<페이지터닝 시스템: 자동 악보넘김 서비스>

20221805 이은혜, 20221809 이채은, 20223078 서혜원

1. 요약

악기를 연주하다 악보를 넘길 때 많은 불편함이 있어서 그것을 해소하기 위한 시스템입니다. 시중에 나와있는 서비스들의 단점을 보완하기 위해 사용자가 사용할 악보의 파형과 사용자의 전자기기로 입력된 사용자 연주의 파형을 비교하여, 악보의 한 페이지가 95%정도 채워질 때쯤 페이지를 자동으로 넘겨주는 페이지터닝 시스템입니다.

2. 대표 그림

3. 서론

[개발 배경]

기존의 악보 넘김 서비스를 사용해보면 사소하지만 연주를 방해하는 요소들이 있습니다. 시중에 나와있는 서비스를 살펴보면 모션을 취해서 악보를 넘기는 방식, 얼굴인식을 사용해서 악보를 넘기는 방식, 자동 스크롤 기능, 블루투스로 연결해서 패달을 밟아 페이지를 넘기는 페이지터너를 이용하는 방식이 있습니다. 기존의 서비스를 사용해본 결과 연주하면서 모션을 취하는 행위가 몰입하는 연주자들의 연주를 방해한다고 느꼈습니다. 개인적인 경험으로 밴드에서 세션연주를 맡고 있는데, 악기를 연주하면서 악보를 넘길 때 한음을 날리고 악보를 넘겨야 하거나 악보넘김 타이밍을 못잡아서 뒷부분을 놓쳐버리는 경우가 많았습니다. 그래서 저희는 그 단점을 보완하기 위해 페이지터닝 시스템을 고안하였습니다.

[시중에 나와있는 악보 넘김 앱의 한계점]

시중에 나와있는 앱들은 Piascore, Forscore, musescore, mobile sheets 로 크게 4 가지가 있었습니다. 이 앱들이 각각 지원하는 서비스를 보면 얼굴인식을 사용하는 방식은 2 개의 앱이 지원하고 페이지터너는 모든 4 개의 앱이 지원하고, 자동스크롤 기능은 2 개의 앱이 지원하고 있음을 확인할 수 있었습니다.

모션을 취해서 악보를 넘기는 기능과 페이지 터너를 사용하는 방식은 위에서 말씀드린 것과 같이 연주 도중에 특정 제스처를 취해야하기 때문에 연주자의 몰입을 깬다는 단점이 있습니다.

심지어 페이지터너는 약 4-8 만원정도의 돈을 주고 구매해야하기 때문에 금전적 부담을 주기도 합니다. 얼굴인식을 사용하는 방법은 많은 사람들과 합주를 할 때 코로나시국에 맞춰 마스크를 끼면 얼굴인식을 사용하지 못한다는 단점이 있습니다. 얼굴인식을 사용하는 방식은 윙크하기, 입을 움직이기, 고개를 양옆으로 젓기 크게 이 세가지 방식으로 나뉩니다.

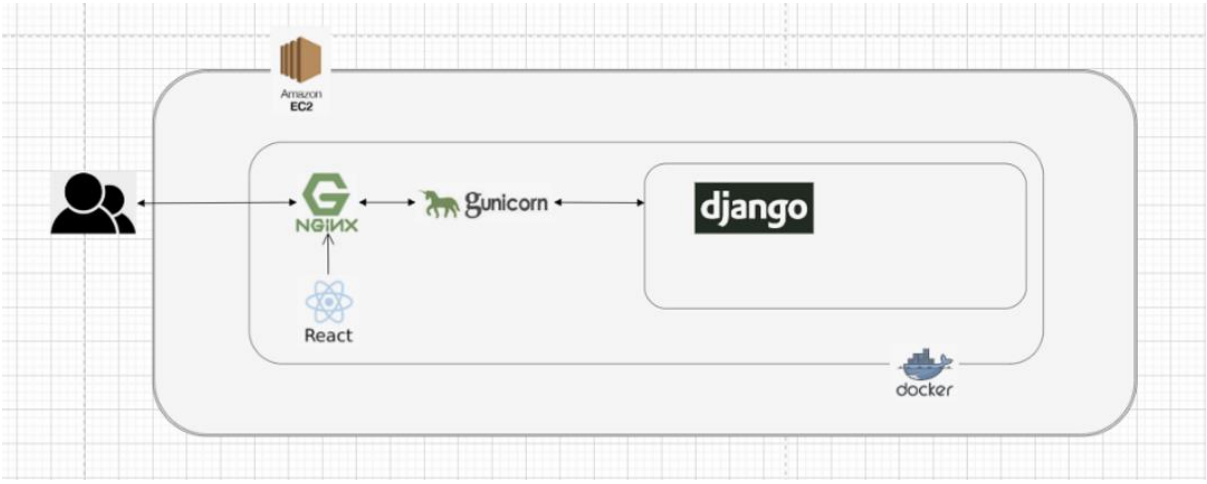
모션을 취할 때 마스크를 끼고 있지 않다고 해도 이 셋 다 동작을 아주 크게 해야만 인식이 잘 된다는 점도 사용자에게 불편을 주었습니다. 또한 카메라가 잠금버튼과 가까운 아이패드 사용시, 아이패드를 똑바로 두면 카메라에 얼굴이 잘 안 잡혀 모션인식이 안되고 이 때문에 아이패드를 거꾸로 두어도 잠금버튼이 눌려 화면이 꺼진다는 사례가 존재했습니다.

