[Project #2]
Deep Learning for Image Classification with EM Algorithms

[Team Name (#팀 07)]

	이름	학번	학년	E-mail
팀장	임성현	20180325	3	jfknfj920@naver.com
조원 1	박진혁	20180293	3	jh040597@naver.com
조원 2	박병수	20170283	3	byeongsu@soongsil.ac.kr

1 INTRODUCTION

영상 분류를 위해 deep learning 모델을 구축한다. 어떻게 feature space 를 관측 및 분석하여 정확도를 개선할 수 있는지 탐색하고 더 나은 deep learning model 구축 방법을 모색한다. 다음과 같은 방법으로 위 내용을 실행하였다.

1. Feature space visualization
   - PCA
   - GMM
2. xAI visualization
   - CAM

2 MAIN STRUCTURE

```
cnn = tf.keras.models.Sequential()
cnn.add(tf.keras.layers.Conv2D(32, (3, 3), activation='relu', padding='same',
                               input_shape=(32, 32, 3)))
cnn.add(tf.keras.layers.BatchNormalization())

cnn.add(tf.keras.layers.Dropout(0.25))
cnn.add(tf.keras.layers.Conv2D(32, (3, 3), activation='relu', padding='same'))
cnn.add(tf.keras.layers.BatchNormalization())

cnn.add(tf.keras.layers.Dropout(0.25))
cnn.add(tf.keras.layers.Conv2D(32, (3, 3), activation='relu', padding='same'))
cnn.add(tf.keras.layers.BatchNormalization())

cnn.add(tf.keras.layers.MaxPooling2D((2, 2)))

cnn.add(tf.keras.layers.Dropout(0.25))
cnn.add(tf.keras.layers.Conv2D(64, (3, 3), activation='relu', padding='same'))
cnn.add(tf.keras.layers.BatchNormalization())

cnn.add(tf.keras.layers.Dropout(0.25))
cnn.add(tf.keras.layers.Conv2D(64, (3, 3), activation='relu', padding='same'))
cnn.add(tf.keras.layers.BatchNormalization())

cnn.add(tf.keras.layers.Dropout(0.25))
cnn.add(tf.keras.layers.Conv2D(64, (3, 3), activation='relu', padding='same'))
cnn.add(tf.keras.layers.BatchNormalization())

cnn.add(tf.keras.layers.MaxPooling2D((2, 2)))

cnn.add(tf.keras.layers.Dropout(0.25))
cnn.add(tf.keras.layers.Conv2D(128, (3, 3), activation='relu', padding='same'))
cnn.add(tf.keras.layers.BatchNormalization())

cnn.add(tf.keras.layers.Dropout(0.25))
cnn.add(tf.keras.layers.Conv2D(128, (3, 3), activation='relu', padding='same'))
cnn.add(tf.keras.layers.BatchNormalization())

cnn.add(tf.keras.layers.Dropout(0.25))
cnn.add(tf.keras.layers.Conv2D(128, (3, 3), activation='relu', padding='same'))
cnn.add(tf.keras.layers.BatchNormalization())

cnn.add(tf.keras.layers.GlobalAveragePooling2D())

cnn.add(tf.keras.layers.Dropout(0.25))
cnn.add(tf.keras.layers.Dense(64, activation='relu'))
cnn.add(tf.keras.layers.Dense(10, activation='softmax'))

cnn.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
```

그림 1 CNN 으로 구현한 Deep Learning model

- 입력 데이터로 (32, 32, 3) 인 이미지를 50,000 장 입력 받아 학습한다. 해당 CNN 모델 구성은 다음과 같다.
- 총 9 개의 convolution layer 로 구성된다.
  - 일반적으로 convolution filter(이하 kernel)의 수를 늘리는 것보다 깊게 convolution layer 를 구성하는 것이 성능이 좋다. 따라서, 하나의 layer 당 최대 128 개의 kernel 수를 갖도록 하였다.
- Convolution layer 를 통과하면 Batch normalization 하였다.
  - layer 가 깊어질수록 gradient vanishing problem 이 발생이 빈번하게 된다. 즉, covariate shift 현상이 발생하므로 batch normalization 을 하였다.
- Dropout 기법으로 규제하였으며 25%만큼 dropout 하였다.
- GAP 로 fully connected layer 를 대신하였다.
- loss function 으로 교차 엔트로피를 활용, activation function 으로 softmax 를 활용하였다.
  - True, False 로 분류하는 것이 아닌 10 가지의 범주로 분류하는 경우이므로 sigmoid 가 아닌 softmax 를 활용하였다.

