가 사용할 like_lenna.png 파일입니다. 다음 코드로 이미지 데이터를 가져와서 사용합니다.

```
!wget https://raw.githubusercontent.com/Cobslab/imageBible/main/image/like_lenna224.
png -O like_lenna.png
```

wget은 리눅스의 터미널의 명령어 중 하나로, 인터넷 주소에 있는 파일을 다운로드할 수 있게 해줍니다. 따라서 이 코드를 실행하면 코랩 서버에 바로 사용할 수 있도록 like_lenna.png 파일이 준비됩니다.

이제 파이썬의 OpenCV 라이브러리를 사용하여 해당 이미지 파일을 읽어보겠습니다. 파이썬에서 OpenCV를 사용할 때는 다음 코드처럼 cv2를 import하여 사용합니다.

```
import cv2

image = cv2.imread('like_lenna.png', cv2.IMREAD_GRAYSCALE)
if image is not None:
    print("이미지를 읽어왔습니다.")
else:
    print("이미지를 읽어오지 못했습니다.")
print(f"변수 타입: {type(image)}")
```

```
이미지를 읽어왔습니다.
변수 타입: <class 'numpy.ndarray'>
```

출력 결과를 보니 OpenCV에서 해당 이미지 파일을 numpy.ndarray로 불러왔네요. 앞에서는 이미지를 픽셀 단위의 숫자들로 구성할 수 있음을 살펴보았습니다. 이러한 숫자들은 그 수가 매우 많기 때문에 효율적으로 저장하고 관리해야 합니다. 파이썬의 기본 자료형인 리스트는 매우 편리하지만, 연산할 때 다른 언어에 비해 느리다는 단점이 있습니다. 따라서 이미지 전체 픽셀에 대해 연산을 해야 하는 상황처럼 연산량이 많아진다면 기본 자료형인 리스트가 아닌 NumPy의 numpy.ndarray를 사용하면 속도가 빨라지고, 더 적은 메모리 공간을 차지하기 때문에 연산할 때 큰 이점이 있습니다. NumPy 라이브러리는 파이썬 도구이지만, 마치 C 언어에서 처리하는 것과 같은 속도로 각 연산을 처리해줍니다. 또 추후 데이터를 GPU 연산 장치로 옮길 때도 효율적입니다.

현재는 조금 단순한 이미지를 먼저 다뤄보기 위해 cv2.IMREAD_GRAYSCALE 인수를 주어 이미지를 흑백으로 받아왔습니다. 이어서 이미지를 출력해보겠습니다.

```
from google.colab.patches import cv2_imshow
cv2_imshow(image)
```

코랩에서는 OpenCV에서 지원하는 이미지 출력 함수 cv2.imshow를 사용할 수 없지만, 비슷하게 사용 가능하도록, cv2_imshow 함수를 만들어 제공합니다. 이를 이용하여 더 다양하게 이미지를 변환해볼 것입니다.

그러면 우리가 읽어온 이미지 image 변수는 어떤 형태일까요?

```
print(f"이미지 배열의 형태: {image.shape}")
```

```
이미지 배열의 형태: (224, 224)
```

OpenCV는 이미지를 읽어와 각 픽셀에 해당하는 수를 가로 방향 길이, 세로 방향 길이 배열(numpy.ndarray)로 변수에 저장해줍니다. 이로 인해 우리는 이미지의 각 픽셀에 접근해서 수정 혹은 변환하는 것이 가능해집니다.

이미지 변환

OpenCV를 사용해서 해볼 수 있는 다양한 변환 기능이 있습니다. 그중 많이 사용하는 사이즈 변환과 대칭 변환, 회전 변환 기능을 사용해보겠습니다.

사이즈 변환

cv2.resize 함수를 사용해 이미지의 사이즈를 바꿀 수 있습니다. resize 함수는 다음처럼 사용합니다.

```
def resize(src: MatLike, dsize: Size | None, ds: MatLike | None = ..., fax: float
= ..., fey: float = ..., interpolation: int = ...) -> MatLike
```

cv2.resize 함수에 들어가는 각 요소들은 다음과 같은 의미를 갖습니다.

표 1-1 cv2.resize 함수의 인수 소개

항목	의미
src	입력 이미지 혹은 입력 영상을 입력합니다.
dsize	변환 후 이미지의 형태로, (가로의 길이, 세로의 길이) 형식으로 튜플로 입력합니다.
ds	출력 이미지를 저장할 numpy.ndarray 변수를 입력합니다.
fx, fy	입력 이미지와 출력 이미지의 배율을 의미합니다. 해당 인수를 사용하기 위해서는 dsize 인수에 None 값을 대입합니다.
interpolation	이미지를 확대할 때 비어 있는 픽셀을 채울 규칙을 지정합니다.

앞에서 불러왔던 이미지의 image 변수를 사용해 이미지의 사이즈를 바꿔보겠습니다.

```
image_small = cv2.resize(image,(100,100))
cv2_imshow(image_small)
```

(224,224) 사이즈였던 이미지를 (100,100) 사이즈로 수정했습니다. 이렇게 이미지의 가로, 세로 픽셀의 개수 단위를 정해여 이미지의 사이즈를 바꿀 수 있습니다.

이번에는 배율을 지정하는 방식을 사용해보겠습니다. 원래 image의 사이즈 값을 넣어주었던 dsize 인수에 이미지의 사이즈 대신 None을 대입하고, fx, fy 값을 원하는 배율로 입력합니다.

```
image_big = cv2.resize(image, dsize=None, fx=2, fy=2,)
cv2_imshow(image_big)
```

대칭 변환

이미지 대칭 변환은 이미지를 수평, 수직 또는 두 축 모두에 대해 반전시키는 작업을 의미합니다. OpenCV에서는 cv2.flip() 함수를 사용하여 이미지를 대칭, 변환할 수 있습니다. flip 함수는 image 변수와 대칭으로 돌릴 축을 0, 1 등으로 명시하여 변환을 수행합니다. 0으로 명시하면 다음처럼 수평축으로 반전됩니다.

```
image_fliped = cv2.flip(image, 0)
cv2_imshow(image_fliped)
```

1로 명시하면 다음처럼 세로축으로 반전됩니다.

```
image_fliped = cv2.flip(image, 1)
cv2_imshow(image_fliped)
```

회전 변환

이미지 회전 변환은 이미지를 주어진 각도만큼 회전시키는 작업을 말합니다. OpenCV에서는 `cv2.getRotationMatrix2D` 함수를 사용하여 회전 변환 행렬을 생성하고, `cv2.warpAffine` 함수를 사용하여 실제로 이미지를 회전시킵니다.

cv2.getRotationMatrix2D 함수는 center 인수로 어떤 점을 기준으로 회전시킬지를, angle 인수로 얼마나 회전시킬지를, scale 인수로 결과 이미지의 배율을 어떻게 할지를 지정하여 다음처럼 해당 변환 행렬을 반환합니다.

```
def getRotationMatrix2D(center: Point2f, angle: float, scale: float) -> MatLike
```

cv2.warpAffine 함수는 앞에서 만들어진 변환 행렬과 이미지를 받아 다음처럼 실제 회전 변환을 수행합니다.

```
height, width = image.shape
matrix = cv2.getRotationMatrix2D((width/2, height/2), 90, 1)
result = cv2.warpAffine(image, matrix, (width, height))
cv2_imshow(result)
```

다음처럼 원하는 각도로 이미지를 회전시킬 수도 있습니다.

```
matrix = cv2.getRotationMatrix2D((width/2, height/2), 30, 1)
result = cv2.warpAffine(image, matrix, (width, height), borderValue=200)
cv2_imshow(result)
```