마지막으로 자동스크롤 기능은 일정한 속도로 악보가 내려간다는 점이 편리하긴 했지만 악보가 계속 움직인다는 점이 연주하면서 굉장히 신경쓰이는 점이였습니다. 악보와 건반을 번갈아가면서 봐야하는데 악보가 계속 움직이면 보던 지점을 놓쳐버리게 되는 경우가 많았기 때문입니다.

[문제 정의 및 극복 방안]

이러한 문제점들에 입각하여 저희는 사용자가 사용할 악보의 파형과 사용자의 전자기기로 입력된 사용자 연주의 파형을 비교하여, 악보의 한 페이지가 95%정도 채워질 때쯤 페이지를 자동으로 넘겨주는 페이지터닝 시스템을 구현하게 되었습니다.

4. 본론



[시스템 아키텍쳐]

페이지터닝 시스템을 하나의 완성된 서비스로 구현하게 된다면, 악보를 자동적으로 넘겨주는 기능을 탑재한 하나의 웹 서비스로 배포하려 합니다. 사용자가 웹에 회원가입을 하고 자신이 원하는 악보를 선택하면, 사용자의 전자기기 마이크를 통해 사용자의 연주를 바탕으로 악보를 언제 넘길지 판단하게 됩니다. 웹을 구성하기 위해서 프론트엔드는 리액트, 그리고 대부분 파이썬을 활용하려고 계획했기 때문에 백엔드는 장고로 구현할 것입니다. 그리고 서버의 경우에는 마찬가지로 파이썬 기반 프레임워크를 사용한 웹을 배포하게 되기 때문에 엔진 x 와 더불어서 위스키 서버 역할을 할 수 있는 구니콘 또한 사용할 것입니다.

개발 환경은 컨테이너 환경을 제공하면 서버를 항상 같은 상태로 만들 수 있기에 도커로 정했습니다. 저희의 핵심 기능인 자동적으로 악보를 넘길 때 사용하는 이미지 비교 모델 또한 파이썬으로 구현된 오픈소스를 활용하였습니다. 그래서 전체적인 시스템의 흐름을 살펴보자면 사용자가 전자기기의 마이크를 통해 피아노 소리를 입력했을 때, 리액트를 통해 그 소리를 백엔드로 전달하고 저희가 쓰려는 모델이 소리를 분석하며 그에 맞는 처리 작업을 장고를 통해 실행하게 됩니다. 그리고 해당 정보를 다시 프론트로 보내어서 악보가 넘어가는 화면이 사용자의 전자기기의 화면에 출력되게 됩니다.

[왜 파형인가?]

페이지터닝 시스템의 주요 기능은 사용자의 연주 소리를 파형으로 변환하고, 해당 악보를 올바르게 연주한 파형과 이미지 비교를 통해 사용자가 악보를 끝까지 연주했을 때, 자동적으로 악보가 다음 페이지로 넘어가게 하는 것입니다. 이번에는 왜 저희가 파형을 선택했고, 음원을 파형으로 변환하는 것과 이미지 비교를 통해 악보의 끝부분에서 다음 페이지로 넘긴다는 판단을 할 수 있도록 하는 라이브러리 및 모델 선정 이유에 대해서 설명하겠습니다. 소리의 음높이는 사람이 인식하는 음파의 진동수로 정의가 됩니다. 진동수가 높아지면 음높이는 올라가고,

진동수가 낮아지면 음높이는 내려가게 됩니다. 또한 소리의 파형은 소리의 세기, 높낮이, 맵시에 따라 그 형태가 달라집니다.

