영상처리

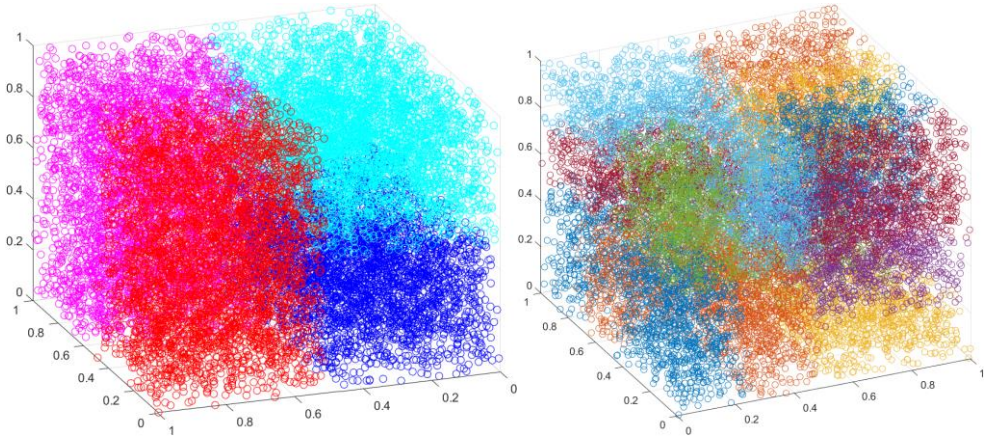
PROJECT 3

과목	영상처리
이름	차원범
학번	2016124217
교수님	오병태 교수님
제출일	20.06.14

1. KD Tree

1) Implement KD Tree

KD 트리는 노드에 입력된 데이터의 분산을 최대화하는 방향으로 자식 노드를 나눈다. 3차원 KD 트리에서는 분산을 최대화하는 차원과 그 값을 기준으로 분할을 실시한다. 0과 1사이의 16000여개의 점으로 트리를 구성한 결과는 아래와 같다.



➢ 깊이가 2인 노드들이 분리한 데이터(좌), 깊이가 4인 노드들이 분리한 데이터(우)

이렇게 분산을 최대화하는 방향으로 계속해서 데이터를 쪼갠다면, 모든 데이터에 대해 노드를 구성할 수 있다. 16000여개의 점으로 나온 깊이는 13이었다. 이를 검증하기 위하여 계산한 결과, $\Sigma_{n=0}^{13} 2^n = 16383$ 이므로 올바르게 구성되었다는 것을 알 수 있다.

2) Find closest point using KD Tree

이렇게 구성된 트리에 새로운 데이터를 넣는다면 leaf node까지 타고 내려가 자신과 가장 가까운 node를 찾을 수 있다. 하지만 이렇게 찾은 node는 유클리디안 거리로 환산하였을 때, 항상 최근접 점을 찾을 수 있는 것은 아니다. 트리는 각 차원의 분산을 최대화하는 방향으로 구성되었기 때문에, 가까운 점은 찾을 수 있지만 반대편 경계선을 넘어 더 가까운 점이 존재할 수 있다.

1000개의 임의의 점(k)를 생성하고, 유클리디안 거리로 계산한 최근접 점과, 트리를 이용하여 찾은 최근접 점을 비교한 결과는 아래의 표와 같다.

정확도	평균 오차	오답 오차
23.9%	0.0190	0.0435

평균 오차는 트리로 분류한 값과 유클리디안 거리로 찾은 최근접 점 간의 거리 차이를 계산한 값이다. 오답 오차는 여기서 거리가 0인(최근접점을 올바르게 찾은) 점을 제외하고, 오답인 경우만의 거리 차이를 계산한 값이다. 데이터셋이 0과 1사이에 분포하므로 0.05정도의 거리 차이는 아주 가까운 점을 찾았다는 것을 의미한다. 하지만 근접한 점만 찾았을 뿐, 76%가량의 데이터는 정답을 맞히지 못하였다.

