

모바일프로그래밍기초 산출물 보고서

Image to WebP Converter

정보융합학부 2024404060 강민혁

I. 서론

최근 몇 년간 디지털 이미지의 사용량이 폭발적으로 증가함에 따라, 고효율 이미지 포맷에 대한 수요도 크게 늘어났다. 특히, 페이지 로딩 속도와 데이터 사용량, 사용되는 서버 자원을 최소화 하는 것이 주요 목표인 웹 환경에서 고압축률과 고품질을 동시에 만족하는 이미지 포맷을 요구하고 있다. 이러한 요구에 부응하여 Google에서 기존 JPEG, PNG와 같은 품질을 가지지만, 더 우수한 압축 효율을 가진 WebP 포맷을 개발하여 기존 포맷들을 대체하려 하는 중이다.



20240529_131525.webp
May 29 13:15 12.63 MB



20240529_131525.jpg
May 29 13:15 12.63 MB

[그림1] 원본 이미지와 확장자 수정으로 변경한 이미지

안드로이드는 파일의 확장자를 변경하는 것으로 이미지를 WebP형식의 이미지로 변환할 수 있다. 하지만, 이러한 방식으로 변환할 경우, 용량은 같지만 품질은 비슷하거나, 오히려 떨어지는 것을 볼 수 있다. WebP의 본 목적인 훌륭한 품질과 적은 용량이라는 장점이 사라지는 것이다.

본 프로젝트는 이러한 WebP 포맷의 장점을 그대로 유지하며 사용자가 손쉽게 다양한 이미지 파일을 WebP 포맷으로 변환할 수 있도록 하는 안드로이드 애플리케이션을 개발하는 것을 목표로 한다. 이 애플리케이션은 사용자 친화적인 직관적인 유저 인터페이스를 제공하며, 다양한 해상도와 이미지 품질 옵션을 지원하여 사용자가 원하는 대로 이미지를 최적화할 수 있도록 설계하였다.

본 보고서에서는 이 애플리케이션의 개발 과정을 상세히 설명하고, 개발 중에 직면했던 주요 과제와 그 해결 방안, 그리고 애플리케이션의 성능과 효율성에 대한 평가 결과를 제시한다.

II. 배경

1. WebP

WebP는 2010년 웹에서의 사용을 목표로 하는 구글이 개발한 이미지 포맷으로 VP8 비디오 코덱을 기반으로 하여 손실 압축은 블록 기반 예측 코딩과 변환/양자화 과정을 거쳐 데이터를 압축하며, 무손실 압축은 변환 기반 압축과 레퍼런스 매칭 기법을 사용한다. 일반적으로 JPEG보다 25~34%, PNG보다 26% 더 효율적인 압축률을 보여준다.¹⁾ 손실 및 무손실 압축과 알파 채널, 애니메이션까지도 지원하여 모든 이미지 포맷을 대체할 수 있다. 하지만, WebP 아직 널리 보급되지 않았기에 Apple 및 임베디드용 ap에 렌더링에 기반이 되는 VP8 하드웨어 디코더가 없는 경우가 대부분이며, 이러한 경우에는 렌더링 과정에서 cpu/메모리의 점유율을 과점하는 경우가 생긴다.

본 프로젝트에서는 WebP가 안정적으로 렌더링 될 수 있는 API 28 (android Pie)을 바탕으로 개발하였다.

2. Bitmap

Bitmap은 디지털 이미지를 구성하는 가장 기본적인 데이터 구조 중 하나로, 화면에 표현하고자 하는 이미지를 픽셀 단위로 나누어 각 픽셀의 색상 정보를 배열 형태로 저장하는 방식을 취한다. Bitmap 데이터를 직접 조작하는 방식으로 이미지 수정 및 변환 등을 진행한다. 이미지를 한 포맷에서 다른 포맷으로 변환할 때, Bitmap은 중간 단계의 역할을 한다. JPEG를 WebP로 변환할 때, JPEG 이미지 파일을 DCT 알고리즘을 통해 Bitmap 형태로 디코딩 하고, WebP 압축 알고리즘을 통하여 인코딩된다.²⁾

본 프로젝트에서는 graphics 패키지의 Bitmap 클래스를 사용하여 이미지 크기, 품질, 포맷 변환을 처리하였다.

