CS485CW2

20220352 손유호,20210256 박주현

Q1. K-means codebook

Vocabulary Size 는 K-means Clustering 의 클러스터 수 K 와 동일하다. 클러스터의 centroids 는 visual vector 에 해당하며 계산 시간은 k 값의 증가에 따라 선형적으로 증가한다. Vocabulary Size 가 클수록 이미지의 다양한 특징을 포착하는 것을 볼 수 있었다. 하지만 Vocabulary Size 가 너무 커지니 overfitting 을 일으켜 오히려 퍼포먼스가 떨어지는 모습을 보였다. 반대로, Vocabulary Size 를 작게 하니 underfitting 이 발생하여 성능이 떨어졌다.

bag-of-words histograms 은 이미지를 visual Vocabulary 의 각 word 에 대한 발생 빈도로 표현한다. 그렇기에 이 히스토그램을 통해, 클래스의 이미지들끼리 어떤 visual 특징을 가지는지 시각적으로 볼 수 있다. 또한, 이미지의 핵심적인 정보를 요약해서 볼 수 있기에 유용하다고 볼 수 있다. 특히, 특정 클래스의 이미지에서만 자주 나타나는 word 의 경우 해당 클래스의 중요한 visual 특징을 포함하고 있을 것임을 추론할 수 있다.
실제로, 이미지의 bag-of-words histograms 에서 같은 class 인 경우 histogram 의 분포가 유사함을 볼 수 있었다.
fig 1 의 경우, image 1-image 5 가 히스토그램을 통해 표현되어있다. 주요 피크점들을 비교해 보기 위해 임의의 두 히스토그램을 뽑아서 비교해봤을 때 모두정확도가 30%를 넘지 않았다. 따라서 image1-5 는 전부 다른 그림임을 추측할 수 있다.

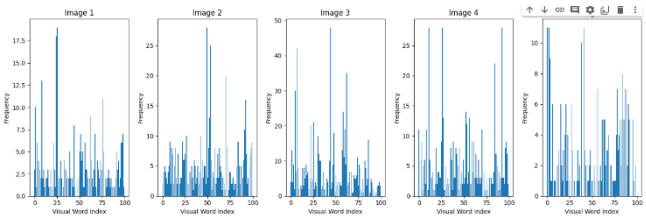

fig 1.단어가방 히스토그램 시각화

vector quantisation process 는 연속적으로 샘플링된 진폭값들을 그룹핑하여, 이 그룹단위를 몇개의 대표값으로 양자화하는 것이다. 의미있는 데이터와 아닌 데이터를 구분하는 작업으로 데이터 압축에 활용되어 왔다. 즉, visual vocabulary 에서 가장 가까운 visual vector 의 인덱스에 descriptor 를 매핑하는 것으로 재정의 내릴 수 있다. 따라서 vector quantisation process 를 통해 얻은 결과 그림은 fig 2 와 같다. quantisation process 는 각 descriptor 들을 모든 centroids 와 비교하기에 많은 계산 시간이 소요되며 실제로도 vocabulary size 에 선형적으로 비례하는 모습을 보였다.