3) Find closest point using Backtracking Algorithm

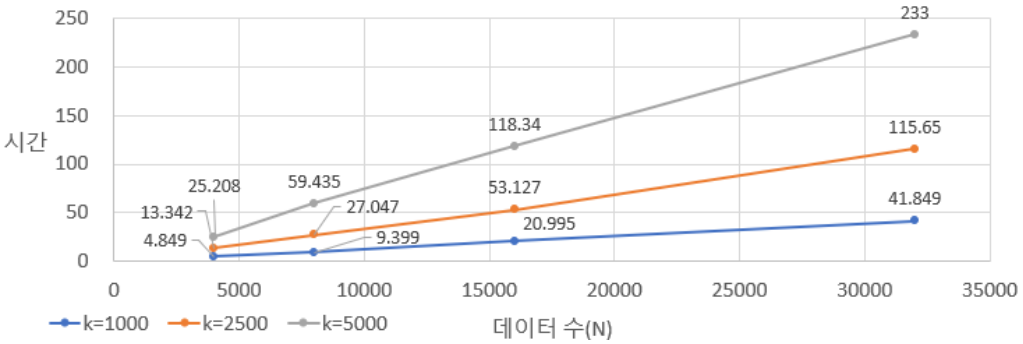
따라서 Backtracking을 통해 자신이 거쳐 온 노드를 다시 확인한다. 트리를 거슬러 올라가며 반대쪽 노드를 모두 탐색한다. 하지만 이렇게 하면 과도한 시간복잡도를 가진다. 따라서 현재 찾은 최근접 점과의 거리와 탐색할 노드(분할 평면)와의 거리를 비교하여 평면까지의 거리가 더 작을 때에만 반대편 노드 탐색을 시작한다. 이렇게 찾은 결과는 아래의 표와 같다.

정확도	평균 오차	오답 오차
100%	0.00	-

Backtracking을 통해 모든 최근접 점을 찾을 수 있었다. 따라서 오차는 0이며, 정확도는 100%를 달성하였다.

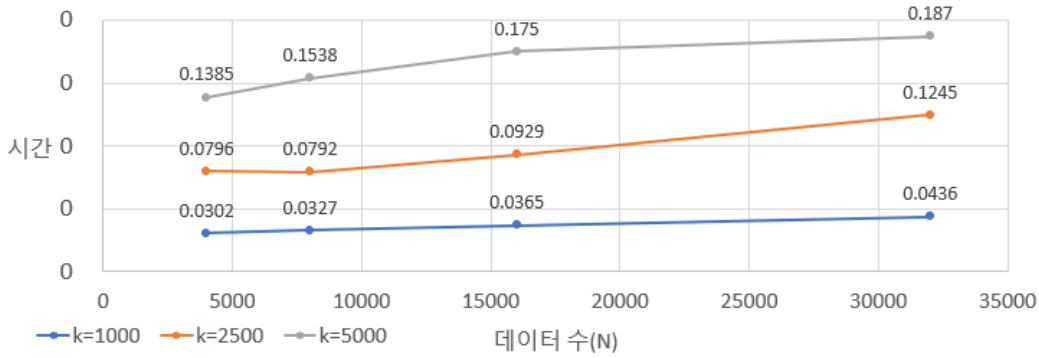
4) Measure & compare

3가지의 경우에 대해 시간을 측정하였다. 첫 번째는 임의의 점(k)와 주어진 데이터셋과의 차이의 norm을 이용하여 유클리디안 거리를 구할 수 있다. 모든 데이터에 대해 하나씩 비교를 실시하며, 이의 최소가 되는 점이 자명한 최근접 점이다. 두 번째는 KD Tree의 순방향 분류만을 이용하여 찾는 알고리즘이며, 세 번째는 Backtracking까지 사용하여 최근접 점을 찾는 알고리즘이다.