3 TRAINING THE NETWORK

1. **epoch=100 모델**
   - epoch=100, Batch size=256, validation data=test data
   - epoch 수가 높을수록 deep learning model 의 정확도가 높아지는 것은 자명하다. 다만, colab 을 통해 학습을 진행하는 특성상 여러번의 학습시 런타임이 중단되며 무엇보다도 학습 시간이 오래 걸린다. 우리는 정확도와 loss 에 관한 그래프를 통해 test data 에 대한 정확도가 epoch=30 을 넘어가는 순간부터 크게 유의미한 결과가 나오지 않는다는 것을 확인했다. 따라서, 지금 기술한 모델은 feature space visualization 시 dramatic 한 비교 관측을 위해 사용하나 그 외에는 아래에 기술된 방식으로 학습했다.

2. **epoch=30 모델**
   - epoch=30, batch size=256, validation data=test data

- 위 모델과 epoch 수만 줄어들었으며 학습 시간을 절약하기 위해 학습하였다. 해당 모델로 convolution layer 별로 feature space 를 관측했다. Dramatic 한 비교를 위해 바로 위에 기술된 모델을 일부 사용하였다.

4 EXPERIMENTAL RESULTS

- 테스트 환경은 Colab 에서 진행되었으며 TensorFlow 라이브러리를 사용하였다.

1. 정확도 비교
   - conv 는 convolution layer 를 의미하며 이를 토대로 수치를 비교하였다.

Model	Layer	Train data		Test data	
		Loss	Accuracy	Loss	Accuracy
epoch=30	conv_1	1.4811	0.4538	1.4939	0.4469
	conv_2	1.3089	0.5346	1.3432	0.5271
	conv_3	0.9907	0.6471	1.0454	0.6349
	conv_4	0.6528	0.7689	0.7434	0.7427
	conv_5	0.4635	0.8390	0.5787	0.8020
	conv_6	0.3846	0.8661	0.5186	0.8257
	conv_7	0.3217	0.8868	0.4886	0.8323
	conv_8	0.2589	0.9075	0.4646	0.8442
	conv_9	0.1750	0.9401	0.4336	0.8641
epoch=100	conv_9	0.0278	0.9933	0.4117	0.8836

표 1 train data 와 test data 에 따른 loss 와 정확도 비교

- Train data 를 예측한 경우 conv 가 깊어질수록 정확도가 향상된 모습을 보인다. 물론 test data 의 경우도 이와 같으나 conv_6 부터 정확도가 향상되는 폭이 줄어들게 되며 epoch 를 높여도 성능 향상폭이 train data 에 비해 작다.

2. F1-score 비교
   - 같은 convolution layer 일 때, epoch 가 다른 CNN 모델을 비교하였다.

Conv_9 for x_train Classification Report:

	precision	recall	f1-score	support
0	0.95	0.93	0.94	5000
1	0.96	0.99	0.97	5000
2	0.96	0.89	0.93	5000
3	0.87	0.88	0.87	5000
4	0.96	0.92	0.94	5000
5	0.89	0.90	0.89	5000
6	0.95	0.97	0.96	5000
7	0.96	0.97	0.96	5000
8	0.92	0.99	0.96	5000
9	0.98	0.96	0.97	5000
accuracy			0.94	50000
macro avg	0.94	0.94	0.94	50000
weighted avg	0.94	0.94	0.94	50000

CNN epoch=100 for x_train Classification Report:

	precision	recall	f1-score	support
0	0.99	0.99	0.99	5000
1	1.00	1.00	1.00	5000
2	0.99	0.99	0.99	5000
3	0.99	0.98	0.98	5000
4	0.99	0.99	0.99	5000
5	0.98	0.99	0.98	5000
6	1.00	1.00	1.00	5000
7	0.99	1.00	1.00	5000
8	1.00	1.00	1.00	5000
9	1.00	1.00	1.00	5000
accuracy			0.99	50000
macro avg	0.99	0.99	0.99	50000
weighted avg	0.99	0.99	0.99	50000

그림 2 (좌) epoch=30 모델의 train data 분류 리포트, (우) epoch=100 모델의 train data 분류 리포트

   - epoch 수가 높을수록 f1-score 가 높다. 이는 feature 간 분리가 잘 되었음을 의미하며 정확도가 향상되는 지표로 파악할 수 있다.
   - 특정 범주(2:새, 3:고양이, 5:개)가 상대적으로 f1-score 가 낮으며 방향성을 고려한 data augmentation 이 고려된다.

