IS/2021

다양한 기법을 활용한 CIFAR 10 Classification 성능 향상

- CNN -

안태진(taejin7824@gmail.com)
GitHub(github.com/taejin1221)
상명대학교 소프트웨어학과
201821002

Contents

- 프로젝트 개요
- 방법
  - 실험 방법 및 환경
  - 4 Layer CNN
  - 6-7 Layer CNN
  - 다양한 기법을 적용한 CNN
  - 성능 더 높이기
- 결론
- 한계 및 추후 연구

Contents

프로젝트 개요

- 주제 선택 이유
  - CNN?
    - 평소 CNN을 이용한 classification에 대한 관심이 많았음
    - 최근의 수업에서 CNN에 대해 공부하면서 직접 모델을 만들어보고 싶다고 생각
  - CIFAR10?
    - 다양한 수업에서 예제로 많이 사용하는 MNIST를 제외한 다른 데이터를 사용하고 싶다고 생각
    - 너무 쉽지도 않지만 너무 어렵지도 않고 쉽게 얻을 수 있는 CIFAR10을 데이터로 선택

프로젝트 개요

- 프로젝트 설명
  - CNN을 이용한 CIFAR 10 classification을 구현 및 성능을 최대한 높임
    - 기본 방향은 성능 높이기
  - 최대한 기본적인 구조로 어느정도 성능을 낼 수 있는지를 분석
    - Sequential한 구조로 4 ~ 12개의 layer를 가지는 CNN
  - 다양한 기법들을 적용하여 여러 개의 기법을 적용하였을 때 성능을 비교
    - 성능 지표
      - Test loss, test accuracy
      - Overfitting 정도
        - 직접적인 수치로 나타내기 어렵지만 accuracy graph를 보고 바로 확인 가능하므로 눈으로 판단

Contents

방법

- 실험 방법 및 환경
  - 실험 방법
    - 다양한 모델을 만들어 학습을 시킨 뒤 loss, accuracy graph를 그려 성능 확인
    - 다양한 기법들을 적용시키고, 이전 장에서 설명한 성능 지표가 어떤 식으로 변하는지 확인
      - e.g., Layer 증가, filter 변경, Dropout, Batch Normalization 등
    - 총 17 번의 실험을 진행
  - 실험 환경
    - 오른쪽 Table 1 참고

Table 1: Experimental Setup Overview

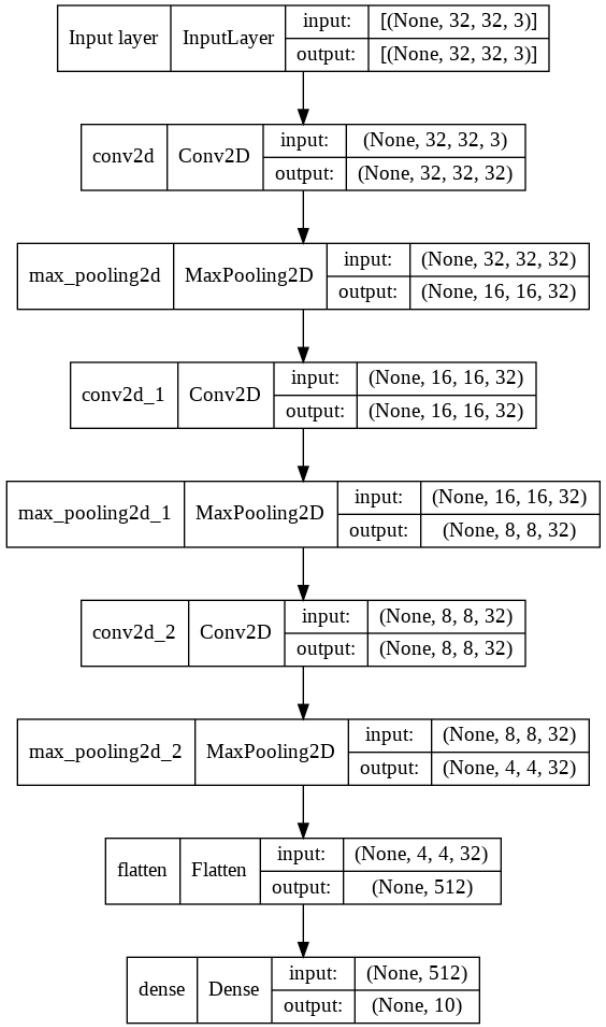
실험 환경	**Version**
OS and Device	Google Colab
Batch Size	512
Epochs	300
Optimizer and LR	Adam, 0.001
Loss	Categorical Crossentropy

