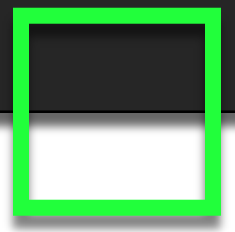
#수치 해석
#과제 10

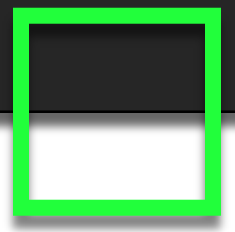
Discrete Fourier Transform

컴퓨터소프트웨어학부
2018008395 박정호



#1

구현



설명에 앞서…

이번 과제는 DFT를 이용해서 fabric pattern을 구별할 수 있는 recognition model을 구현하는 것으로, 우선 DFT를 사용하는 것이 필수였다. 다만 이를 직접 구현하는 것은 번거로울 뿐더러, 실행 속도 또한 너무 느리기 때문에, 라이브러리에서 제공하는 FFT함수를 사용했다.

사용한 라이브러리는 numpy.fft.fft2와 numpy.fft.fftshift이며, 각각 2차원에서의 DFT를 빠르게 수행하고, 이 과정에서 생긴 DC 성분을 중앙으로 shift하는 함수이다. DFT로 구한 패턴 정보의 시각화 과정에서, DC 성분이 중앙에 가는 편이 강의 자료에서 보인 패턴 성분과 유사하게 보일 것이기 때문에 일부러 shift를 수행했다.

이전 과제에서도 언급했지만 numpy는 pip만 설치하면 쉽게 설치 가능한 오픈 라이브러리이다.

모델링 – DFT를 통한 패턴 정보 분석

다음 챕터에서 어떤 패턴을 사용했는지에 대해 설명하겠지만, 이미지의 크기는 640*640이고, DFT에 사용할 블록은 64*64이기 때문에 무작위로 선정된 블록 하나에 대해서 구해진 DFT 결과만을 가지고 이미지 인식을 수행하기에는 무리가 있다고 판단했다.

따라서 10개의 무작위 블록을 선정해서, 이들의 DFT coefficient 를 각각 구한 후, 그 평균을 이미지의 패턴 정보로 저장했다. 블록의 위치에 따라 발생할 약간의 노이즈를 평균을 통해서 조절하고자 한 것이다.

물론 DFT의 결과물은 복소수 행렬이기 때문에, 각 항의 절댓값, 즉 magnitude 를 구해서 그 행렬을 패턴 정보로 사용했다. 시각화할 때는 이 magnitude가 0~255 사이에 속하지 않기 때문에, 우선 log 를 취해서 어느 정도 scaling 하고, 이 값을 다시 0~255로 linear하게 fitting 하는 것으로 시각화 문제를 해결했다.

실제로, log scaling 없이 바로 0~255로 linear fitting한 결과, DC 성분 이외에는 모두 검게 표시되는 것을 확인할 수 있었다.

모델링 – 패턴 인식 방법

필자는 여러가지 패턴 인식 방법을 시도해보았는데, 이 슬라이드에서는 그 실패작을 먼저 설명하겠다.

패턴 인식을 위해서 단순히 패턴 정보의 제 1 사분면에 해당하는 부분(32*32)을 1024차원의 벡터로 나타내어 vector distance 를 구해본 결과, 74~86% 라는 그리 만족스럽지 못한 정확도를 갖게 되었다.

그 다음으로 구상한 방법은 조금 복잡했는데, 20개의 이미지에서 도출한 패턴 정보에서 DC를 제외한 가장 큰 성분의 위치 50개씩의 인덱스를 추려냈다. 이렇게 선정된 50~1000개의 인덱스(중복이 있다면 제거하기 때문에 선정된 인덱스의 범위가 생긴다.)에 해당하는 행렬의 항을 가지고 vector distance 를 구해 보았다. 다만, 이 방식은 중복된 인덱스가 많았고, 해당하는 항의 값이 대략적으로 비슷했기에, 놀랍게도 0~7%라는 아주 안 좋은 결과를 얻을 수 있었다.

모델링 – 패턴 인식 방법

앞의 두 실패를 통해서, 결국은 vector distance 를 사용하는 방법을 택하게 되었는데, 실행에 따라 너무 정확도의 편차가 컸기 때문에 이에 대한 개선이 필요했다. 따라서 그 이유에 대해 분석해보았는데, 매 실행에 따라 패턴 정보로 나타나는 이미지의 형태는 거의 변경이 없었으나, 그 밝기의 변화는 꽤 눈에 띌 정도였다. 랜덤하게 블록을 뽑아서 DFT를 수행하기에 나타나는 결과인 듯 했는데, 이를 해결하기 위해서 새로운 아이디어를 도입했다.