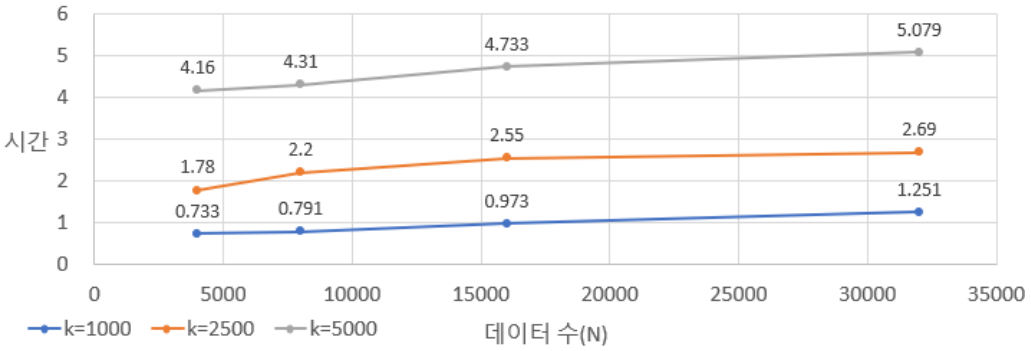
➢ 유클리디안 거리를 이용하여 구했을 때 걸린 시간

유클리디안 거리를 이용하여 최근접 점을 찾을 때 최악의 조건(N=32000, k=5000) 에서 최대 233초가 걸렸다. 또한 N에 대한 시간복잡도는 O(N)으로 선형적으로 증가하였다. k에 대한 시간복잡도 또한 O(N)으로 추측된다. 해당 알고리즘은 N값과 k값의 증가에 취약하며 특히 k값의 증가에 민감하다.



➢ Tree의 순전파를 이용하여 구했을 때 걸린 시간

Tree의 순전파만을 이용하여 측정하였다. 이 때 최악의 조건(N=32000, k=5000)에서 0.187초만 걸렸으며, 정확도는 떨어지지만 근사치를 찾을 때 매우 효율적인 방법이라고 불 수 있다. N에 대한 시간복잡도는 O(logN)으로 보이며, k에 대한 시간복잡도 또한 O(logN)으로 추측된다.



➢ Backtracking을 이용하여 구했을 때 걸린 시간

Backtracking을 사용하였을 때에는 최악의 조건(N=32000, k=5000)에서 약 5초를 달성하였다. 유클리디안 거리를 이용하여 계산하였을 때보다 월등히 빠르지만 정확도는 같다는 점에서 더 우월하다고 할 수 있다. N에 대한 시간복잡도는 O(logn)으로 보이며, k에 대한 시간복잡도는 O(N)으로 보인다.

2. Matching

1) Least Mean Square Error

➢ SURF Feature를 이용하여 두 이미지의 공통점을 찾아 매칭한 모습

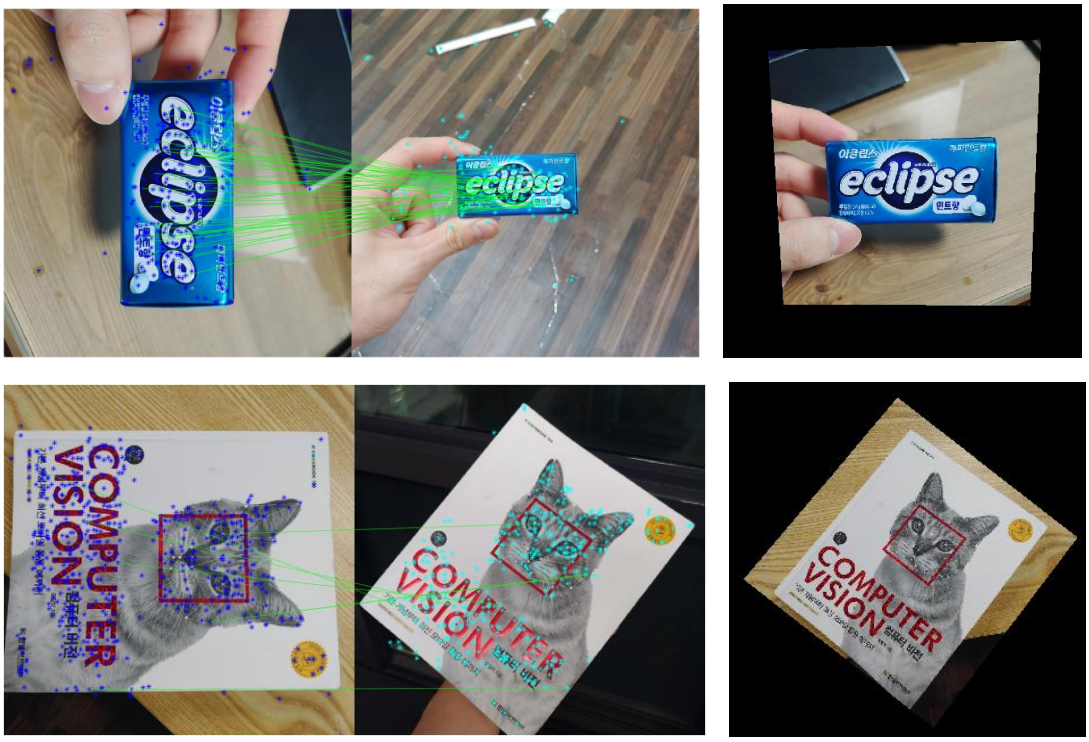
SURF Feature를 이용하여 두 이미지 간의 특징점을 추출할 수 있다. 위 사진은 그 특징점들 중, 특징 벡터들 가의 거리가 가까운(유사한) 상위 특징점만 추출하여 매칭한 결과이다. 이 점들을 이용하여 최소제곱법을 실시할 수 있다.