Contents

방법

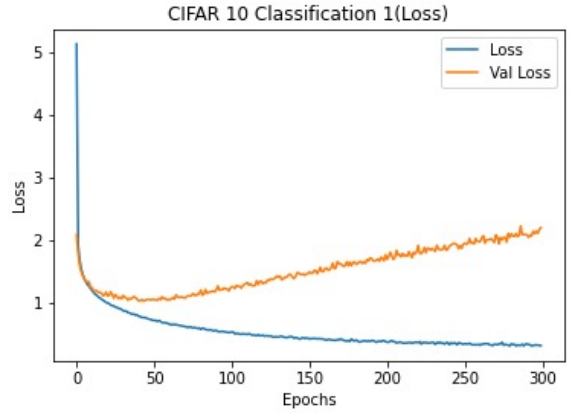
- 4 Layer CNN (1/4)
  - 4-layer model 1
    - 가장 기본적인 4 Layer CNN 구축
      - C-P-C-P-C-P-FC

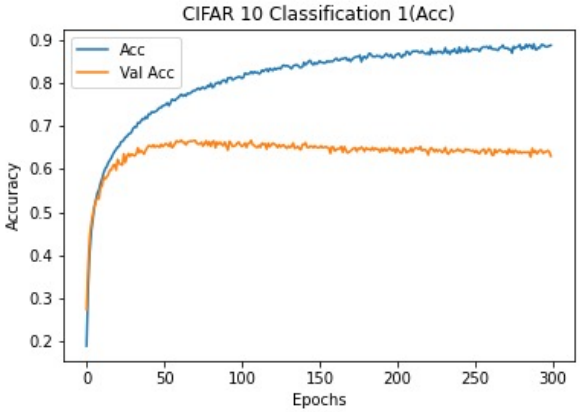
```
# Use keras functional API
input = layers.Input((HEIGHT, WIDTH, CHANNEL), name = 'Input layer')

x = layers.Conv2D(32, (3, 3), padding = 'same', activation = 'relu')(input)
x = layers.MaxPooling2D((3, 3), (2, 2), padding = 'same')(x)

x = layers.Conv2D(32, (3, 3), padding = 'same', activation = 'relu')(x)
x = layers.MaxPooling2D((3, 3), (2, 2), padding = 'same')(x)

x = layers.Conv2D(32, (3, 3), padding = 'same', activation = 'relu')(x)
x = layers.MaxPooling2D((3, 3), (2, 2), padding = 'same')(x)

x = layers.Flatten()(x)
output = layers.Dense(NUM_CLASS, activation = 'softmax')(x)
```



방법

- 4 Layer CNN (2/4)
  - 4-layer model 1
    - Loss (Test Loss: 2.1981)
      - Epoch가 증가할 수록 validation loss가 증가
      - 학습 진행 X
    - Accuracy (Test Accuracy: 63.00%)
      - Epoch가 증가할 수록 training acc는 증가하지만 val acc는 정체
      - Train accuracy는 약 88%를 달성했지만 test acc는 63%로 저조
        - overfitting이 발생