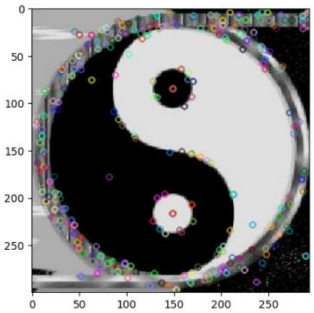

fig 2. image [0]에 descriptor mapping, key point plot

Q2. RF classifier

RF parameters 에는 trees 의 수, trees 의 depth, degree of randomness parameter, type of weak-learners 같은 것이 있다. 각 parameters 의 값을 바꿔보며 실험을 진행하였다.
먼저, tree 의 수는 많을 수록 모델의 안정성과 정확도가 향상되는 모습을 보였다. 하지만 일정 수준 이상으로 tree 수가 많아질 경우 안정성은 꾸준히 늘었지만 정확도는 그대로인 모습을 보였다. 오히려 training 시간과 모델의 크기는 꾸준하게 tree 의 수에 비례하여 커졌기에 이러한 사항들을 고려하여 tree 의 개수를 적절한 수준으로 사용하는 것이 적합할 것 같다.
그리고 Vocabulary Size 가 작을 경우, tree 의 수에 따른 정확도 향상의 차이가 작았고 낮은 수준의 정확도에서 머물렀는데 이는 Vocabulary Size 가 작을 경우 이미지의 특징을 제대로 포착하지 못하였기 때문으로 추측된다.
trees 의 depth 는 깊을 수록 복잡한 데이터 패턴을 학습할 수 있기에 높은 정확도를 기대할 수 있지만 오히려 overfitting 으로 인해 정확도가 떨어질 수도 있다. 또, 깊을 수록 training 시간과 예측 시간이 길어질 것임을 예측할 수 있다. 깊이를 서서히 늘리며 실험해보면, 처음에는 정확도가 올라가는 경향을 보였지만 어느 순간부터는 정확도가 내려가기도 하였다. 이는 앞서 말한 overfitting 으로 인해 정확도가 떨어진 것으로 보인다.

randomness parameter 를 통해 모델의 다양성 증가와 정확도 향상을 시도할 수 있다. randomness parameter 가 클 경우 tree 가 비슷해지고, 작을 경우 tree 간의 다양성이 증가한다. 따라서 다양성이 증가할 경우 앙상블 모델의 정확도가 향상될 것이라 추측할 수 있으며, 실제 실험 결과도 이와 같은 양상을 보였다. randomness parameter 가 커질 수록 정확도가 낮아졌으며, 작아질수록 정확도가 높아졌다. training 시간은 randomness parameter 에 비례하는 모습을 보였다.
weak-learners 를 axis-aligned 이나 two-pixel test 로 바꿔가며 학습을 한 결과, 정확도의 관점에서 두 weak-learner 는 큰 차이가 없었다. 따라서 이 실험의 경우 weak-learner 보다 다른 parameter 들이 성능에 직접적인 영향을 미칠 것이라 예측된다.

```
Training took 0.26 seconds.
Recognition Accuracy: 0.6869565217391305
Confusion Matrix:
 [[ 5  1  0  0  0  1  1  0  0  0]
 [ 0 15  1  2  0  3  0  0  0  0]
 [ 0  0  8  0  0  2  0  0  3  3]
 [ 0  2  3 14  0  0  0  0  0  0]
 [ 0  1  0  0  4  1  0  0  0  0]
 [ 0  0  0  2  0  7  0  1  0  0]
 [ 0  2  0  1  1  1  2  0  0  0]
 [ 0  0  0  0  0  0  0  5  0  0]
 [ 0  0  1  0  0  0  0  0  5  0]
 [ 0  1  2  0  0  0  0  0  0 14]]
Testing took 0.01 seconds.
Image 1 - Actual: wrench, Predicted: wrench
Image 2 - Actual: trilobite, Predicted: trilobite
Image 3 - Actual: umbrella, Predicted: umbrella
Image 4 - Actual: yin_yang, Predicted: yin_yang
Image 5 - Actual: yin_yang, Predicted: yin_yang
Image 6 - Actual: water_lilly, Predicted: water_lilly
Image 7 - Actual: umbrella, Predicted: yin_yang
Image 8 - Actual: tick, Predicted: tick
Image 9 - Actual: trilobite, Predicted: trilobite
Image 10 - Actual: water_lilly, Predicted: wheelchair
```

fig3. max depth=none, n_estimator=100, rand_state=0 일때 accuracy, confusion matrix