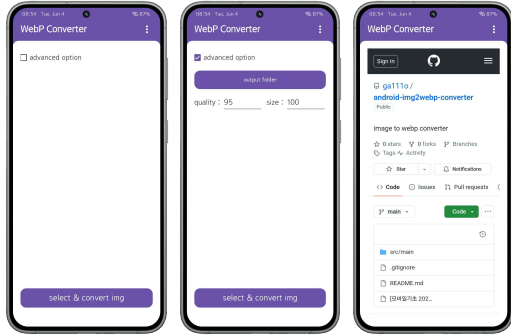
1) An image format for the Web | WebP. (2024, April 15). Google for Developers. Retrieved May 29, 2024, from <https://developers.google.com/speed/webp>

2) Hudson, G., Léger, A., Niss, B., & Sebestyén, I. (2017). JPEG at 25: Still Going Strong. IEEE MultiMedia, 24(2). 10.1109/MMUL.2017.38

III. 애플리케이션 개발

수업시간에 배운 내용을 기반하였으며, 기획한 내용을 모두 구현할 수 있도록 수업시간에 배운 내용 외의 개념 및 기술들을 사용하였다.

1. Main 화면 구현



[그림2] Main 화면

- (1) CustomAppBar: 더욱 디테일한 설정을 할 수 있도록 android theme에서 AppBar를 설정하는 것이 아닌, LinearLayout을 사용하여 임의의 AppBar를 구현하였다. 우측 아이콘을 클릭시, onClick을 통해 showPopupMenu가 호출되고, 제작자 정보와 프로그램 소스코드를 볼 수 있도록 하는 아이템이 나온다.
- (2) WebView: 프로그램의 소스코드를 볼 수 있는 WebView로, Custom AppBar 바로 아래에 위치한다. 평소에는 Visibility가 gone으로 설정되어 있다가 Popup item을 클릭하면 Visible로 설정되어 이미지 변환 관련된 화면을 덮는다. onBackPressed()를 전역으로 설정하고, WebView가 Visible 상태에 뒤로가기 버튼을 클릭하면, WebView의 Visibility가 gone으로 설정되게 하는 방식으로 뒤로가기를 구현하였다.
- (3) advancedOptionsCheckBox: CheckBox를 이용하여 이미지 변환과 관련한 추가적인 설정을 할 수 있는 layout의 Visible 설정을 가능하도록 하였다. 이를 통해 대부분의 경우에서 크게 신경쓰지 않아도 되는 설정들을 숨겨 시야분산을 최소화하였다. 또한, CheckBox를 눌러 layout의 Visible 설정을 변경할 때, 상하로 이동하는 애니메이션 효과와 페이드인 효과를 넣어 부드러운 인터랙션이 가능토록 하였다.
- (4) outputFolderButton: output folder를 설정할 수 있는 Button으로, 버튼을 클릭하면 사용자가 디렉토리를 선택할 수 있는 시스템 UI를 열 수 있는 Intent를 생성하고, 사용자가 디렉토리를 선택하면, 해당 디렉토리에 대한 읽기 및 쓰기 권한을 얻으며 결과가 startActivityForResult로 반환되도록 하였다.

- (5) imgQualityEditText: 이미지의 퀄리티를 설정할 수 있도록 하는 EditText로, bitmap.compress()에 이미지의 품질을 정하는 두 번째 인자로 들어간다. EditText에 유효한 float 값이 없을 경우에는 try-catch문을 통해 Toast message를 이용하여 Invalid input가 출력되도록 한다.
- (6) imgScaleEditText: 이미지의 크기를 설정할 수 있도록 하는 EditText로, EditText에 유효한 float 값이 없을 경우에는 try-catch문을 통해 Toast message를 통해 Invalid input가 출력되도록 한다. 이후 Bitmap 객체로 변환된 이미지와 받은 float값을 resizeImage()에 파라미터로 넣고, transAndDownload()를 호출하여 이미지를 변환 및 다운로드 한다. 만약 이미지 load 및 resize 과정에서 오류가 발생한다면, try-catch문을 통해 IOException을 처리한다.
- (7) selectAndTransButton: 이미지를 선택하고 변환할 수 있도록 하는 Button이다. Button을 클릭하면, openImageChooser()을 통해 이미지를 선택할 수 있도록 한다. 이미지가 선택되었다면 onActivityResult()를 호출하며 이미지 변환이 진행되며, 최종적으로는 다운로드 폴더 혹은 사용자가 선택한 폴더에 변환된 이미지가 저장되도록 한다.

가장 많이 사용할 Button을 하단에 배치하여 주로 엄지손가락으로 조작을 하는 스마트폰이 디바이스에 최적화된 유저 경험을 제공한다.