자르기

이미지 자르기 기능은 파이썬 배열에서의 슬라이싱[1] 기능을 사용합니다. 다음 코드는 이미지의 왼쪽 상단을 기준으로 픽셀 단위 가로 100, 세로 100만큼의 위치를 잘라줍니다. 컴퓨터에서의 배열 특성에 따라 `image[:100,:100]`에서 (,) 앞쪽의 `:100`은 세로 방향의 사이즈, 뒤쪽의 `:100`은 가로 방향의 사이즈를 나타냅니다.

```
cv2_imshow(image[:100, :100])
```

다음 코드는 이미지의 왼쪽 상단을 기준으로 가로, 세로 모두 51번째 픽셀부터 150번째 픽셀까지의 위치를 잘라줍니다.

```
cv2_imshow(image[50:150, 50:150])
```

1 열에서 일부를 잘라오는 파이썬의 주요 기능으로, 인덱스를 사용해 원하는 위치를 복사하거나 참조합니다. 파이썬 기본 자료형인 리스트에서 슬라이싱을 사용하면 얕은 복사로, NumPy의 `numpy.ndarray`에서 사용하면 참조로 동작합니다.

numpy.ndarray의 슬라이싱은 원본 객체의 값을 그대로 참조합니다. 즉, 할당되어 있는 픽셀의 숫자 값을 바꿔주면 원본 자체의 픽셀 값이 함께 변합니다. 다음 코드를 통해 자른 이미지에 다른 값을 할당시키면 원본 사진 자체가 변한 것을 확인할 수 있습니다.

```
croped_image = image[50:150, 50:150]
croped_image[:] = 200
cv2_imshow(image)
```

이렇게 이미지 일부를 잘라서 변화를 주고 싶지만, 원본 이미지에 영향을 미치고 싶지 않을 때는 자른 객체에 대한 깊은 복사[2]를 하여 사용해야 합니다. NumPy의 numpy.ndarray는 copy 메서드로 깊은 복사 기능을 제공합니다.

```
image = cv2.imread('like_lenna.png', cv2.IMREAD_GRAYSCALE)
croped_image = image[50:150, 50:150].copy()
croped_image[:] = 200
cv2_imshow(image)
```

2 얕은 복사와 깊은 복사: 얕은 복사는 데이터 구조의 주소만을 복사하므로 원본과 복사본이 상호 영향을 미치지만, 깊은 복사는 데이터의 실제 값을 복사하여 원본과 완전히 분리된 복사본을 생성합니다.

이와 같이 copy 메서드를 사용하여 할당된 cropped_image 변수에는 다른 값을 할당해도 원본 이미지의 값이 변하지 않습니다.

도형 그리기

OpenCV는 읽어온 이미지 위에 원하는 도형을 그릴 수 있는 기능을 제공합니다. 이 기능을 사용해 이미지 위에 객체를 구분하는 박스를 그리거나, 글씨를 쓰는 일 등의 작업을 할 수 있습니다. 자주 사용하는 기능과 함수는 다음과 같습니다.

- 선 그리기: cv2.line
- 원 그리기: cv2.circle
- 직사각형 그리기: cv2.rectangle
- 타원 그리기: cv2.ellipse
- 다각형 그리기: cv2.polylines, cv2.fillPoly

사용법에 대해 하나씩 살펴보겠습니다.

선 그리기: cv2.line

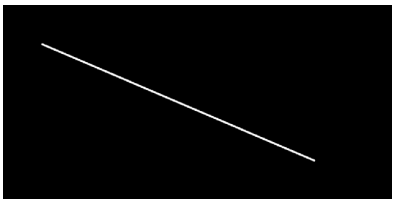
cv2.line 함수는 다음처럼 사용합니다.

```
def line(img: MatLike, pt1: Point, pt2: Point, color: Scalar, thickness: int = ...,
lineType: int = ..., shift: int = ...) -> MatLike
```

선을 그리는 cv2.line 함수는 이미지와 두 개의 점을 받아서 이어줍니다. color, thickness 등의 옵션으로 선을 다양하게 그려볼 수 있습니다. 다음 코드는 색상의 픽셀 값으로 255를 설정해 흰색 선을 그립니다.


```
space = np.zeros((500, 1000), dtype=np.uint8)
line_color = 255
space = cv2.line(space, (100, 100), (800, 400), line_color, 3, 1)

cv2_imshow(space)
```



위와 같이 평면에서 두 개 점의 좌표를 받아 선을 그려줍니다.

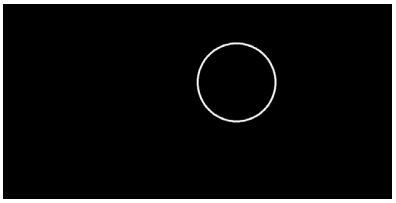
원 그리기: cv2.circle

cv2.circle 함수는 원의 중심 좌표와 반지름 값을 받아 원을 그립니다.

```
def circle(img: MatLike, center: Point, radius: int, color: Scalar, thickness: int
= ..., lineType: int = ..., shift: int = ...) -> MatLike
```

```
space = np.zeros((500, 1000), dtype=np.uint8)
color = 255
space = cv2.circle(space, (600, 200), 100, color, 4, 1)

cv2_imshow(space)
```



직사각형 그리기: cv2.rectangle

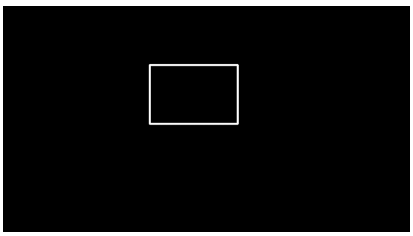
직사각형을 그리는 기능은 객체 탐지에서 특정 대상이 있을 때, 상자 표시로 해당 대상을 나타내는 데 사용하는 기능입니다. 다음 코드와 같이 이미지와 두 개의 좌표를 받아 이미지 위에 직사각형을 그려줍니다. pt1, pt2는 각각 사각형의 왼쪽 위와 오른쪽 아래의 꼭짓점 좌표를 의미합니다.

```
def rectangle(img: MatLike, pt1: Point, pt2: Point, color: Scalar, thickness: int
= ..., lineType: int = ..., shift: int = ...) -> MatLike
```

다음 코드를 실행하여 확인해보세요.

```
space = np.zeros((768, 1388), dtype=np.uint8)
line_color = 255
space = cv2.rectangle(space, (500, 200), (800, 400), line_color, 5, 1)

cv2_imshow(space)
```



타원 그리기: cv2.ellipse

이미지와 타원의 중심 및 축, 축의 각도, 타원을 그릴 각도(시작점과 끝점)를 받아, 이미지 위에 원하는 타원을 그립니다.

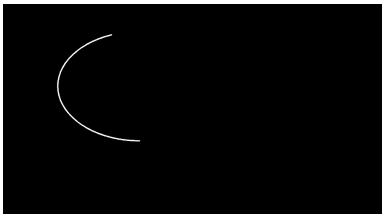
```
def ellipse(img: MatLike, center: Point, axes: Size, angle: float, startAngle:
float, endAngle: float, color: Scalar, thickness: int = ..., lineType: int = ...,
shift: int = ...) -> MatLike
```

center에는 타원 중심의 좌표, axes에는 축의 좌표, angle에는 타원축의 각도, startAngle에는 타원을 그리기 시작할 각도 값, endAngle에는 타원 그리기를 끝낼 각도, color에는 색상(숫자로 기입), thickness에는 선의 두께를 입력합니다. lineType과 shift는 어떤 타원을 그릴 것인지와 직결되는 인수가 아니므로 생략합니다.