$$\begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x_1u_1 & -y_1u_1 \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -x_1v_1 & -y_1v_1 \\ & & & & & \cdot & & \\ & & & & & \cdot & & \\ & & & & & \cdot & & \\ & & & & & \cdot & & \\ & & & & & \cdot & & \\ x_n & y_n & 1 & 0 & 0 & 0 & -x_nu_n & -y_nu_n \\ 0 & 0 & 0 & x_n & y_n & 1 & -x_nv_n & -y_nv_n \end{bmatrix} \begin{bmatrix} t_{11} \\ t_{12} \\ t_{13} \\ t_{21} \\ t_{22} \\ t_{23} \\ t_{31} \\ t_{32} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

최소제곱법을 이용해 두 특징 벡터들 간의 변환 행렬 T(DoF=8)를 위의 식을 이용해 도출해 낼 수 있다. 이를 이용하면 특징 벡터들간의 변환을 실시할 수 있고, 두 특징 벡터가 제대로 매칭되는지 확인하여 원하는 T를 얻었는지 확인할 수 있다.

➢ 특징 벡터(좌측 이미지 빨간색 점)에 변환 행렬 T를 곱하여 우측 이미지에 매칭

위와 같이 스케일과 각도를 변형시켰음에도 변환 행렬 T를 통해 정상적으로 두번째 이미지에 사영되는 것을 확인할 수 있었다. 따라서 좌측의 이미지 자체를 넣어도 우측과 같이 표현될 것이다.



➢ 입력 이미지(좌측, 중간)에 의해 만들어진 변환행렬 T로 다시 입력 이미지(좌)에 변환 행렬 T를 곱하여 변환한 이미지(우)

두 이미지의 특징 벡터만을 이용하여 만든 변환 행렬 T 를 다시 입력 이미지에 곱한 결과 목표 이미지의 물체와 같은 구도를 만들 수 있었다.

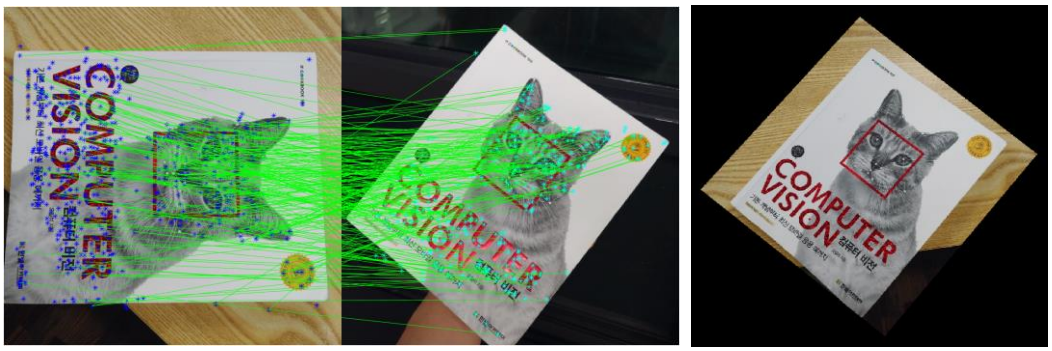
2) RANSAC

변환 행렬을 통하여 이미지를 목표 이미지에 맞게 변환시켰다. 하지만 해당 알고리즘은 outlier에 대해 굉장히 민감하다. 따라서 너무 많은 특징 벡터를 사용하면 제대로 매칭이 되지 않을 수 있다.