따라서 피아노 연주 시 악보의 각 마디의 음의 추세와 빠르기를 통해 같은 마디인지 아닌지 구별이 가능하기 때문에 저희는 음원을 파형으로 바꾸고, 악보의 파형과 사용자의 파형의 이미지 비교를 통해 일치율을 구했습니다.

[librosa 라이브러리 및 오픈소스 모델 선정 이유]

저희는 음원을 파형으로 바꾸고 파형의 이미지 유사도를 구하는 주요 기능만을 구현했는데, 음원을 파형으로 바꾸기 위해서는 librosa 라이브러리를 활용했고, 이 사진과 같은 코드의 이미지 비교 모델을 선정했습니다.

이 모델로 선정한 이유는 하나의 기준만 사용하여 이미지를 비교하는 것이 아니라, 상관관계, 카이제곱, 교차와 같이 다양한 수학적 기준을 통해 이미지의 일치율을 비교했고, 따라서 그 중에서 테스트를 해보며 하나를 선정해 보다 정확한 일치율을 구할 수 있었기 때문입니다.

[코드 설명]

첫번째 코드를 통해 librosa 라이브러리를 이용하여 음원 파일을 파형으로, 즉 소리를 이미지 형태로 나타내는 작업을 합니다.

```
import librosa
import librosa.display

import matplotlib.pyplot as plt

audio_path = 'wave.wav'

y, sr = librosa.load('C:\Users\dldms\Downloads\music\wave.wav')

print('sr:', sr, ', audio shape:', y.shape)
print('length:', y.shape[0]/float(sr), 'secs')

plt.figure(figsize = (10,5))
librosa.display.waveshow(y, sr=sr)
plt.ylabel("Amplitude")
plt.show()
```

정상적으로 악보를 연주했을 때의 파형과 연주자가 연주한 부분이 일치하는 지 판단해 일치한다면 프린트 '1'을 하여 일치한다고 알려주는 코드입니다. 이미지의 픽셀 값의 분포를 비교하여 서로 비슷하다면 유사한 이미지로 판단하고 분포가 다르면 서로 다른 이미지일 확률이 높다는 사실을 이용하여 이미지의 유사도를 측정합니다. 즉, 두 이미지의 히스토그램을 비교하는 것입니다. 여기서 OpenCV 는 히스토그램을 비교하여 두 이미지가 얼마나 유사한지 판단해주는 함수를 제공합니다. 이 알고리즘은 4 가지 방법으로 비교를 합니다. 간단하게 설명하자면 1 번과 3 번 방법에서는 1: 완전 일치, -1: 완전 불일치, 0: 무관계. 2 번과 4 번 방법은 0: 완전 일치, 무한대: 완전 불일치를 나타내 줍니다.

```python
import cv2, numpy as np
import matplotlib.pylab as plt

img1 = cv2.imread('../img/taekwonv1.jpg')
img2 = cv2.imread('../img/taekwonv2.jpg')
img3 = cv2.imread('../img/taekwonv3.jpg')
img4 = cv2.imread('../img/dr_ochanomizu.jpg')

cv2.imshow('query', img1)
imgs = [img1, img2, img3, img4]
hists = []
for i, img in enumerate(imgs) :
    plt.subplot(1,len(imgs),i+1)
    plt.title('img%d'% (i+1))
    plt.axis('off')
    plt.imshow(img[:,:,::-1])
    #---① 각 이미지를 HSV로 변환
    hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
    #---② H,S 채널에 대한 히스토그램 계산
    hist = cv2.calcHist([hsv], [0,1], None, [180,256], [0,180,0, 256])
    #---③ 0~1로 정규화
    cv2.normalize(hist, hist, 0, 1, cv2.NORM_MINMAX)
    hists.append(hist)


query = hists[0]
methods = {'CORREL' :cv2.HISTCMP_CORREL, 'CHISQR':cv2.HISTCMP_CHISQR,
           'INTERSECT':cv2.HISTCMP_INTERSECT,
           'BHATTACHARYYA':cv2.HISTCMP_BHATTACHARYYA}
for j, (name, flag) in enumerate(methods.items()):
    print('%-10s'%name, end='\t')
    for i, (hist, img) in enumerate(zip(hists, imgs)):
        #---④ 각 메서드에 따라 img1과 각 이미지의 히스토그램 비교
        ret = cv2.compareHist(query, hist, flag)
        if flag == cv2.HISTCMP_INTERSECT: #교차 분석인 경우
            ret = ret/np.sum(query)        #비교대상으로 나누어 1로 정규화
        print("img%d:%7.2f"% (i+1 , ret), end='\t')
```