다음 코드를 실행하여 확인해보세요.

```
space = np.zeros((768, 1388), dtype=np.uint8)
line_color = 255
space = cv2.ellipse(space, (500, 300), (300, 200), 0, 90, 250, line_color, 4)

cv2_imshow(space)
```



다각형 그리기: cv2.polylines, cv2.fillPoly

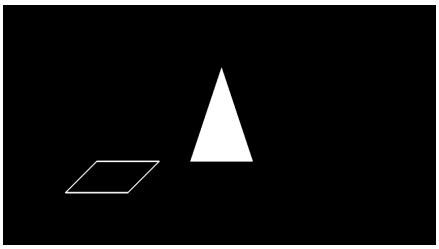
두 가지 함수로 다각형을 그릴 수 있습니다. cv2.polylines 기능은 여러 점을 잇는 선을 그려서 다각형을 그릴 수 있게 지원하고, cv2.fillPoly 기능은 색칠된 면을 갖는 다각형을 그립니다.

```
def polylines(img: MatLike, pts: Sequence[MatLike], isClosed: bool, color: Scalar,
thickness: int = ..., lineType: int = ..., shift: int = ...) -> MatLike


def fillPoly(img: MatLike, pts: Sequence[MatLike], color: Scalar, lineType: int =
..., shift: int = ..., offset: Point = ...) -> MatLike
```

다음 코드는 두 가지 방법으로 다각형을 그립니다.

```
space = np.zeros((768, 1388), dtype=np.uint8)
color = 255
obj1 = np.array([[300, 500], [500, 500], [400, 600], [200, 600]])
obj2 = np.array([[600, 500], [800, 500], [700, 200]])
space = cv2.polylines(space, [obj1], True, color, 3)
space = cv2.fillPoly(space, [obj2], color, 1)
cv2_imshow(space)
```



이 코드에서 cv2.polylines 함수는 [[300, 500], [500, 500], [400, 600], [200, 600]] 위치의 평행사변형을, cv2.fillPoly 함수는 [[600, 500], [800, 500], [700, 200]] 위치의 삼각형을 그려줍니다.

지금까지 OpenCV의 다양한 특성과 기능을 간단히 살펴보았습니다. OpenCV는 좀 더 다양하고 강력한 기능을 많이 가지고 있습니다. 잠시 뒤에 이미지 처리와 컴퓨터 비전에 대한 다양한 이야기를 다룰 예정이며, 더 깊이 있는 기능들도 살펴볼 것입니다.

1.2.3 텐서플로

그림 1-14 텐서플로

텐서플로(TensorFlow)는 구글 브레인 팀에서 개발되어 2015년에 공개된 오픈 소스 머신 러닝 라이브러리입니다. 공개 이후로 텐서플로는 머신 러닝 및 딥러닝 분야에서 중요한 역할을 하고 있으며, 연구자와 개발자들 사이에서 널리 사용되고 있습니다.

텐서플로는 이름에서 알 수 있듯이 텐서(다차원 배열)를 기반으로 하는 연산을 수행하며, 텐서플로를 사용하면 복잡한 머신 러닝 모델과 알고리즘을 쉽게 구현할 수 있습니다. 또한 텐서플로는 데이터 플로 그래프를 사용하여 연산을 표현합니다. 이 그래프는 노드와 에지로 구성되어 있으며, 노드는 연산, 에지는 노드 사이를 이동하는 데이터(텐서)를 나타냅니다. 데이터 플로 그래프를 사용해서 텐서플로는 병렬 처리와 지연 실행을 쉽게 수행할 수 있습니다.

텐서플로의 몇 가지 특징을 살펴봅시다.

편의성

텐서플로의 특징 중 하나는 편의성에 있습니다. 고수준 API 지원, 텐서플로 개발진들이 미리 개발해둔 사전 빌드된 층 및 모델 제공, 즉시 실행 모드 등이 특징입니다.

고수준 API 지원

텐서플로의 가장 사용자 친화적인 기능 중 하나는 고수준 API인 케라스(Keras)가 있다는 것입니다.

그림 1-15 케라스

케라스 API는 간단하고 일반적인 사용 사례에 최적화되어 있어 거의 모든 표준 모델을 정의하고 구성할 수 있는 명확하고 간결한 방법을 제공합니다. 앞서 설명한 것처럼 간단한 선형 회귀부터 복잡한 심층 신경망까지, 단 몇 줄의 코드만으로 모델을 설정하고 컴파일할 수 있습니다.

또한 케라스는 모델 아키텍처를 자유롭게 정의할 수 있는 기능도 제공합니다. 예를 들어 순차적 모델을 사용하면 각 층에 정확히 하나의 입력 텐서와 하나의 출력 텐서가 있는 층을 쉽게 스택처럼 차곡차곡 쌓아 올릴 수 있습니다. 이는 정보의 흐름이 입력에서 출력으로 한 방향으로만 이루어지는 모델에 유용합니다.

이와는 대조적으로 Model 클래스 API를 사용하면 더 복잡한 모델을 만들 수도 있습니다. 이 클래스는 층 그래프를 정의하는 데 사용됩니다. 다중 출력 모델, 방향성 비순환 그래프 또는 공유 층이 있는 모델(데이터 흐름이 엄격하게 선형적이지 않은 상황)을 만들 때 유용합니다. 모델의 복잡성에 관계없이 케라스는 직관적이고 사용자 친화적인 모델 생성 방식을 제공합니다.

이식성 및 호환성

플랫폼 간 이식성과 호환성은 텐서플로의 가장 큰 특징 중 두 가지로, 다양한 환경과 기기에서 작업하는 개발자를 위한 매우 다재다능한 도구입니다. 이러한 기능 덕분에 텐서플로는 하나의 플랫폼이나 특정 카테고리의 디바이스만을 위한 도구가 아니라 다양한 운영 환경에서 머신 러닝 애플리케이션을 위한 범용 솔루션이 될 수 있었습니다. 소프트웨어의 맥락에서 이식성은 동일한 소프트웨어를 다른 환경에서도 사용할 수 있는 사용성을 의미합니다. 이는 텐서플로와 같이 광범위한 애플리케이션에 사용할 수 있도록 설계된 도구의 중요한 특성입니다. 텐서플로 모델의 높은 이식성 덕분에 개발자는 한 환경에서 머신 러닝 모델을 작업한 후 테스트, 배포 또는 추가 개발을 위해

다른 환경으로 쉽게 전환할 수 있습니다. 이는 유연성과 적응성이 핵심인 머신 러닝 분야에서 특히 중요합니다.

이식성은 개발자가 특정 플랫폼이나 기기에 종속되는 것을 방지합니다. 다양한 하드웨어(CPU, GPU, TPU), 소프트웨어 플랫폼(윈도우, macOS, 리눅스) 등 다양한 환경에서 실행할 수 있으므로 개발자는 단일 운영 체제에 국한되지 않습니다. 따라서 텐서플로를 사용하는 개발자, 연구자 및 조직 커뮤니티는 머신 러닝 모델의 기능이나 효율성을 저하시키지 않으면서도 원하는 운영 환경을 선택할 수 있습니다. 개인용 macOS에서 작업하든, 윈도우 워크스테이션에서 작업하든, 리눅스 서버에서 작업하든 동일한 효율로 텐서플로 모델을 개발, 훈련 및 배포할 수 있습니다. 이러한 수준의 플랫폼 독립성은 서로 다른 시스템에서 작업하는 팀 간의 협업을 가능하게 하고 플랫폼별 종속성으로 인한 문제를 제거합니다.

텐서플로 이식성의 또 다른 핵심 측면은 모바일 및 에지 디바이스와의 호환성입니다. 텐서플로에서 제공하는 도구 세트인 텐서플로 라이트(TensorFlow Lite)를 사용하면 모바일 및 에지 장치에서 실행할 수 있는 형식으로 텐서플로 모델을 변환할 수 있습니다. 이를 통해 온디바이스 머신 러닝, 실시간 처리 및 IoT 애플리케이션과 같은 완전히 새로운 텐서플로 모델 애플리케이션의 문이 열립니다.