➢ 임계값을 낮추어 많은 특징 벡터에도 매칭이 되도록 실시한 이미지

이와 같이 임계값을 낮추면, outlier에 의해 변환 행렬 T가 제대로 된 역할을 수행하지 못한다. 이를 해결하기 위해서는 RANSAC알고리즘을 이용한다.

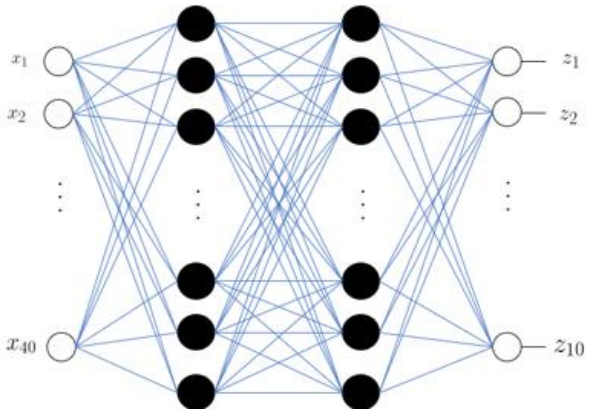
RANSAC 알고리즘은 일종의 voting 알고리즘이다. 주어지는 모든 feature vector을 이용하지 않으며 샘플 데이터 3개 혹은 그 이상의 샘플 몇 개를 가지고 변환 행렬 T를 도출한다. 이렇게 도출한 T를 가지고 입력 사진의 feature vector의 변환을 실시한다. 실시한 변환 값을 목표 사진의 feature vector와 각 거리를 비교한다. 이 때, 미리 설정한 margin 이내에 있는 점들은 inlier가 되며, 이 inlier가 많을수록 높은 점수를 받는다. 해당 과정을 반복하며 최고 점수를 받은 inlier를 가지고 다시 변환 행렬 T를 도출해 내는 것이 해당 알고리즘의 목표이다.



➢ RANSAC 알고리즘을 이용하여 더 많은 특징 벡터에서의 변환을 실시한 이미지

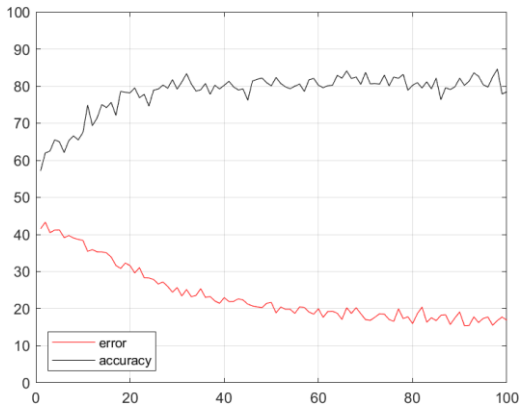
단순히 최소제곱법을 이용했을 때보다 보다 뛰어난 성능을 볼 수 있었다. False positive의 수가 더 많아 졌음에도 정상적인 역할을 수행하였다. 이 때, RANSAC을 돌리는 횟수는 100이상이 안정적이었으며, 100 이하일 때에는 불안정한 출력을 보여주었다.