Conv_9 for x_test Classification Report:

	precision	recall	f1-score	support
0	0.88	0.85	0.86	1000
1	0.89	0.96	0.93	1000
2	0.89	0.76	0.82	1000
3	0.74	0.76	0.75	1000
4	0.87	0.83	0.85	1000
5	0.81	0.81	0.81	1000
6	0.87	0.91	0.89	1000
7	0.91	0.91	0.91	1000
8	0.86	0.96	0.91	1000
9	0.93	0.89	0.91	1000
accuracy			0.86	10000
macro avg	0.86	0.86	0.86	10000
weighted avg	0.86	0.86	0.86	10000

CNN epoch=100 for x_test Classification Report:

	precision	recall	f1-score	support
0	0.88	0.85	0.86	1000
1	0.89	0.96	0.93	1000
2	0.89	0.76	0.82	1000
3	0.74	0.76	0.75	1000
4	0.87	0.83	0.85	1000
5	0.81	0.81	0.81	1000
6	0.87	0.91	0.89	1000
7	0.91	0.91	0.91	1000
8	0.86	0.96	0.91	1000
9	0.93	0.89	0.91	1000
accuracy			0.86	10000
macro avg	0.86	0.86	0.86	10000
weighted avg	0.86	0.86	0.86	10000

그림 3 (좌) epoch=30 모델의 test data 분류 리포트, (우) epoch=100 모델의 test data 분류 리포트

   - f1-score1 가 epoch 수와 관계없이 변동하지 않았다. Epoch 수를 늘려 모델을 개선하는 것보다 data preprocessing, data augmentation 또는 특정 범주(2, 3, 5)에 대한 규제를 가해야 f1-score 가 향상될 것으로 보인다.

3. Feature space visualization: PCA, GMM

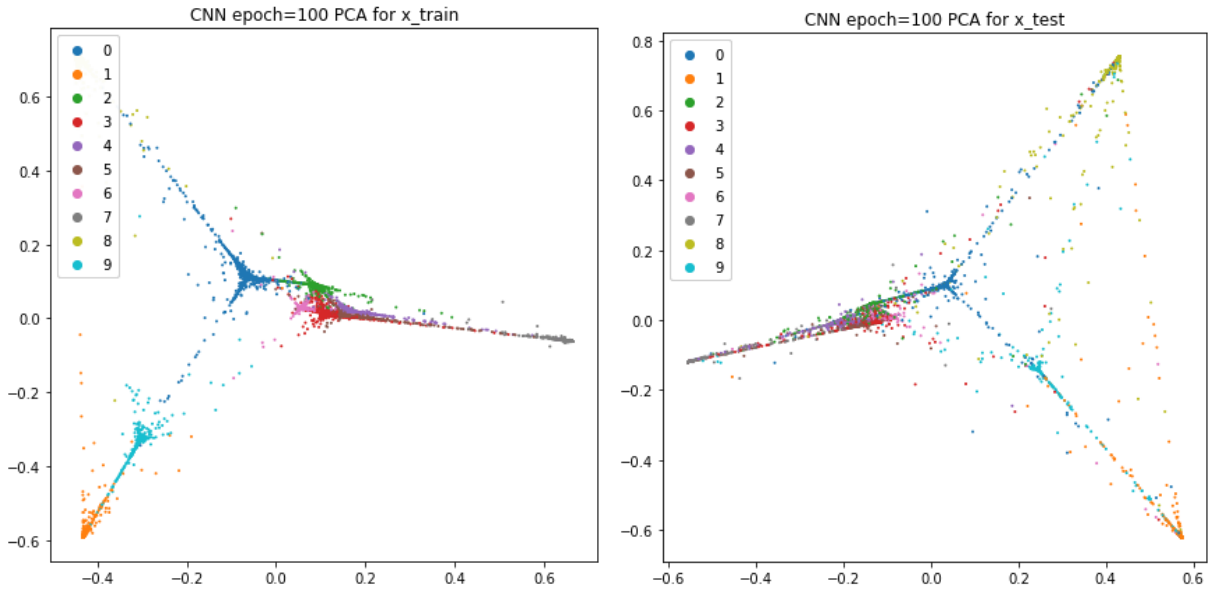

그림 4 (좌) epoch=100 모델의 train data PCA, (우) epoch=100 모델의 test data PCA

- Train data 의 경우 PCA 을 통해 각 feature 들이 잘 분리된 모습을 확인할 수 있다. 단, Test data 의 경우 train data 대비 분리가 덜 된 모습을 관측할 수 있었다.