모바일 및 에지 디바이스에서 머신 러닝 모델을 실행할 수 있는 기능은 상당한 영향을 미칠 수 있습니다. 예를 들어 실시간으로 작동해야 하는 머신 러닝 모델은 온디바이스 추론을 통해 서버 기반 또는 클라우드 기반 솔루션에 비해 지연 시간을 크게 줄일 수 있습니다. 또한 디바이스에서 모델을 실행하면 데이터를 서버로 전송할 필요가 없으므로 사용자 개인정보 보호가 향상될 수 있습니다.

텐서플로는 로컬 환경에만 국한되지 않습니다. 구글 클라우드 플랫폼(Google Cloud Platform, GCP), 아마존 웹 서비스(Amazon Web Services, AWS), 마이크로소프트 애저(Microsoft Azure)와 같은 클라우드 플랫폼과 원활하게 통합되도록 설계되었습니다. 클라우드 호환성 덕분에 개발자는 이러한 플랫폼에서 사용할 수 있는 방대한 계산 리소스를 활용하여 로컬 머신에서는 불가능했던 복잡한 대규모 모델을 학습할 수 있습니다.

텐서플로의 이식성과 플랫폼 간 호환성은 머신 러닝을 위한 매우 유연하고 보편적으로 적용할 수 있는 도구입니다. 개인용 컴퓨터에서 모바일 기기, 단일 머신에서 방대한 클라우드 기반 인프라에 이르기까지 다양한 플랫폼과 기기에서 모델을 개발하고 배포할 수 있는 능력은 텐서플로에게 상당한 우위를 제공합니다. 이러한 기능 덕분에 텐서플로는 개발자에게 실용적인 선택이 될 뿐만 아니라 다양한 애플리케이션에서 머신 러닝의 접근성과 다용도성, 영향력을 높일 수 있습니다.

확장성

텐서플로의 확장성은 다른 머신 러닝 라이브러리와 차별화되는 포인트 중 하나이며 머신 러닝 분야에서 인기가 높은 주요 이유입니다. 확장성이란 시스템, 네트워크 또는 프로세스가 증가하는 작업량을 유능한 방식으로 처리할 수 있는 능력 또는 이러한 증가를 수용하기 위해 확장할 수 있는 잠재력을 말합니다. 텐서플로는 모델 개발과 배포 모두를 위한 확장 가능한 솔루션을 제공하므로 다양한 규모의 머신 러닝 애플리케이션에 이상적인 선택입니다.

머신 러닝 프로젝트는 제한된 양의 데이터와 간단한 모델로 소규모로 시작하는 경우가 많습니다. 하지만 프로젝트가 발전함에 따라 더 큰 데이터 세트를 처리하고, 더 복잡한 모델을 만들고, 더 많은 컴퓨팅 리소스를 필요로 할 수 있습니다. 증가하는 수요를 충족하기 위해 확장할 수 있는 능력은 모든 머신 러닝 프레임워크의 중요한 기능입니다. 이를 통해 소규모 실험에서 대규모 배포로 좀 더 원활하게 전환할 수 있습니다. 필요에 따라 확장할 수 있는 이러한 기능은 프로젝트 초기 단계에 투자한 시간과 리소스를 낭비하지 않고 프로젝트의 수명 주기 내내 동일한 도구와 기술을 사용할 수 있도록 보장합니다.

1. CPU, GPU 및 TPU 확장성

코드 변경 없이, CPU(중앙 처리 장치), GPU(그래픽 처리 장치), 심지어 머신 러닝 워크로드를 가속화하는 데 특별히 사용할 수 있는 구글의 맞춤형 개발 애플리케이션별 집적 회로(ASIC)인 TPU(텐서 처리 장치)에서도 실행할 수 있습니다. GPU와 TPU는 일반적인 CPU보다 초당 훨씬 더 많은 계산을 수행할 수 있으므로 머신 러닝에서 흔히 사용되는 행렬 및 벡터 연산에 훨씬 더 효율적입니다. GPU와 TPU에서 모델을 훈련할 수 있기 때문에 텐서플로는 CPU만 사용할 때보다 훨씬 더 큰 데이터 세트와 더 복잡한 모델을 처리할 수 있습니다.

2. 분산 컴퓨팅

다양한 유형의 하드웨어에서 실행할 수 있을 뿐만 아니라 텐서플로는 분산 컴퓨팅도 지원합니다. 이를 통해 개발자는 여러 머신에서 동시에 모델을 훈련할 수 있습니다. 이는 매우 큰 데이터 세트나 특히 복잡한 모델을 처리하는 데 중요한 기능입니다.

텐서플로의 분산 전략 API는 분산 컴퓨팅의 많은 세부 사항을 추상화하여 모델 학습을 더 쉽게 확장할 수 있게 해줍니다. 여러 GPU에서 동기식 트레이닝을 위한 `tf.distribute.MirroredStrategy`, TPU에서 트레이닝을 위한 `tf.distribute.experimental.TPUStrategy` 등 데이터 및 계산을 분산하기 위한 다양한 전략을 지원합니다.

3. **대규모 모델 배포**

   모델이 학습되면 텐서플로는 대규모로 모델을 배포할 수 있는 도구도 제공합니다. 텐서플로 서빙(TensorFlow Serving)은 프로덕션 환경을 위해 설계된 머신 러닝 모델을 위한 유연한 고성능 서빙 시스템입니다. 텐서플로 모델과 바로 통합할 수 있지만 다른 유형의 모델을 제공하도록 확장할 수 있습니다.

   특히 여러 모델을 서비스하고, 모델 버전 관리를 수행하고, 다양한 유형의 하드웨어에서 모델을 서비스할 수 있습니다. 매우 유연하고 확장 가능하며 대규모로 모델을 제공할 수 있도록 설계되어 프로덕션급 제품으로 전환이 가능합니다.

텐서플로의 핵심적인 기능인 확장성은 모든 규모와 다양한 복잡성을 지닌 머신 러닝 작업을 처리할 수 있습니다. 단일 CPU에서 간단한 모델을 실행하든 분산된 GPU 또는 TPU 네트워크에서 복잡한 모델을 훈련하든, 텐서플로는 작업 규모를 확장하는 데 필요한 도구와 유연성을 제공합니다. 이러한 확장성은 텐서플로의 휴대성 및 광범위한 기능과 결합되어 텐서플로를 머신 러닝을 위한 강력한 도구로 만들어줍니다.

유연성

텐서플로의 방대한 기능들 중 다른 한 핵심은 내재된 유연성입니다. 이러한 유연성 덕분에 간단한 선형 회귀부터 복잡한 신경망 아키텍처에 이르기까지 다양한 작업에 고유하게 적용할 수 있습니다. 모델 아키텍처 설계를 넘어 데이터 전처리부터 배포까지 모든 수준의 개발에 적용됩니다. 지속적으로 진화하는 머신 러닝 영역에서 획일성은 더 이상 통하지 않습니다. 문제의 복잡성과 가변성으로 인해 상황에 따라 고수준 API가 아닌, 저수준 API(low level API), 사용자 정의 층들도 변경할 수 있는 도구와 프레임워크가 필요해졌습니다. 바로 이 지점에서 텐서플로의 유연성이 중요한 특성으로 부각됩니다.

1. **사용자 정의 층**

   텐서플로는 기본적으로 많은 표준 층을 제공하지만, 특정 문제에 따라 고유한 층이 필요한 경우가 있을 수 있습니다. 텐서플로의 저수준 API는 번거로움 없이 이러한 층을 제작할 수 있는 기능을 제공합니다.

2. **사용자 정의 손실 함수**

   상황에 따라 표준 손실 함수를 사용해 모든 문제를 효과적으로 해결할 수 있는 것은 아닙니다. 맞춤형 손실 함수를 생성할 수 있는 도구를 제공함으로써 텐서플로는 자유도 높은 모델 설계를 할 수 있습니다.