Q3.
1. K-means code book
K첫번째로, K-means를 이용하여 시각적 어휘를 구축해야 한다. 클러스터 수 (K)와 100k개의 다중 스케일 SIFT 기술자를 사용하여 초기 center를 선택했고, 각 기술자를 가장 가까운 중심에 할당하여 중심을 업데이트한다. 시각적 어휘는 이러한 center로 구성되는데, K-means의 vector quantisation complexity는O(DN'K)로, 여기서 D는 기술자의 차원, N'은 패치의 수, K는 클러스터 수를 나타낸다.
하지만 초기 중심점 설정의 부정확성은 클러스터링 결과에 부정적인 영향을 줄 수 있다. 최소 제곱 왜곡을 최소화하기 위해 여러 번 클러스터링을 반복하고 적절한 결과를 선택하는 방법을 사용할 수 있다.
Bag-of-words 히스토그램 을통해 이미지를 대표하고, 이를 통해 이미지 간 유사성을 확인할 수 있는게 특징이다. 예를 들어, Tr1과 Te1의 히스토그램에서 둘 다 60번째 bin에 중요한 요소가 있음을 확인할 수 있다. 그러나 Tr1과 Tr2를 비교하면, 패턴이 다르다는 것을 알 수 있다.

2. Randomized Forest(RF) classifier
위에서 설명하는 이미지를 분류하기 위해 이러한 bag-of-words에 RF(Randomized Forest)를 적용해야 한다.
트리 수가 100보다 클 때, vocabulary size가 20인 경우 정확도가 59%였고, vocabulary size가 256인 경우 70%으로 변했다. Vocabulary size가 256일 때, 트리 수가 10인 경우 정확도가 55%이다. 트리 수를 100으로 증가시키면 정확도가 70%로 증가한다. 따라서 트리수가 증가할수록 정확도가 증가한다 볼 수 있다. 또한, vocabulary size도 정확도에 어느정도 기여를 하고 있다 판단할 수 있다. 잠깐 vocabulary size에 대해 이야기하자면, RF의 성능에는 vocabulary size가 중요한 기여를 한다. 위에 내용을 봐도 알 수 있는 부분이다. 만약 vocabulary size가 작다면, 는 동일한 클래스에 다른 특징을 분류하기 쉽다. 만약 vocabulary size가 256보다 크면 정확도가 약 70% 유지되며, 이미지 설명자가 제대로 분류되고 이미지의 특징이 구별하기 쉽다. 다시 본론으로 돌아와서, 트리 수가 증가함에 따라 정확도의, 트리 수가 많을수록 분류 정확도가 더 안정적이라 결론지을 수 있다.
이제 트리의 depth에 따라 생각해보자. 트리 수가 100이고, vocabulary size가 256인 조건에서 트리의 depth를 1-15까지 조정해본 결과, 깊이가 9보다 작은 경우 정확도가 65%에서 70% 사이를 움직인다. 하지만 그 반대의 경우 9-15의 accuracy 대부분이60%에서 65%사이의 accuracy 를 가진다. 따라서 트리의 깊이를 늘리면 트리의 다양성이 증가하지만, 너무 깊으면 RF의 과적합 문제를 야기할 수 있다는 점을 알 수있다. 랜덤성 factor를 조정할경우 , accuracy는 약 5%정도의 감소를 겪었으나, 훈련 프로세스의 계산 시간은 선형적으로 개선된다.
추가적으로, 다른 약한 학습기를 비교한 결과 대부분의 경우 축 정렬과 두 픽셀 사이에는 뚜렷한 차이가 없다.

다만, 트리 수가 2개인 경우 축 정렬 분할 함수의 accuracy가 두 픽셀을 기반으로 한 함수보다 약 10% 높다.

성공 및 실패 사례

Test	 Class 7	 Class 7	 Class 7
Output class	 Class 7	 Class 6	 Class 5
Result	Success	Failure	Failure

Fig4. Success, Fail

Class 7의 일부 이미지가 Class 6 및 Class 5로 잘못 분류되는 경우가 있었다. 또한,RF coding를 적용하면 Class 7의 정확도가 10% 향상됨을 알 수 있었다.

3. RF-codebook

K-means coding과 다르게, RF coding은 분류기 대신 트리를 공간 분할 방법으로 사용한다. vector quantisation process에서 test descriptor에 대해 RF의 각 트리는 고유한 index를 반환한다. 이들의 투표 ensemble 이 전체적인 히스토그램을 형성하며, 이는classify에 이용된다.