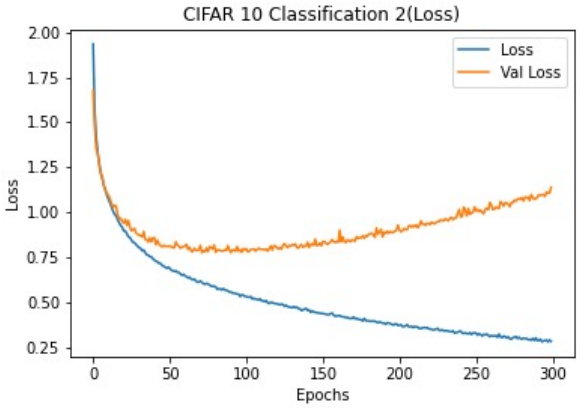
방법

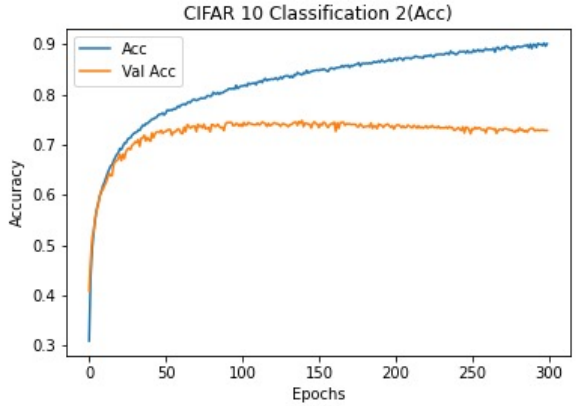
- 4 Layer CNN (3/4)
  - 4-layer model 2
    - 이전과 완전 동일한 모델에 data normalization만 적용
    - x data를 255.0으로 나눠줌으로 normalization 적용

```
(x_train, y_train), (x_test, y_test) = keras.datasets.cifar10.load_data()
x_train, x_test = x_train.astype('float32') / 255., x_test.astype('float32') / 255.
y_train, y_test = keras.utils.to_categorical(y_train, NUM_CLASS), keras.utils.to_categorical(y_test, NUM_CLASS)
```

방법

- 4 Layer CNN (4/4)
  - 4-layer model 2
    - Loss (Test Loss: 1.1398)
      - 이전과 동일하게 Epoch가 증가할 수록 validation loss가 증가
      - 하지만 이전 모델보다 loss는 줄어듬
    - Accuracy (Test Accuracy: 72.87%)
      - 이전과 동일하게 overfitting이 발생
        - Train acc 90% vs test acc 73%
      - 하지만 accuracy는 약 9.87% 증가