3. **새로운 최적화 전략 구현**

   경사 하강법 변형 함수들이 머신 러닝 환경을 지배하고 있지만, 특정 시나리오에서는 혁신적인 최적화 방법이 필요할 수 있습니다. 텐서플로는 이러한 방법을 원활하게 통합할 수 있도록 지원합니다.

4. **tf.data를 사용한 데이터 파이프라인**

   머신 러닝에서는 데이터가 가장 중요합니다. 텐서플로의 tf.data API는 효율적인 데이터 파이프라인을 구축할 수 있는 강력한 유틸리티 세트를 제공합니다. 정형 데이터, 비정형 데이터, 심지어 시계열 및 NLP를 위한 시퀀스까지, `tf.data`는 모든 데이터를 처리할 수 있을 만큼 다용도로 사용할 수 있습니다.

초기 버전부터 지금까지, 텐서플로는 지속적으로 발전해왔으며, AI 발전에 있어 보조를 하기도, 주력으로 사용되기도 하였습니다. 사용자 친화적인 인터페이스나 심층적인 사용자 지정 옵션을 통해 텐서플로는 AI의 대중화를 위해 노력하고 있으며, 누구나 어디서나 머신 러닝의 힘을 활용할 수 있도록 보장합니다.

텐서플로는 도구로써도 강력하지만, 그 잠재력을 진정으로 발휘하는 것은 이론을 바탕으로 한 창의력, 호기심이라는 점을 기억하기 바랍니다. 프레임워크, API, 전략은 도구일 뿐이며, 그 도구가 만들어내는 교향곡은 전적으로 여러분의 몫입니다.

이제 이미지 처리와 컴퓨터 비전의 매혹적인 영역에 대해 더 깊이 알아볼 준비가 되었습니다. 여기서 얻은 인사이트와 지식은 앞으로 이어질 장들에서 중요한 역할을 할 것이므로 잘 기억해두기 바랍니다.

2장 이미지 처리 기초

이미지 처리의 핵심을 살펴보는 여정을 시작하는 장입니다. 이미지 처리의 핵심은 이미지를 조작하고 분석하여 정보를 수집하고, 기능을 향상시키거나, 다양한 애플리케이션에 맞게 이미지를 변환하는 것입니다. 이미지 처리에 신기한 기술들이 다양하게 있지만, 핵심은 기본 원리에 있습니다. 이 장에서는 가장 기본적이면서도 기초적인 개념부터 시작하여 이 매혹적인 영역의 층을 벗겨 보려고 합니다.

이미지 처리의 복잡한 태피스트리(tapestry)는 수학, 예술, 과학, 기술의 실타래로 짜여 있습니다. 여기에서는 이미지 처리를 지원하는 도구와 기술을 알아볼 뿐만 아니라 디스플레이, 자동차, 엔터테인먼트, 소셜 미디어 등 다양한 분야에 미치는 광범위한 컴퓨터 비전에 대해서도 알아보겠습니다.

IMAGE PROCESSING

2.1 이미지란?

아침에 일어나 휴대폰을 확인하는 순간부터 밤에 전자책을 읽거나 동영상을 시청하는 마지막 순간까지 우리는 매일 이미지를 접합니다. 하지만 무엇이 이미지를 구성하는 요소인지 잠시 멈춰 생각해본 적 있나요? 이번 절에서는 직접 이미지를 해체해보겠습니다. 색상의 배열을 넘어 기본 구조를 살펴보고 디지털 표현이 어떻게 우리가 보는 것에 생명을 불어넣는지 살펴볼 것입니다. 또한 이미지의 다양한 관점을 제공하는 다양한 색상 공간에 대해 알아보고, 각각 고유한 시각과 수학의 조화를 살펴볼 것입니다.

2.1.1 디지털 이미지의 구조

디지털 이미지의 구조는 퍼즐 조각처럼 서로 맞물리는 픽셀들의 집합체로, 각 픽셀은 색상과 밝기의 미세한 차이를 통해 전체 이미지에 명확함과 섬세함을 부여합니다.

픽셀

모든 디지털 이미지의 핵심은 **픽셀**이라는 작은 사각형의 모자이크입니다. '픽셀'은 그림(picture)과 요소(element)의 합성어입니다. 이것은 우리가 모니터 화면을 자세히 보면 보이는 가장 작은 요소

이자 이미지를 확대하면 나오는 아주 작은 단일 점입니다. 이미지를 크게 확대하면 이 작은 사각형으로 이미지가 구성되어 있음을 알 수 있습니다.

단일 픽셀 값은 흑백 이미지에서 밝기를 나타내는 숫자입니다. 컬러 이미지에서 픽셀은 빨강, 녹색, 파랑 등 서로 다른 색상 채널을 나타내는 여러 픽셀 값을 가질 수 있습니다. 컬러 이미지에 대한 내용은 2.1.2절에서 자세히 다룰 예정입니다.

해상도

디지털 이미지의 '해상도'는 이미지가 보유하고 있는 픽셀의 양을 나타냅니다. 예를 들어 해상도는 1920×1080과 같은 형식으로 쓰며, 이는 이미지가 1920픽셀 너비와 1080픽셀 높이를 가지고 있다는 것을 의미합니다.

그림 2-1 이미지 해상도 예시

전체 픽셀 수는 이미지의 디테일 양을 결정하는 주요 요소로 간주됩니다. 디지털 이미징 초기에는 해상도가 큰 제약 사항이었습니다. 초기 디지털 카메라와 스캐너는 640×480 픽셀로 이미지를 생성했습니다. 시간이 지남에 따라 디지털 이미지의 해상도는 지금 우리가 익숙한 멀티메가픽셀 이미지로 이어지는 성장을 보였습니다.

이미지 해상도에 영향을 주는 요소들은 다음과 같습니다.

- **카메라의 센서 사이즈**: 디지털 카메라로 촬영한 이미지의 해상도는 주로 카메라의 센서 사이즈에 의해 결정됩니다. 큰 센서는 더 많은 픽셀을 수용할 수 있어, 더 높은 해상도의 이미지를 생성할 수 있습니다.

- **스캐너의 정밀도**: 물리적 미디어를 디지털화할 때, 해상도는 사용된 스캐너의 정밀도와 품질에 의해 결정됩니다.

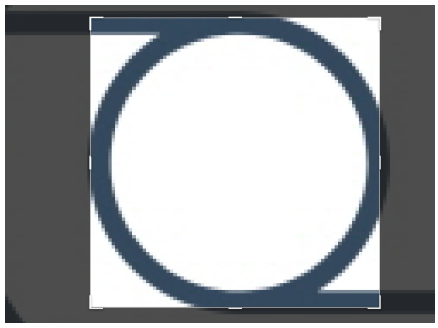
해상도 및 영상 규격을 지칭하는 용어들은 다음과 같습니다.

- **WVGA**: 800×480 픽셀로, 과거 동영상 표준 규격이며 초창기 핸드폰에서 사용하던 규격입니다.
- **HD**: High Definition의 약자로 고화질이라는 의미를 포함합니다. 1280×720 픽셀을 사용하며, 세로 픽셀 개수에 따라 영상에서는 720p로 표현하기도 합니다.
- **FHD**: Full HD의 약자로 1920×1080의 픽셀을 사용합니다. 영상에서는 1080p로 표현하기도 합니다.
- **QHD**: Quad HD의 약자로 2560×1440 픽셀입니다. 영상에는 1440p로 표현합니다.
- **4K와 UHD**: UHD는 디스플레이 브랜드마다 Ultra HD, Ultra High Definition 등 다른 용어로 표기합니다. 4K는 가로 픽셀이 4000 이상을 뜻하지만 이보다 덜하거나 더 사용한 경우에도 4K라고 표현하며 보통 4K UHD를 같이 사용합니다.