RF codebook의 구축 단계에서 깊이를 20으로 설정되고, 트리 수를 10에서 20까지 변화시켰을 때, 정확도는 75%에서 적은 변동폭으로 움직이다 감소했다. 따라서 특정 깊이에서 트리 수가 많아지면 accuracy가 높아진다 판단할 수 있다.

깊이를 높이면 정확도가 향상되지만, 어느정도 크기가 커지면75% 에서 머무르는 것을 알 수 있었다. RF의 각 트리는 깊이가 커지면 overfitting의 가능성이 커지지만 트리들의 앙상블은 과적합의 영향을 굉장히 많이 줄일 수 있다.

이제, vector quantitation complexity를 비교해보자.

k-means의 경우 O(KN'D)로 표현됐었다. RF의 경우는 O(√N' log K)로 나타나 진다. 즉, RF codebook이 K-means code book보다 복잡도가 낮다.

정확도의 경우,

RF-codebook이 vocabulary size가충분히 큰 경우 K-means 코드북보다 약 5%정도 컸음을 알 수 있었다.

Q4.

이 부분은 코드로 설명하는 것이 어느정도 유리할 것 같아 코드로 설명하려 한다.

```python
def build_cnn(input_shape, num_classes):
    model = Sequential()
    model.add(Conv2D(32, (3, 3), activation='relu', input_shape=input_shape))
    model.add(MaxPooling2D((2, 2)))
    model.add(Flatten())
    model.add(Dense(128, activation='relu'))
    model.add(BatchNormalization())
    model.add(Dropout(0.2))
    model.add(Dense(num_classes, activation='softmax'))
    return model

# CNN 모델 생성
model = build_cnn(input_shape=image_size + (3,), num_classes=len(folders))
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
model.summary()
```

Fig 5. Code1

위와 같이 CNN모델을 생성해냈고, 결과는 다음과 같았다.

```
Model: "sequential_8"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_13 (Conv2D)          (None, 126, 126, 32)      896

 max_pooling2d_13 (MaxPooli  (None, 63, 63, 32)        0
 ng2D)

 flatten_8 (Flatten)         (None, 127008)            0

 dense_17 (Dense)            (None, 128)               16257152

 batch_normalization_13 (Ba  (None, 128)               512
 tchNormalization)

 dropout_6 (Dropout)         (None, 128)               0

 dense_18 (Dense)            (None, 10)                1290

=================================================================
Total params: 16259850 (62.03 MB)
Trainable params: 16259594 (62.03 MB)
Non-trainable params: 256 (1.00 KB)
```

Fig6. Model execution 1

```python
# 데이터셋 분할
X_train, X_test, y_train, y_test = train_test_split(images, labels, test_size=0.2, random_state=42)

# 모델 학습
history = model.fit(X_train, y_train, epochs=20, validation_split=0.2)

# 성능 평가
test_loss, test_acc = model.evaluate(X_test, y_test)
print("Test Accuracy:", test_acc)
```

Fig 7.Code 2

이에 따라 데이터셋을 train 과 test 로 0.2으 비율로 분할한다.

그 이후, epochs=20으로 모델을 학습시키고 성능을 평가했다. 자세한 코드들은 주석에 있다.