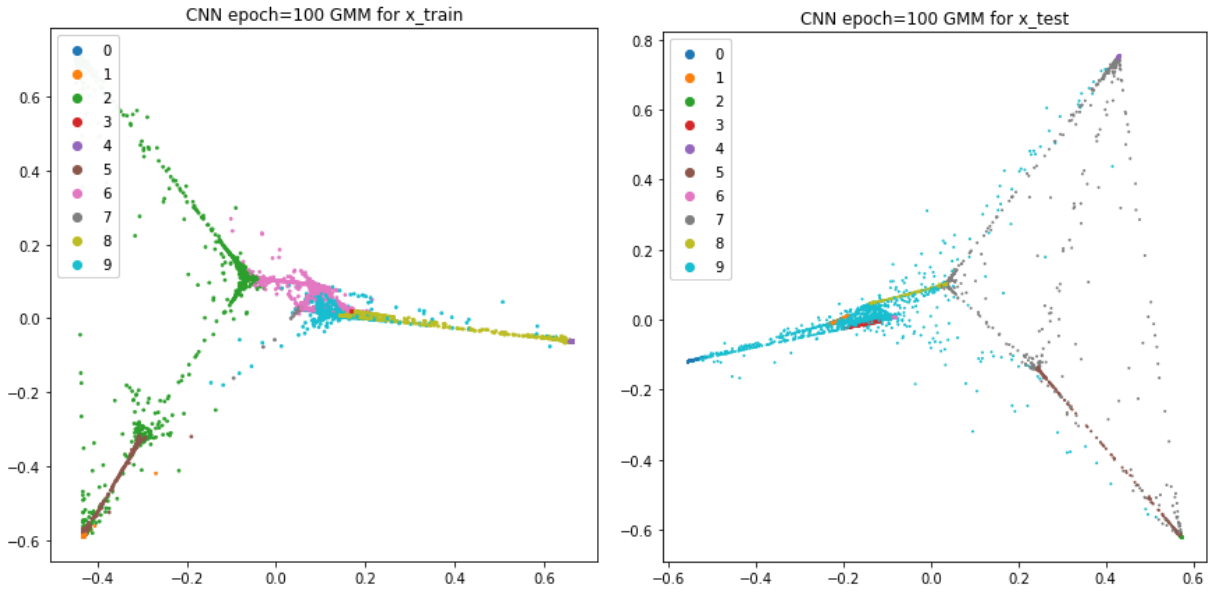

그림 5 (좌) epoch=100 모델의 train data GMM, (우) epoch=100 모델의 test data GMM

- Train data 의 경우 GMM 을 통해 각 feature 들이 잘 분리 되었다. 다만, Test data 의 경우 특정 범주(9)가 뒤덮여 있는데 PCA 를 통해 관측해도 feature 들이 겹쳐 있는 모습을 보아 모델 구조적으로 convolution layer 를 추가하는 식의 개선을 꾀해야 할 필요가 있다.

4. xAI visualization: CAM

그림 6 임의의 데이터에 대한 CAM 가시화

- 총 10 개의 범주에 맞춰 임의의 데이터에 대해 CAM 가시화를 하였다.
- 이 중에서 고양이, 사슴, 배, 트럭의 경우 분류를 제대로 하지 못했다.
  - 고양이의 경우 개라 분류하였다. F1-score 가 낮은 범주지만, 2 순위로 고양이라 분류한 것을 보아 개랑 이목구비(털, 귀 등)가 유사한 동물이므로 방향성을 고려한 data augmentation 시 정확도 향상이 기대된다.
  - 사슴의 경우 새라 분류하였다. 3 순위로 사슴이라 분류하였는데, 새로 분류한 경우 CAM 결과가 사슴이 아닌 사슴주변 수풀을 보고 있었다. 데이터 입력 시 배경(수풀)에 대한 전처리가 가해지면 정확도 향상이 기대된다.
  - 배의 경우 자동차라 분류하였다. 2 순위로 배라 분류하였다. 배의 CAM 결과를 보면 인공지능이 배의 중심보다는 배의 주변을 관측하는 것을 확인할 수 있었다.
  - 트럭의 경우 비행기라 분류하였다. 2 순위로 트럭으로 분류하였는데, CAM 결과만 보았을 때, 트럭 중심을 보았으나 예측 점수가 트럭이 2.0701, 비행기가 2.3425 로 약간 근소하여 분류를 제대로 하지 못하였다. 위에서 언급한 배를 자동차로 분류한 것과 다른 케이스이나 동물이 아닌 사람이 만든 기계로 잘못 분류했다는 것이 공통점으로 확인됐다.

5 DISCUSSION AND CONCLUSION

전반적으로 우리가 만든 모델이 수치나 그래프를 봤을 때 만족스러웠다. 하지만, CAM 을 통해 이미지 분류를 해보니 데이터가 10 개라는 것을 감안해도 잘 분류한다는 느낌은 아니었다. 하지만, 이번 프로젝트를 통해 xAI 와 인공지능의 동작 원리에 대해 알 수 있었고 성장했다는 것에 의의를 두었다.