피아노 음원과 다른 소리들의 음원을 비교했을 때 대부분 1 번과 3 번은 유사도를 제대로 구별하지 못하고 2 번과 4 번만이 유사도를 유의미하게 판별해주는 결과가 나왔습니다. 이 결과들을 통해 저희는 악보를 넘길 때의 기준을 정할 역할로 1 번과 3 번을 배제하였습니다.

그리고 피아노 곡을 느리게, 빠르게 연주하고, 중간중간 음 몇 개를 틀리게 하여 얻은 파형들을 비교해본 결과 2 번과 4 번에서 둘 다 0.04 값이 평균적으로 나와서 2 번과 4 번이 0.04 이하가 나왔을 때 연주자가 마지막 마디를 연주했다고 받아들이고 악보를 넘기도록 하였습니다.

[코드 정확성 검증 실험]

이에 더해 저희는 코드의 정확성을 검증하기 위해 한 가지 실험을 더 해보았습니다. imgs 라는 리스트에 img1, 2, 3, 4 를 저장하고 original 리스트에 img1, 2, 3, 4 를 저장했습니다. 그리고 오리지널 리스트의 요소들을 각각 쿼리 1, 2, 3, 4 로 지정하였습니다. 여기서 imgs 리스트에는 4 개의 요소가 있는데 각각 사용자가 연주한 악보의 첫번째 마디, 두번째 마디, 세번째 마디, 네번째 마디를 뜻합니다. original 리스트는 원본 악보의 마디들을 뜻하게 됩니다. 그러므로 악보의 첫번째 마디, 두번째 마디, 세번째 마디, 네번째 마디를 각각 쿼리 1,쿼리 2, 쿼리 3, 쿼리 4 로 지정한 것이 되고, 사용자가 연주한 파형과 비교해서, 즉 imgs 리스트의 요소들과 query 를 비교하여 마디 1, 2, 3, 4 를 판별해낼 수 있는지 확인한 것이라 할 수 있습니다. 아래 사진은 imgs 와 original 리스트에 있는 파형의 예시입니다. 결과를 보면 이렇게 마디 1 은 마디 1 이라고 하고 마디 2 는 마디 2 라고 하면서 잘 결과가 출력된 것을 알 수 있습니다.

```
import cv2, numpy as np
import matplotlib.pylab as plt
from time import sleep

img1 = cv2.imread("C:/Users/WHO A U/Desktop/p1f.png")
img2 = cv2.imread("C:/Users/WHO A U/Desktop/p2f.png")
img3 = cv2.imread("C:/Users/WHO A U/Desktop/p3f.png")
img4 = cv2.imread("C:/Users/WHO A U/Desktop/p4f.png")


imgs = [img1, img2, img3, img4]
original_imgs = [img1, img2, img3, img4]

hists = []
original_hists = []

for i, img in enumerate(imgs) :
    plt.subplot(1,len(imgs),i+1)
    plt.title('img%d'% (i+1))
    plt.axis('off')
    plt.imshow(img[:,:,::-1])
    #---① 각 이미지를 HSV로 변환
    hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
    #---② H,S 채널에 대한 히스토그램 계산
    hist = cv2.calcHist([hsv], [0,1], None, [180,256], [0,180,0, 256])
    #---③ 0~1로 정규화
    cv2.normalize(hist, hist, 0, 1, cv2.NORM_MINMAX)
    hists.append(hist)

for k, original_img in enumerate(original_imgs) :
    plt.subplot(1,len(original_imgs),k+1)
    plt.title('img%d'% (k+1))
    plt.axis('off')
    plt.imshow(original_img[:,:,::-1])
    #---① 각 이미지를 HSV로 변환
    original_hsv = cv2.cvtColor(original_img, cv2.COLOR_BGR2HSV)
    #---② H,S 채널에 대한 히스토그램 계산
    original_hist = cv2.calcHist([original_hsv], [0,1], None, [180,256], [0,180,0, 256])
    #---③ 0~1로 정규화
    cv2.normalize(original_hist, original_hist, 0, 1, cv2.NORM_MINMAX)
    original_hists.append(original_hist)


query1 = original_hists[0]
query2 = original_hists[1]
query3 = original_hists[2]
query4 = original_hists[3]
```

5. 결론

[파형 > 스펙트로그램 > Centroid 값]

저희가 준비할 때는 음악을 파악하기 위해서 파형이 제일 최선의 방법이라고 생각했지만, 막상 해본 결과 파형을 통해서는 대략적으로 리듬, 음, 노래의 흐름 정도는 따라갈 수 있었지만 멈춰진 이미지로 비교하기 때문에 노래를 빠르게 연주했을 때와 같이, 많은 변화를 주면 정확하게 인지하지 못하는 한계가 있었습니다.