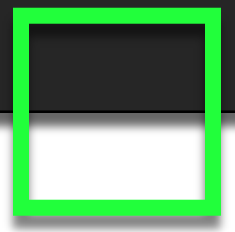
Cosine Similarity라는 벡터 간 유사도 검사 방법이 있었는데, 이는 단순히 벡터가 비슷한 방향을 가지고 있는지만 체크하는 방식이다. 즉, 비교에 사용되는 벡터의 길이를 무시하고, 두 벡터 간의 각도만 가지고 판단하는 것이다.

이 방식을 사용한 이유는, 무작위하게 선정된 블록들로 인해 DFT의 결과물인 64*64 행렬이 대체로 비슷한 형태를 보였지만, 그 절댓값에 어느 정도의 차이가 있었기 때문이다. 따라서 “형태” 만을 가지고 인식을 수행하기 위해 vector distance를 구하기 위한 벡터들을 모두 단위 벡터화해서, 거리가 온전히 두 벡터 간의 각도로 인해 도출되도록 했다.

정확도는 후술하겠다.

#2

결과와 비교



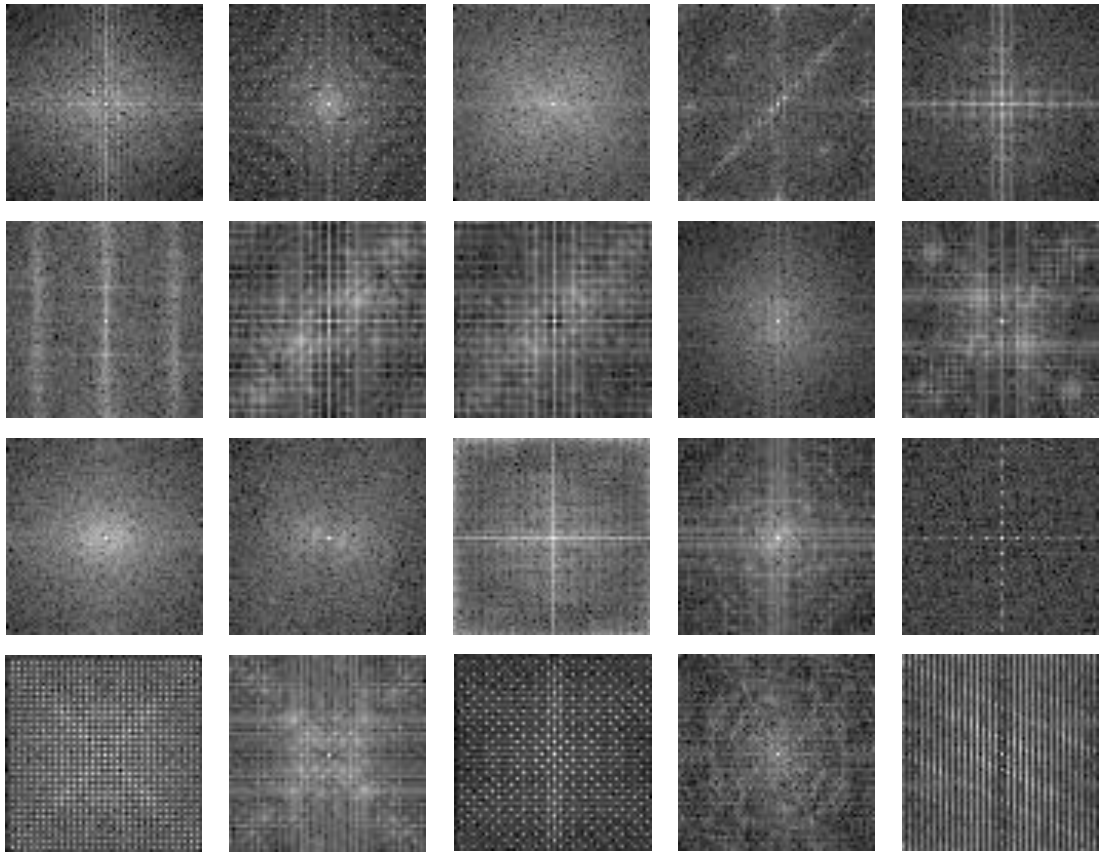
사용한 패턴 이미지

내가 사용한 패턴 이미지이다. 실제로 사용한 이미지는 이 이미지들이 각각 4번씩 반복된 형태인데, 패턴의 형태를 좀 더 잘 보이기 위해서 ¼ 부분만 나타냈다.

상당히 규칙적인 체크 무늬, 물방울 무늬, 사선, 물결 등 부터, 호피, 가죽(11번이 가죽 패턴이다.)과 같이 상당히 불규칙적인 무늬까지 사용했다.

테스트를 끝내고 알게 되었는데, 6번과 7번은 색상이 다른 타탄 체크 패턴이었다.

DFT result : Coefficient Image



DFT를 통해서 구한 패턴 정보이다. 앞에서 설명한 대로 시각화한 결과이다.