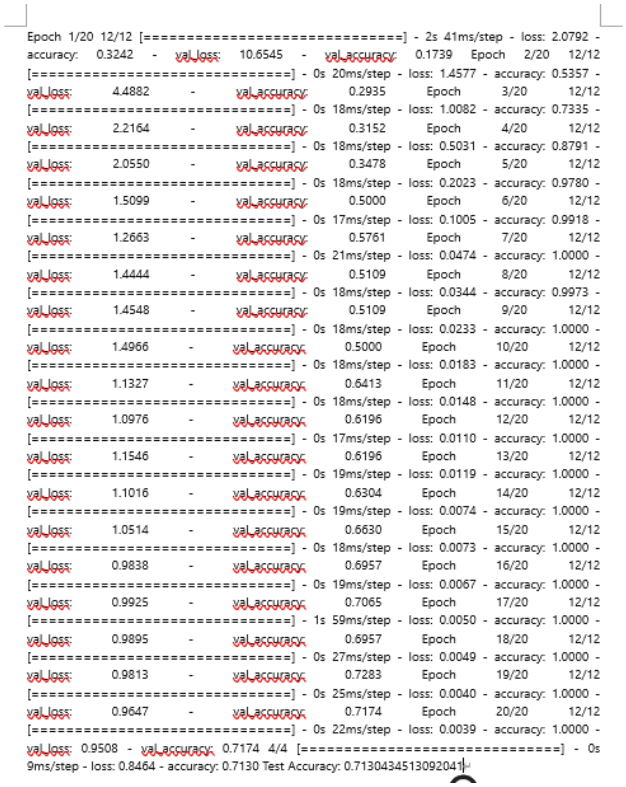

Fig 8. Execution

평균적으로 val loss=0.9508로 나타났고, val accuracy=0.7174, Test accuracy=0.713으로 나타났다.

우리는 다양한 변수들을 조정해보며,

다양한 네트워크 아키텍처 즉, 모델이 복잡할수록 성능 좋아지지만 너무 레이어가 많아지면 성능 감소한다는 사실을 알았고, 배치 정규화를 사용하면 모델학습이 안정적으로 되며 성능이 향상된다는 것을 알았다.

또한 드롭아웃 및 정규화를 통해 성능이 향상되고 오버피팅을 방지할수 있다.

손실함수를 따라 성능이 달라지며, 손실이 적어질수록 모델의 정확도가 올라갔다.

또한 신경망 압축을 통해 모델 압축을 하면 성능은 떨어지지만, 그만큼 용량이 줄어져 학습과 추론시간이 빨라진다는 사실을 알 수 있었으며,

배치크기가 크면 클수록 빠르게 모델을 학습할수 있지만, 그만큼 학습이 불안해짐을 알 수 있었다.

마지막으로, 사전 훈련된 CNN을 사용하면 더 빠르게 loss를 수렴시킬수 있었다.

Appendix

```
from google.colab import drive
drive.mount('/content/drive')
```

```
import os
import cv2
import numpy as np
from sklearn.model_selection import train_test_split
from keras.utils import to_categorical
from keras.preprocessing.image import ImageDataGenerator
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from keras.layers import BatchNormalization, Dropout
```

```
def load_and_preprocess_images(folders, image_size=(128, 128)):
    images = []
    labels = []
    label_map = {folder: idx for idx, folder in enumerate(folders)}

    for folder in folders:
        for filename in os.listdir(folder):
            img_path = os.path.join(folder, filename)
            img = cv2.imread(img_path)
            if img is not None:
                img = cv2.resize(img, image_size)
                img = img / 255.0  # Normalize pixel values
                images.append(img)
                labels.append(label_map[folder])

    images = np.array(images)
    labels = to_categorical(labels, num_classes=len(folders))

    return images, labels
```

```
base = "/content/drive/MyDrive/2023_2/Caltech_101/101_ObjectCategories/"
folders = [base + 'tick',base + 'trilobite',base + 'umbrella',base + 'watch',base + 'water_lilly', base + 'wheelchair', base + 'wild_cat',base + 'wi
```

```
image_size = (128, 128)
images, labels = load_and_preprocess_images(folders, image_size)
```

```
def build_cnn(input_shape, num_classes):
    model = Sequential()
    model.add(Conv2D(32, (3, 3), activation='relu', input_shape=input_shape))
    model.add(MaxPooling2D((2, 2)))
    model.add(Flatten())
    model.add(Dense(128, activation='relu'))
    model.add(BatchNormalization())
    model.add(Dropout(0.2))
    model.add(Dense(num_classes, activation='softmax'))
    return model

# CNN 모델 생성
model = build_cnn(input_shape=image_size + (3,), num_classes=len(folders))
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
model.summary()
```

```
# 데이터셋 분할
X_train, X_test, y_train, y_test = train_test_split(images, labels, test_size=0.2, random_state=42)

# 모델 학습
history = model.fit(X_train, y_train, epochs=20, validation_split=0.2)

# 성능 평가
test_loss, test_acc = model.evaluate(X_test, y_test)
print("Test Accuracy:", test_acc)
```