2. 주요 기능 구현

외부 저장소 권한 요청







- requestPermission(), checkPermission(), onRequestPermissionsResult(): 외부 저장소 쓰기 권한 요청 및 권한 확인한다. 전역으로 선언한 PERMISSION_REQUEST_CODE를 통해 권한을 판단하며, 권한이 있다면 true, 없다면 false를 반환한다. 권한이 거부된다면 Toast message를 통해 권한이 필요함을 알린다.
- advancedOptions(), onBackPressed(), showPopupMenu(), getAndProcessImgQuality(), getAndProcessImgScale(), resizeImage(), openImageChooser(): 각각 '가.'의 (3), (2), (2), (5), (6), (6), (7)을 처리하는 함수들이다.
- onCreate(): activity_main을 표시하며 메인 화면의 CheckBox와 Button의 상태 처리 및 권한 설정을 관리한다.
- onActivityResult(): 이미지를 선택하기 위해 시작된 인텐트로부터 결과를 받았는지 확인 후, 받았다면 선택된 이미지의 URI를 selectedImageUri에 저장한 후 getAndProcessImgScale()를 호출하여 이미지 처리를 진행한다. 만약 그렇지 않다면, 폴더를 선택하도록 하고, 유저가 폴더를 선택하였다

면, `getContentResolver()`를 통해 해당 폴더에 반영구적인 읽기 및 쓰기 권한을 얻은 후, 진행하도록 한다.

- `transAndDownload()`: 임의의 파일을 만들고, `Bitmap` 객체를 `WEBP` 형식으로 압축하여 저장한다.
 1. 폴더 URI 확인: `selectedFolderUri`가 null인 경우, 기본 다운로드 폴더의 URI를 설정하고 사용자에게 `Toast message`를 통해 알린다.
 2. 이미지 위치 및 이름 확인: `MediaStore.Images.Media.DISPLAY_NAME` 컬럼을 사용하여 쿼리를 실행하여 `selectedImageUri`를 통해 이미지 위치와 이름을 가져온다.
 3. 파일 생성: `DocumentFile.fromTreeUri`를 사용하여 선택된 URI로부터 `DocumentFile` 객체를 생성하고, `pickedDir.createFile`을 통해 새로운 파일을 생성한다. 파일 형식은 `image/webp`로 하며, 이름은 가져온 이미지 이름으로 한다.
 4. WebP로 압축: `getContentResolver().openOutputStream`을 통해 `ContentResolver`를 사용하여 파일에 데이터를 쓸 수 있도록 하고, `bitmap.compress`를 사용하여 이미지를 `WEBP` 형식으로 압축한다.

중간에 발생할 수 있는 모든 예외를 try-catch문을 통해 처리할 수 있도록 한다. 예외가 발생할 경우 `StackTrace`를 출력하도록 한다.

IV. 테스트

	20240529_131525.jpg May 29 17:48	12.68 MB
	1000021372.webp May 29 17:48	10.79 MB
	20240529_175036.jpg May 29 17:50	5.74 MB
	1000021406.webp May 29 17:51	1.91 MB
	20240529_175220.jpg May 29 17:52	5.79 MB
	1000021410.webp May 29 17:55	253 KB

[그림3] 원본 이미지와 변환 후 이미지의 용량











[그림4] 원본 이미지와 변환 후 이미지

case 1. 8160x6120 해상도의 이미지로, 변환 전 12.68MB에서 변환 후 10.79MB의 용량으로 약 14.9%의 용량이 감소한 것을 확인할 수 있다. 그리고, 각 이미지를 10배 크롭하여 확인하였을 때, 눈에 띄는 차이점이 존재하지 않는 것을 알 수 있다.

case 2. 4000x3000 해상도의 이미지로, 변환 전 5.74MB에서 변환 후 1.91MB로 약 66.7%의 용량이 감소한 것을 확인할 수 있다.

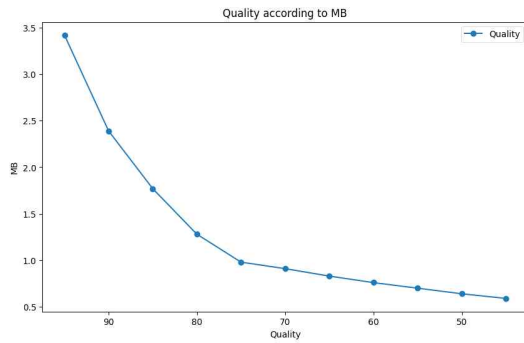
case3. 4000x3000 해상도의 이미지로, 변환 전 5.79MB에서 변환 후 253KB로 약 95.6%의 용량이 감소한 것을 알 수 있다.