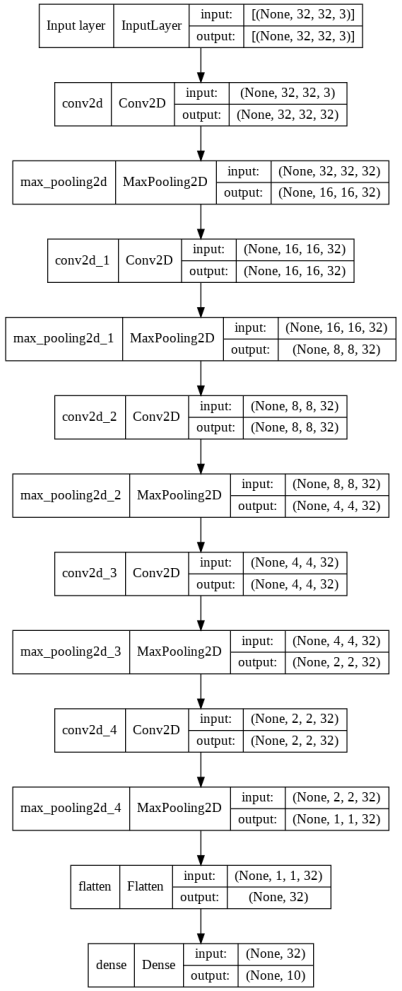


Contents

방법

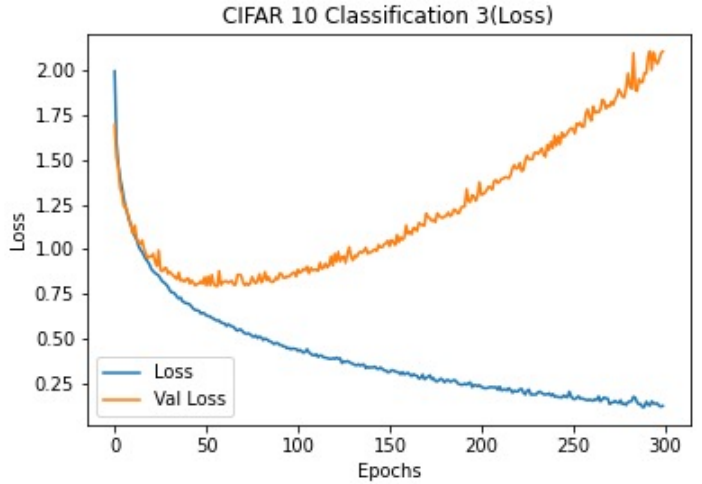
- 6 Layer CNN (1/6)
  - 6-layer model 1
    - 만족할만한 성능이 아니기 때문에 layer를 증가 시키기로 함

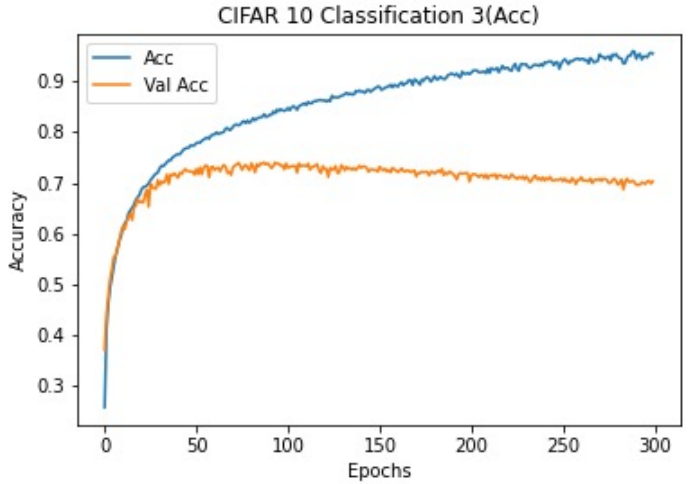
```
input = layers.Input((HEIGHT, WIDTH, CHANNEL), name = 'Input layer')

x = layers.Conv2D(32, (3, 3), padding = 'same', activation = 'relu')(input)
x = layers.MaxPooling2D((3, 3), (2, 2), padding = 'same')(x)

x = layers.Conv2D(32, (3, 3), padding = 'same', activation = 'relu')(x)
x = layers.MaxPooling2D((3, 3), (2, 2), padding = 'same')(x)

x = layers.Conv2D(32, (3, 3), padding = 'same', activation = 'relu')(x)
x = layers.MaxPooling2D((3, 3), (2, 2), padding = 'same')(x)

x = layers.Conv2D(32, (3, 3), padding = 'same', activation = 'relu')(x)
x = layers.MaxPooling2D((3, 3), (2, 2), padding = 'same')(x)

x = layers.Conv2D(32, (3, 3), padding = 'same', activation = 'relu')(x)
x = layers.MaxPooling2D((3, 3), (2, 2), padding = 'same')(x)

x = layers.Flatten()(x)
output = layers.Dense(NUM_CLASS, activation = 'softmax')(x)
```



방법

- 6 Layer CNN (2/6)
  - 6-layer model 1
    - Loss (2.1026)
      - 이전과는 다르게 loss가 폭발적으로 증가
      - 이전 모델의 loss보다 loss가 더 높아짐
    - Accuracy (70.27%)
      - 이전 모델(72.87%)보다 오히려 성능이 낮아지는 결과 초래