이 한계점을 보완하기 위해서 파형을 대신할 다른 기술에 대해 찾아보던 중, 스펙트로그램에 대해 알게 되었습니다. 스펙트로그램은 시간상 진폭 축의 변화를 시각적으로 볼 수 있는 파형과 주파수상 진폭 축의 변화를 시각적으로 볼 수 있는 스펙트럼의 특징이 모두 결합된 구조로, 사람의 귀 또한 이와 유사한 매커니즘을 가지고 있습니다. 따라서 저희는 스펙트로그램을 활용하면 보다 음악의 특징을 자세하고 정확하게 집어낼 수 있겠다는 생각을 했지만, 음원 파일을 스펙트로그램으로 구현하려할 때, 지원하는 라이브러리나 오픈소스를 찾지 못해 구현하기가 어려웠습니다.

그래서 개선할 수 있는 다른 방법을 찾던 중 Centroid 값을 알게 되었습니다. Centroid 값이란 주파수와 관련이 있는 음악의 특징값입니다. 즉, 각 주파수에 주파수에서의 푸리에 변환의 크기를 곱한 것들을 모든 주파수에서 합친 것을 푸리에 변환의 크기, magnitude 의 총합으로 나눈 것이며, Centroid 값을 활용한다면 음의 높낮이, 음의 변화의 크기 등 음원데이터를 분류하기 위한 정보들을 모두 집어낼 수 있게 됩니다. 더불어서 Centroid 값은 파이썬을 활용해 구현이 가능했고, 그래서 저희는 Centroid 값을 사용하여 보다 정확하게 음원을 분석하고자 하는 결정을 내렸습니다.

현재는 Centroid 값을 활용하기로 했지만, 추후에 저희의 기술과 지식이 더 는다면, 스펙트로그램으로 음원을 변환하여 더욱 정확한 페이지터닝 시스템으로 개선하고자 합니다.

[하드웨어적으로 구현하기]

페이지터닝 시스템은 사용자의 연주를 사용자의 전자기기의 마이크를 통해 입력받습니다. 따라서 하드웨어적으로 음원을 입력받을 수 있도록 설계하는 것 또한 페이지터닝 시스템을 완성된 서비스로 만들기 위한 과제가 될 것입니다.

[페이지터닝 시스템의 핵심 이미지 분류 및 판단 기술이 적용될 수 있는 다양한 분야들]

1. 응급차가 사거리를 지나갈 때 신호등에 부착된 소리 센서가 소리의 파형을 입력받고 응급차가 접근하고 있다고 판단이 되면 신호등을 빨간불로 바꿔 사거리에 있는 차량들을 통제합니다. 이를 통해 현재 도로에 있는 차량들로 인해 지켜지지 않는 골든 타임을 지킬 수 있는 가능성을 높입니다.

2. 화장실 등 CCTV 설치가 불가능한 장소에 소리 감지 센서가 설치된다면 비명 등을 감지하여 위기상황에 빠른 대처가 가능할 것입니다.

3. 기계의 소리을 통해 작동 이상 여부를 모니터링합니다. 파형의 변화를 감지하여 초반에 기계의 이상을 감지하여 기계에 문제가 더 생기기 전에 유지보수 비용을 절반으로 줄이고 기계의 수명을 두 배로 늘릴 수 있을 것입니다.

6. 출처

Python 을 이용한 음악분류 & 추천을 위한 Feature 값 (1)Centroid | by KimJungeui | Medium

Getting to Know the Mel Spectrogram | by Dalya Gartzman | Towards Data Science

https://engineering.linecorp.com/ko/blog/voice-waveform-arbitrary-signal-to-noise-ratio-python/

OpenCV - 12. 이미지 유사도 비교, 사람 얼굴과 해골 합성, 모션 감지 CCTV (tistory.com)

AI 에게 어떻게 음성을 가르칠까? (kakaoenterprise.com)