모든 케이스에 대해 이미지 용량이 줄었다는 것을 알 수 있으며, case 2와 case 3 알고리즘에 따라 압축되는 것이기에, 같은 해상도라 하더라도 변환 및 압축 후의 용량은 매우 크게 차이가 난다는 것을 알 수 있다.

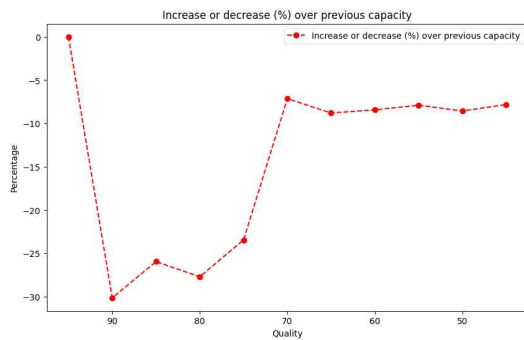
	20240529_181311.jpg May 29 18:13	7.87 MB		1000021427 (5).webp May 29 18:23	0.91 MB
	1000021427.webp May 29 18:23	3.42 MB		1000021427 (6).webp May 29 18:23	830 KB
	1000021427 (1).webp May 29 18:23	2.39 MB		1000021427 (7).webp May 29 18:24	763 KB
	1000021427 (2).webp May 29 18:23	1.77 MB		1000021427 (8).webp May 29 18:24	708 KB
	1000021427 (3).webp May 29 18:23	1.28 MB		1000021427 (9).webp May 29 18:24	644 KB
	1000021427 (4).webp May 29 18:23	0.98 MB		1000021427 (10).webp May 29 18:25	596 KB

[그림4] quality만 달리하여 변환한 이미지들

위에서부터 아래로, 왼쪽에서 오른쪽 순서대로 원본 이미지와 변환 quality 95%부터 45%까지 5%씩 줄이며 변환한 이미지들이다. 원본 이미지에서 변환 이미지는 용량이 약 56.5% 감소한 것을 알 수 있으며, 나머지는 최대 30%에서 10% 까지 선형적으로 감소하는 용량이 감소한 것을 알 수 있다. 이를 그래프로 나타내면 아래와 같다.



[그림5] quality에 따른 용량



[그림6] quality에 따른 용량 증감률

<그림5>와 같이 quality에 따른 용량은 로그함수의 개형을 따르며, quality가 70 혹은 그 이하부터는 용량 감소율이 10% 미만이 되는 것을 알 수 있다.

즉, 충분한 품질과 적당한 용량을 원한다면 quality를 95로 설정하고, 적당한 품질과 적은 용량을 원한다면 quality를 75로 설정하는 것이 가장 이상적이라 판단된다.

V. 결론

본 프로젝트에서는 디지털 이미지의 사용량 증가와 고효율 이미지 포맷에 대한 수요 증가에 대응하여 다양한 이미지 파일을 WebP 포맷으로 사용자가 손쉽게 변환할 수 있도록 하는 애플리케이션을 개발하였다.

테스트 결과, 이미지의 용량이 줄었음을 확인하였으며, 같은 해상도라 하더라도 이미지에 따라 변환 및 압축 후의 용량은 매우 크게 차이가 난다는 것을 알 수 있었다. 또한, WebP 변환 시에 이미지 품질에 따른 용량은 로그함수의 형태를 따르며, 품질이 70 이하부터는 용량 감소율이 10% 미만이 되는 것을 확인하였다. 이러한 데이터를 통해 사용자가 자신의 이미지 파일의 용량을 관리하거나 웹 서버 운영자가 서버 부담 등을 이유로 이미지 파일을 변환 및 압축할 때의 가이드라인을 제시하여 디지털 미디어의 효율적인 관리와 활용에 기여할 것으로 기대된다.

Reference

- [1] Hudson, G., Léger, A., Niss, B., & Sebestyén, I. (2017). JPEG at 25: Still Going Strong. *IEEE MultiMedia*, 24(2). 10.1109/MMUL.2017.38
- [2] *An image format for the Web / WebP*. (2024, April 15). Google for Developers. Retrieved May 29, 2024, from <https://developers.google.com/speed/webp>
- [3] Google for Developers. (n.d.). *Bitmap*. Android Developers. Retrieved May 29, 2024, from <https://developer.android.com/reference/android/graphics/Bitmap>
- [4] Akshay. (2023, September 5). *TutorialsAndroid/FilePicker: Android Library to select files/directories from Device Storage*. GitHub. Retrieved May 29, 2024, from <https://github.com/TutorialsAndroid/FilePicker>