픽셀 밀도

픽셀 밀도는 디스플레이의 픽셀이 얼마나 촘촘하게 배열되어 있는지를 나타내는 척도입니다. 일반적으로 인치당 픽셀 수(PPI, Pixel Per Inch), 때로는 센티미터당 픽셀 수로 측정됩니다. PPI가 높을수록 주어진 공간에 더 많은 픽셀이 채워져 있습니다. 해상도가 높은 이미지를 보더라도 PPI가 낮다면 이미지가 울퉁불퉁하게 보입니다 반대로 픽셀 밀도가 높을수록 개별 픽셀의 사이즈가 작아져 육안으로 구분하기 어렵습니다. 그 결과 이미지가 더 부드러워지고 텍스트와 그래픽이 더 선명해집니다. 예를 들어 높은 PPI 디스플레이에서 텍스트를 읽을 때 글자 가장자리가 더 매끄럽고 픽셀화가 덜 된 것처럼 보입니다.

그림 2-2 이미지 깨짐 예시



픽셀 밀도는 디스플레이 기술의 발전과 함께 계속해서 중요해지고 있습니다. 초기의 컴퓨터 모니터나 텔레비전은 지금의 디스플레이에 비해 상대적으로 낮은 PPI를 가지고 있었습니다. 그러나 기술의 발전과 함께, 특히 모바일 기기와 고해상도 모니터의 등장으로 PPI는 크게 증가하였습니다. 이러한 발전은 이미지와 텍스트의 선명도를 크게 향상시켰습니다.

픽셀 밀도는 사용자 경험에도 큰 영향을 미칩니다. 높은 PPI를 가진 디스플레이는 이미지와 텍스트를 더 선명하게 표시하므로, 사용자는 더 풍부하고 현실적인 경험을 얻을 수 있습니다. 이는 특히 VR 및 AR과 같은 현실 증강 기술에서 중요합니다. 높은 PPI는 이러한 기술을 통해 제공되는 경험의 질을 크게 향상시킵니다. 또한 디스플레이의 에너지 효율성과 밝기에도 영향을 미칩니다. 일반적으로 PPI가 높은 디스플레이는 더 많은 에너지를 소비할 수 있습니다. 그러나 최신 디스플레이 기술은 높은 PPI를 유지하면서도 에너지 효율성을 개선하고 있습니다.

서브 픽셀

이미지 디지털 디스플레이 영역에서 서브 픽셀은 단순한 이미지 처리보다 더 복잡한 역할을 합니다. 서브 픽셀은 본질적으로 화면의 픽셀을 구성하는 작은 컬러 요소입니다. 디스플레이는 이러한 RGB 서브 픽셀에서 방출되는 빛의 강도를 조작해서 우리가 보는 색상의 스펙트럼을 재현할 수 있습니다. 강렬한 빨간색부터 차분한 파란색까지 화면의 각 색상은 이러한 미세한 RGB 요소들이 조화롭게 작동하여 생동감을 더합니다.

더 밝고 에너지 효율적인 디스플레이를 추구하면서 RGBW(white)라고 하는 흰색 서브 픽셀이 추가로 통합되었습니다. 흰색 서브 픽셀을 사용하면 디스플레이는 RGB 서브 픽셀을 완전히 활성화하지 않고도 흰색을 표현할 수 있으므로 에너지를 절약할 수 있습니다.

픽셀과 서브 픽셀을 이해하면 이미지가 어떻게 형성되고 표시되는지 명확하게 알 수 있지만, 디지털 이미지의 중요한 특징 중 하나는 이러한 이미지가 저장되고 공유되는 방식도 마찬가지로 중요하다는 점에 주목할 필요가 있습니다. 결국 기존 RGB로 구성된 이미지든, 고급 RGBW 픽셀로 구성된 이미지든, 모든 이미지는 저장하고 전송하거나 불러와야 합니다. 그리고 이는 디지털 이미징의 필수 요소인 압축으로 이어집니다.

무손실 압축과 손실 압축

서로 연결된 디지털 세상에서 방대한 양의 데이터를 효율적으로 저장하고 전송하는 능력은 매우 중요합니다. 특히 고화질 이미지를 표준으로 사용하는 현대 시대에서 이미지 압축은 매우 중요합니다. 이미지 압축의 중요성은 단순한 저장과 전송을 넘어 컴퓨터 비전 모델의 효율성 및 효과와도 밀접한 관련이 있습니다.

1. **효율적인 훈련**

   압축된 이미지는 모델 훈련 시 I/O 오버헤드를 줄여주므로 특히 데이터 세트가 방대한 경우 프로세스 속도가 빨라집니다.

2. **전송 시간 감소**

   자율 주행이나 원격 수술과 같은 실시간 애플리케이션에서는 시각 데이터 전송 지연 시간이 매우 중요할 수 있습니다. 압축된 이미지는 더 빠른 전송을 보장하며, AI와 결합하면 더 빠른 의사 결정 프로세스를 촉진할 수 있습니다.

3. **컴퓨터 비전 모델 성능 향상**

   압축 이미지로 AI 모델을 훈련하면 때때로 더 강력한 모델을 만들 수 있습니다. 압축으로 인한 미세한 왜곡은 일종의 데이터 증강으로 작용하여 약간 품질이 저하된 이미지에서도 모델이 패턴을 인식하도록 학습시킬 수 있습니다.

이미지 데이터는 손실 압축 방법과 무손실 압축 방법 두 가지가 있습니다 먼저 무손실 압축부터 알아보겠습니다.

무손실 압축

무손실 압축은 이름에서 알 수 있듯이 정보의 손실 없이 데이터 사이즈를 줄이는 것을 말합니다. 압축이 풀리면 데이터는 원본과 동일한 원래 상태로 돌아갑니다. 이러한 원본 정보 보존은 압축 과정에서 일부 데이터가 손실되는 손실 압축과 무손실 압축을 구분합니다.

1. PNG

   휴대용 네트워크 그래픽(Portable Network Graphics)의 약자인 PNG 형식은 웹 그래픽과 이미지 무결성이 가장 중요한 모든 애플리케이션에 선호되는 형식입니다. 다른 포맷에 비해 PNG의 근본적인 장점은 무손실 압축을 사용한다는 점입니다. PNG의 핵심 속성은 다음과 같습니다.

   - **비트 심도**: PNG는 1비트(이진 이미지)에서 16비트(광범위한 색상 또는 그레이 스케일 제공)에 이르는 다양한 비트 심도를 지원합니다.
   - **색상 유형**: 그레이 스케일, RGB, 인덱스 컬러, 알파(투명도)가 있는 그레이 스케일, 알파가 있는 RGB가 포함됩니다. 이러한 옵션을 사용하면 간단한 아이콘이나 고품질 사진 등 다양한 용도로 PNG를 사용할 수 있습니다.
   - **투명도**: PNG의 대표적인 기능은 알파 채널 투명도를 지원하여 이진 투명/불투명뿐만 아니라 미묘한 투명도 수준을 설정할 수 있다는 점입니다.

PNG 이미지는 대표적으로 **디플레이트**(deflate) 알고리즘을 사용합니다. 디플레이트는 LZ(렘펠-지브)77과 허프만 코딩 알고리즘을 사용합니다.

❶ LZ77

LZ77의 핵심은 데이터에서 반복되는 시퀀스를 찾는 것입니다. 반복되는 시퀀스를 찾으면 반복된 부분을 이전 인스턴스에 대한 참조로 대체합니다. 이 참조에는 이전 시퀀스와의 거리와 반복되는 바이트 수를 나타내는 길이라는 두 가지 값이 포함됩니다. 이미지에 파란색 픽셀의 긴 가로줄이 있는 경우, 각 픽셀 값을 저장하는 대신 LZ77은 값을 한 번 저장한 다음, 후속 반복에 참조를 사용합니다.