3. MLP



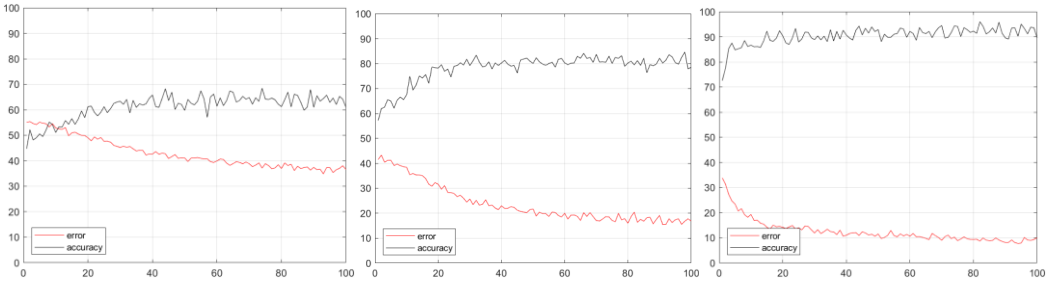
MLP(Multi-Layer Perceptron)은 2층 이상의 Full connected hidden Layer를 가지며 오류 역전파 알고리즘을 통해 분류, 회귀 모델의 학습이 가능한 기법이다. Mnist dataset을 이용해 MLP의 학습을 진행하였다. Mnist dataset의 숫자 1~10은 one hot encoding을 통해 10가지 출력으로 대신할 수 있다.

Learning rate	Hidden Layer's parameter	Activation Function
0.01	10*10	Sigmoid



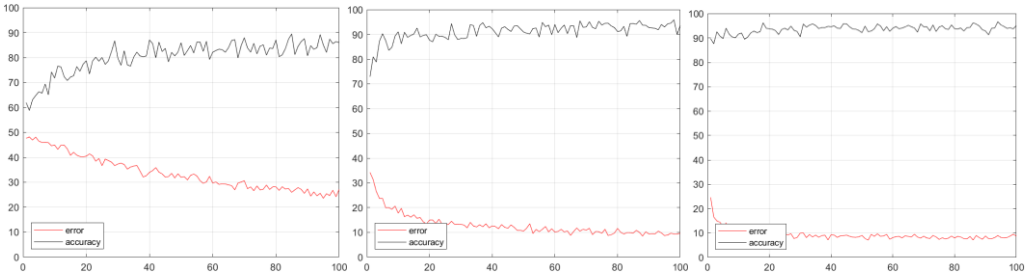
➢ 최초 구성한 MLP로 학습을 진행한 결과

최초에는 60%정도였던 정확도가 세대(가로축)가 지남에 따라 최대 80%의 정확도를 가지는 것을 볼 수 있다. 하지만 30번째 세대부터는 더 이상 정확도가 오르지 않았다. 이는 learning rate 혹은 hidden layer의 parameter의 문제로 추측해 볼 수 있다.



➢ 좌측부터 5*5, 10*10, 50*50의 Hidden Layer를 가지고 학습을 진행한 결과

Hidden Layer의 parameter를 더욱 늘린 결과 정확도가 90%에 가까우며, 20세대 이전에 최대치에 가까운 결과를 보여주었다. 하지만 parameter를 줄인 결과 최대 70%의 정확도를 보여주었다. 이는 parameter의 부족으로 적절한 학습이 진행되지 않은 결과이다.



➢ 좌측부터 $0.001, 0.01, (0.1 * 0.96^{epoch})$의 학습률을 가지고 진행한 결과*(Hidden Layer : 50*50)*

학습률을 변화시키며 진행한 결과이다. 0.001의 학습률은 꾸준한 정확도 향상을 보여주었으나 90%의 정확도를 달성하기까지 너무 많은 세대를 소비하였다. 또한 최초로 진행했던 0.01의 학습률도 우수하였으나, 학습 초기의 정확도를 더욱 개선시키기 위하여 $0.1 * 0.96^{epoch}$로 학습률을 변경하여 테스트하였다. 그 결과 학습 초기의 정확도가 크게 개선되었으며 세대가 지날수록 학습률이 작아진다. 따라서 많은 세대가 지난 후에도 작지만 약간의 정확도 향상도 이룰 수 있었다.

이렇듯 MLP는 최초 parameter가 적절하다면 그 결과물이 매우 우수하다. 하지만 parameter를 선정하는 과정이 산술적으로 도출되지 않으며, 직접 사용자가 조절하며 최적의 값을 찾아야 한다는 단점이 있다.