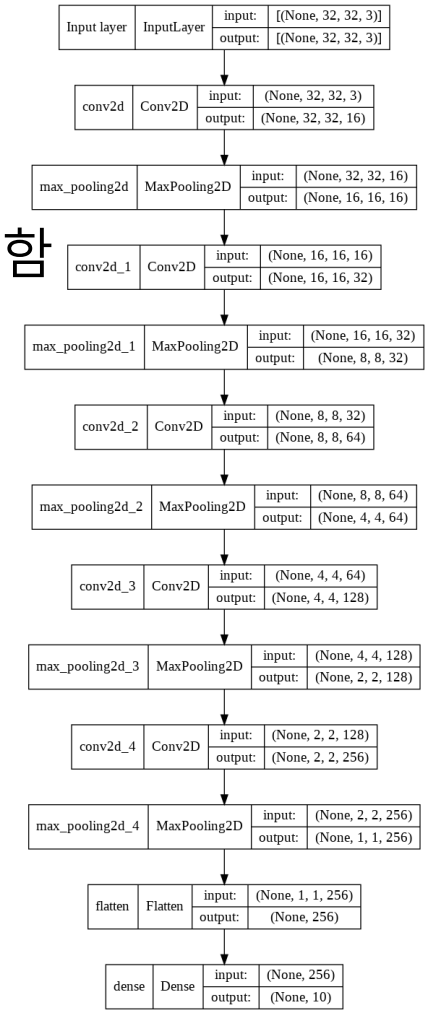




방법

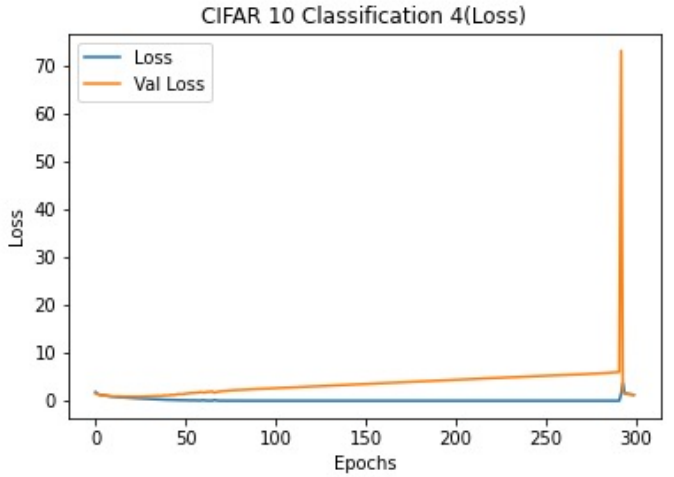
- 6 Layer CNN (3/6)
  - 6-layer model 2
    - 예상치 못한 결과가 나와서 문제가 있다고 판단, 이번엔 filter를 변경해보기로 함

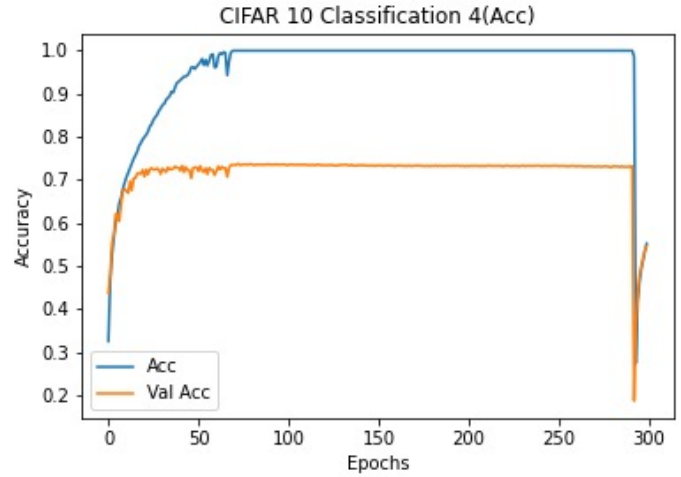
```
input = layers.Input((HEIGHT, WIDTH, CHANNEL), name = 'Input layer')

x = layers.Conv2D(16, (3, 3), padding = 'same', activation = 'relu')(input)
x = layers.MaxPooling2D((3, 3), (2, 2), padding = 'same')(x)

x = layers.Conv2D(32, (3, 3), padding = 'same', activation = 'relu')(x)
x = layers.MaxPooling2D((3, 3), (2, 2), padding = 'same')(x)

x = layers.Conv2D(64, (3, 3), padding = 'same', activation = 'relu')(x)
x = layers.MaxPooling2D((3, 3), (2, 2), padding = 'same')(x)

x = layers.Conv2D(128, (3, 3), padding = 'same', activation = 'relu')(x)
x = layers.MaxPooling2D((3, 3), (2, 2), padding = 'same')(x)

x = layers.Conv2D(256, (3, 3), padding = 'same', activation = 'relu')(x)
x = layers.MaxPooling2D((3, 3), (2, 2), padding = 'same')(x)

x = layers.Flatten()(x)
output = layers.Dense(NUM_CLASS, activation = 'softmax')(x)
```