❷ 허프만 코딩

허프만 코딩은 엔트로피 기반 코딩 시스템입니다. 이 코딩 시스템은 값(이 경우 픽셀 값 또는 LZ77 참조)의 빈도를 분석하여 빈도가 높은 값에 더 짧은 코드를 할당합니다. LZ77이 반복되는 시퀀스를 참조하여 사이즈를 줄인 후, 허프만 코딩은 더 나아가 일반적인 시퀀스를 더 짧은 코드로 표현합니다.

정리하자면 LZ77로 데이터의 반복 시퀀스를 효과적으로 축소한 뒤, 허프만 코딩이 그 뒤를 이어 빈도가 높은 패턴을 짧은 코드로 변환함으로써, 이미지 파일의 압축률을 극대화합니다 이런 프로세스를 통해 이미지 데이터를 최대한 효율적으로 압축할 수 있습니다.

손실 압축

이름에서 알 수 있듯이 손실 압축은 압축 과정에서 원본 데이터가 일부 손실되는 압축입니다. 원본 정보의 모든 비트를 보존하는 대신 시각의 한계를 활용하여 감각으로 인지하기 어려운 특정 데이터를 제거합니다. 이를 통해 무손실 방식에 비해 훨씬 더 높은 압축률을 달성할 수 있습니다. 특히 이미지, 비디오, 오디오와 같은 미디어에서 유용한 압축 방식입니다.

1. JPEG

JPEG(Joint Photographic Experts Group) 포맷은 손실 압축의 가장 유명한 확장자입니다. JPEG를 기준으로 손실 압축을 설명하겠습니다.

JPEG는 주로 정지 이미지, 특히 사진을 압축하기 위해 설계되었습니다. 선 그림이나 글자처럼 가장자리가 날카롭고 대비가 있는 이미지에는 아티팩트(artifact)가 생성되기 때문에 적합하지 않습니다.

그림 2-3 이미지 아티팩트 예시

이 형식은 변환, 양자화 및 엔트로피 코딩의 조합을 사용합니다.

❶ RGB에서 YCbCr로 변환: 서브 샘플링

RGB 형식의 이미지가 먼저 YCbCr 색상 공간으로 변환됩니다. 여기서 Y는 휘도(밝기)를 나타내고, Cb와 Cr은 색차(색상)를 나타냅니다.

사람의 시각은 색상보다 빛에 더 민감합니다. 따라서 눈에 띄는 이미지 품질의 손실 없이 색상을 더 많이 압축할 수 있습니다.

❷ 양자화

그런 다음 복잡하고 어려운 픽셀의 패턴은 인간의 시각으로 모두 확인하기 어렵기 때문에 이를 단순화하게 됩니다. 이를 양자화(quantization)라고 하며 이 단계에서는 '손실' 압축으로 인한 손실이 발생합니다. 디테일을 얼마나 유지하거나 제거할지 선택할 수 있습니다. 많이 제거하면 이미지 파일은 작아지지만 약간 이상하게 보일 수 있습니다. 조금만 제거하면 이미지가 더 좋아 보이지만 파일 용량이 커집니다.

그림 2-4 양자화 예시

1.31	4.52	−3.17	40.9
−1.37	4.98	3.17	−4.09
−1.36	45.2	3.17	4.03
1.31	31.2	31.7	−40.9

양자화

1	4	−3	40
−1	4	3	−4
−1	45	3	4
1	31	31	−40

❸ **엔트로피 코딩**

이미지 세부 정보를 단순화한 후에는 허프만 코딩과 같은 방법을 사용하여 나머지 세부 정보를 효율적으로 저장합니다. 또한 양자화된 계수는 실행 길이로 인코딩되고 허프만 코딩을 사용하여 추가로 압축됩니다.

손실 압축은 약간의 디테일 손실이 육안으로 바로 감지할 수 없는 사진에 매우 효과적입니다. 그러나 압축 정도를 조절할 수 있으므로 압축률이 높을수록 파일 사이즈는 작아지지만 아티팩트도 더 뚜렷해집니다. 원하는 파일 사이즈를 달성하면서 이미지 정보를 유지하는 균형을 찾는 것이 중요합니다.

이미지를 편집하여 JPEG로 저장할 때마다 손실 압축이 다시 적용되므로 편집을 계속할수록 품질이 저하될 수 있습니다. 이러한 이유로 컴퓨터 비전 연구자들은 전처리 및 모델링 과정에서 무손실 형식인 PNG로 작업하고 최종 결과물을 배포하기 위해 JPEG로 저장하는 경우가 많습니다.

2.1.2 색 공간 이해하기

이미지 구조와 압축의 기본을 이해했으니 이제 더 디테일한 색 공간에 대해 알아볼 차례입니다. 우리 주변에서 볼 수 있는 수많은 색상은 어떻게 표현할까요? 다양한 포맷과 디바이스는 색상을 어떻게 해석하고 표시할까요? 이번에는 그레이 스케일에서 CMYK에 이르기까지 이러한 색 공간이 이미지 표현, 처리 및 인식에서 어떻게 중추적인 역할을 하는지 살펴볼 것입니다.

색 공간

픽셀은 이미지의 가장 작은 구성 요소로, 밝기와 색에 대한 정보를 가지고 있습니다. 그렇다면 이 색 정보는 어떻게 저장되고 처리가 될까요? 바로 여기서 색 공간의 역할이 시작됩니다.

그레이 스케일

그레이 스케일의 개념은 복잡한 스펙트럼 분포보다는 빛의 강도에 초점을 맞춥니다. 세상에 존재하는 생생한 색조를 포착하지는 못하지만, 색상을 구별하는 것보다는 명암과 구조적 세부 사항을 강조하는 응용 분야에서 여전히 기본적인 표현 방법으로 사용됩니다. 그레이 스케일의 계보는 이미지를 흑백의 음영으로 표현하던 사진의 초기 시대로 거슬러 올라갑니다. 1830년대 최초의 사진

술, 즉 다게레오타입은 은판에 이미지를 캡처하는 프로세스를 사용하여 정교한 디테일의 흑백 이미지를 만들어냈습니다.

그림 2-5 다게레오 사진기

현재 그레이 스케일 사진들은 그 자체로 하나의 예술 형식이 되었고, 안셀 애덤스(Ansel Adams)와 같은 사진가들은 컬러가 아닌 빛과 그림자를 통해 이야기를 전달하는 상징적인 이미지를 만들기 위해 그레이 스케일을 사용했습니다.

그림 2-6 그레이 스케일

디지털 이미지 처리에서 그레이 스케일은 각 픽셀에 다양한 색상 스펙트럼이 없습니다. 대신 밝기를 나타내는 값이 주어지며, 0은 빛이 없는 상태(검은색)를 나타내고 255는 최대 밝기(흰색)를 나타냅니다. 이러한 단순성 덕분에 많은 계산 시나리오에서 저장 공간을 줄이고 처리 속도를 높일 수 있습니다. 또한 가장자리 감지나 텍스처 분석과 같은 특정 이미지 처리 작업의 경우 색상이 방해가 될 수 있으므로 그레이 스케일이 더 적합합니다.

RGB

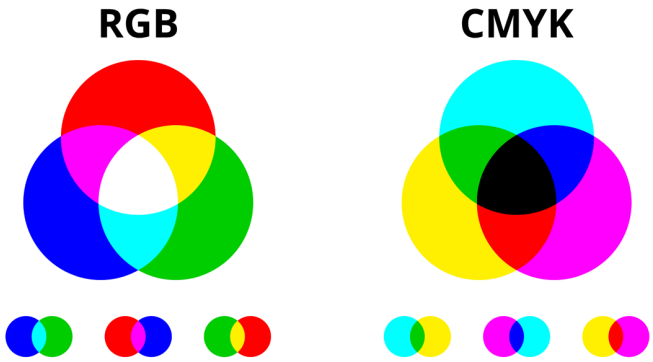
빛의 세계에는 특정한 세 가지 기본 색상이 있습니다. 바로 빨강(Red), 초록(Green), 파랑(Blue)입니다. RGB라는 이름은 바로 이 세 가지 색상의 첫 글자를 따서 명명된 것입니다. 이 세 가지 색상은 우리가 일상에서 눈으로 보는 거의 모든 색을 표현해낼 수 있는 기본이 됩니다.