순서는 바로 앞 슬라이드에서 보여준 순서와 동일한데, 여기서 6, 7번 이미지를 보면 역시 거의 같은 형태를 보여주는 것을 알 수 있다. 여타 이미지들도 비슷한 형태를 보여주는 경우가 있는데, 이들이 실제로 어떤 인식 결과를 보여 주었는지는 바로 다음 슬라이드에서 보여주겠다.

여담으로, 가장 특이한 패턴을 보이는 19번 이미지는 물결 무늬의 이미지였고, 역시 특이한 패턴을 보이는 18번 이미지는 antique pattern에 속할 법한 육각 형태의 무늬였다.

테스트 실행 결과

```
PROBLEMS 2   OUTPUT   DEBUG CONSOLE   TERMINAL

PS C:\Users\pch68\HYU_MAT3008\HW10> python .\Fabric_Recognition.py
DFT Analysis on Images...
Make Criterion for Fabric Recognition...
Manual Test...
Input Filename or "quit" to Stop Testing > patterns/0.png
Recognized as 0.png
Input Filename or "quit" to Stop Testing > patterns/6.png
Recognized as 6.png
Input Filename or "quit" to Stop Testing > patterns/7.png
Recognized as 6.png
Input Filename or "quit" to Stop Testing > patterns/19.png
Recognized as 19.png
Input Filename or "quit" to Stop Testing > patterns/11.png
Recognized as 11.png
Input Filename or "quit" to Stop Testing > patterns/18.png
Recognized as 18.png
Input Filename or "quit" to Stop Testing > patterns/12.png
Recognized as 12.png
Input Filename or "quit" to Stop Testing > patterns/8.png
Recognized as 8.png
Input Filename or "quit" to Stop Testing > quit
Evaluate Accuracy...
Accuracy : 95%
```

필자가 구현한 프로그램에서 출력한 결과이다. 우선 Manual Test는 직접 filename을 입력하는 수동 테스트이며, “quit”로 수동 테스트를 종료하면, 각 이미지에 대해 무작위의 블록 5개를 뽑아서 총 100번의 이미지 인식 테스트를 수행하는 Accuracy Evaluation이 수행된다.

수동 테스트

```
PROBLEMS 2   OUTPUT   DEBUG CONSOLE   TERMINAL

PS C:\Users\pch68\HYU_MAT3008\HW10> python .\Fabric_Recognition.py
DFT Analysis on Images...
Make Criterion for Fabric Recognition...
Manual Test...
Input Filename or "quit" to Stop Testing > patterns/0.png
Recognized as 0.png
Input Filename or "quit" to Stop Testing > patterns/6.png
Recognized as 6.png
Input Filename or "quit" to Stop Testing > patterns/7.png
Recognized as 6.png
Input Filename or "quit" to Stop Testing > patterns/19.png
Recognized as 19.png
Input Filename or "quit" to Stop Testing > patterns/11.png
Recognized as 11.png
Input Filename or "quit" to Stop Testing > patterns/18.png
Recognized as 18.png
Input Filename or "quit" to Stop Testing > patterns/12.png
Recognized as 12.png
Input Filename or "quit" to Stop Testing > patterns/8.png
Recognized as 8.png
Input Filename or "quit" to Stop Testing > quit
```

앞에서도 설명했듯이, 수동으로 filename 을 입력하면 그 이미지에서 무작위로 64*64 블록을 하나 뽑아서, 그 블록의 DFT 결과를 가지고 어떤 이미지의 블록인지 테스트하는 과정이다.

대체로 괜찮은 결과를 보여주는 것을 알 수 있지만, 7.png의 인식에 실패하는 것도 확인할 수 있다.

이 이유는 결국 6.png와 7.png가 같은 패턴에 다른 색상을 가지는 이미지였기 때문이다. 내 구현이 coefficient vector 의 크기를 무시하고 비교를 진행하기 때문에 색상의 차이가 인식에서 큰 의미를 갖지 못했기 때문이다.

정확도 테스트

```
Evaluate Accuracy...
Accuracy : 95%
```

각 이미지 20장에 대해서 무작위 블록을 5개씩 뽑고, 각 블록들을 가지고 패턴 인식을 시도해서, 그 결과를 세어 본 결과이다. 20*5 = 100이므로 성공한 횟수를 세기만 하면 백분율로 정확도를 나타낼 수 있었다.

95%라고 하는 아주 높은 정확도를 보이고 있는데, 실제로 여러 번 시도한 결과 88~96% 정도의 정확도를 보이는 것을 알 수 있었다. 4~12%의 부정확한 결과의 이유를 분석한 결과는 다음과 같았다.

1. 6.png와 7.png의 동일한 패턴
   색상만 다르고 아예 같은 패턴을 갖고 있었기에, 이 두 이미지를 구분하는데 성공하기는 아주 어려웠다.

2. 이미지의 흑백화에 따른 왜곡
   사용한 이미지는 컬러지만, 이를 흑백으로 변환한 다음에 인식에 사용했기 때문에, 유사한 이미지가 생겼을 수 있다. 이로 인한 인식의 부정확성이 있었을 것이다.