디지털 세계에서 이 세 가지 색상은 **채널**이라는 이름으로 표현됩니다. 채널은 각 색상의 정보를 독립적으로 담고 있는 정보의 통로, 또는 저장 공간이라고 생각하면 됩니다. 따라서 RGB 이미지에서는 빨간색의 정보를 담고 있는 빨간색 채널(Red channel), 초록색의 정보를 담고 있는 초록색 채널(Green channel), 파란색의 정보를 담고 있는 파란색 채널(Blue channel)로 구성됩니다.

디지털 세계, 특히 디지털 화면과 카메라 내부에서, 이 RGB 색 공간은 매우 중요한 역할을 합니다. 모니터나 텔레비전 화면에서는 수많은 작은 픽셀들이 이루어져 있고, 각각의 픽셀은 빨강, 초록, 파랑 빛의 조합으로 원하는 색상을 표현합니다.

예를 들어 RGB에서 어떤 픽셀이 빨간색 채널의 정보에 따라 빨간빛을 최대 강도로 발하고, 초록색과 파란색 채널에서는 빛을 발하지 않으면 그 픽셀은 빨간색으로 보이게 됩니다.

그림 2-7 RGB, CMYK

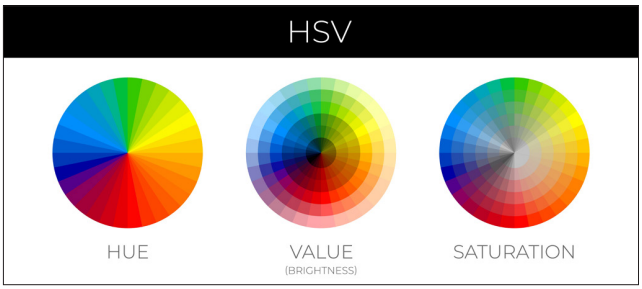


CMYK

전자 화면의 발광과는 달리 인쇄물에는 색 표현에 대한 다른 접근 방식이 필요하며, 바로 이 점에서 CMYK가 중요한 역할을 합니다. 인쇄 산업에서 시작된 CMYK는 빛을 빼는 원리로 작동합니다. 청록색, 자홍색, 노란색 잉크는 종이에서 반사된 흰빛에서 색을 빼는 데 사용되며, 이를 결합하면 다양한 색상을 표현할 수 있습니다. 그러나 잉크의 불완전성으로 인해 이들을 조합해도 완벽한 검은색이 나오지 않는 경우가 많으므로 키(검정) 잉크가 포함됩니다.

HSV

❯ 그림 2-8 HSV



인간의 지각에서 파생된 HSV 모델은 우리가 자연스럽게 색을 묘사하는 방식과 더욱 일치하도록 설계되었습니다. 사람들은 색을 떠올릴 때 빨간색의 불 같은 느낌이나 파란색의 차분함 등 특정 음영이나 색조를 떠올리는 경우가 많습니다. HSV 공간은 색상 경험을 세 가지 요소로 나누어 이러한 뉘앙스를 포착합니다.

- **Hue(색조)**: 우리가 보고 있는 색의 유형 또는 품질을 나타냅니다. 빨강, 파랑, 초록과 같은 색의 기본 본질을 나타냅니다.
- **Saturation(채도)**: 색조의 강도 또는 선명도를 설명합니다. 채도가 높은 색상은 풍부하고 충만해 보이며, 채도가 낮은 색상은 색이 바랜 것처럼 보이거나 흐릿해 보입니다.
- **Value(값, 밝기)**: 색상의 밝기 또는 어두움을 알려줍니다. 값이 높으면 색상이 밝고 생생하게 보이고 값이 낮으면 어둡거나 칙칙하게 보입니다.

특정 감정이나 반응을 불러일으키려는 그래픽 디자인 및 아트에서 HSV 모델을 이해하면 특히 유용할 수 있습니다. 예를 들어 채도를 조정하면 장면의 생동감이나 차분한 느낌을 결정할 수 있습니다.

비트

컴퓨팅 세계에서 '비트'는 데이터의 가장 기본적인 단위입니다. 비트는 0 또는 1, 두 가지 값 중 하나를 가질 수 있습니다. 텍스트, 오디오, 비디오, 이미지 등 모든 디지털 정보는 그 핵심이 비트의 연속으로 표현됩니다. 비트는 이진수라고 부르기도 합니다.

디지털 이미지의 비트 심도

'비트 심도'는 이미지의 각 픽셀에 대한 컬러 또는 그레이 스케일 정보를 표현하는 데 사용되는 비트 수를 나타냅니다.

- **1비트**: 두 가지 가능한 값(0 과 1)을 제공합니다. 그레이 스케일 이미지에서 회색 음영이 없는 흑백 이미지입니다.

그림 2-9 이미지 1비트 예시

- **8비트**: 0(검정)부터 255(흰색)까지 256개로 표현 가능한 값입니다. 이를 통해 부드러운 그라데이션과 그레이 스케일 스펙트럼을 세밀하게 표현할 수 있습니다.

그림 2-10 이미지 8비트 예시

- **16비트**: 0부터 65,535까지 표현 가능하며, 이는 더욱 부드러운 색을 표현하여 의료 영상과 같이 미묘한 음영 차이가 중요한 전문 환경에서 사용됩니다.

❯ 그림 2-11 이미지 16비트 예시

- **24비트(채널당 8비트)**: 256×256×256 = 16,777,216개의 가능한 값입니다. 이를 '트루 컬러'라고도 하며 대부분의 애플리케이션에 적합한 방대한 색상을 표현할 수 있습니다.
- **48비트(채널당 16비트)**: 좀 더 광범위한 색상 값을 제공하여 디지털 아트 및 고화질 사진 등 정확한 색상 표현이 중요한 전문 애플리케이션에 유용합니다.

비트 심도의 중요성

- **이미지 품질**: 비트 심도가 높을수록 그라데이션이 더 부드러워져 밴딩 아티팩트가 줄어듭니다.
- **파일 사이즈**: 비트 심도가 높을수록 픽셀당 더 많은 데이터를 저장할 수 있으므로 파일 사이즈가 커집니다.
- **편집의 유연성**: 특히 컬러 이미지에서 비트 심도가 높을수록 이미지 품질 저하 없이 편집할 수 있는 공간이 더 넓어집니다. 예를 들어 더 높은 비트 심도를 사용하는 RAW 포맷으로 촬영하는 사진작가는 포스트 프로덕션에서 노출이나 컬러 밸런스를 좀 더 효과적으로 조정할 수 있습니다.
- **특수 이미징**: 천체 사진이나 의료 영상과 같은 특정 애플리케이션에서는 디테일 손실 없이 필요한 모든 데이터를 캡처하기 위해 특정 비트 심도가 필요할 수 있습니다.

비트 심도는 디지털 이미징에서 중요한 개념입니다. 비트 심도는 색상이나 그레이 스케일 음영을 표현하고 구분할 수 있는 정밀도를 결정합니다. 웹 브라우저나 일반 사진 촬영과 같은 일상적인 애플리케이션에서는 표준 비트 심도를 고수하는 경우가 많지만, 전문 이미징 분야에서는 최대한 정확하고 세밀하게 디지털로 표현하기 위해 더 높은 비트 심도를 지속적으로 요구하고 있습니다.