방법

- 6 Layer CNN (4/6)
  - 6-layer model 2
    - Loss (1.2527)
      - 역시나 loss가 증가하지만 갑자기 너무 폭발적으로 증가
    - Accuracy (54.60%, 73%)
      - Loss와 동일하게 어느 순간 갑자기 train, validation acc가 폭발적으로 감소
      - 최대 test accuracy는 약 73%
        - 이때의 Loss는 6.0071
        - Loss는 높지만 오히려 이때의 Acc가 제일 높음
        - 이때부터 Train Accuracy는 100%가 나오기 시작

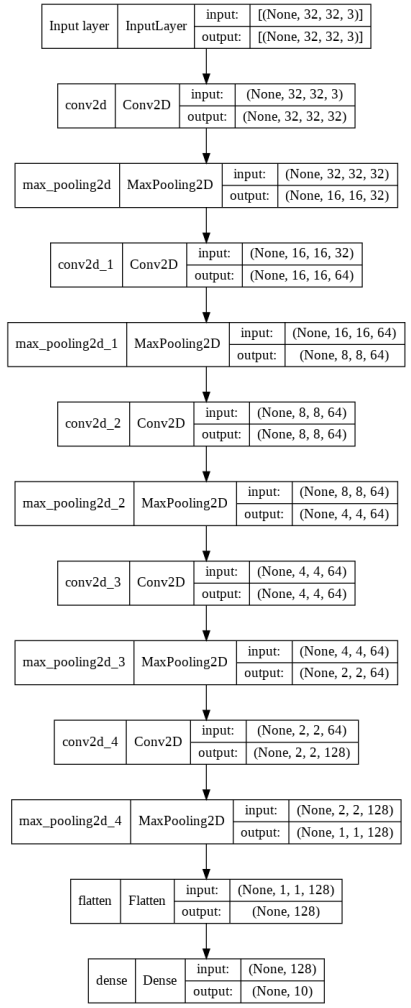




방법

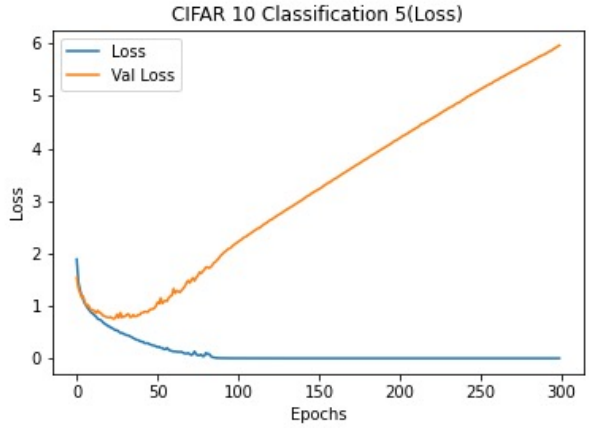
- 6 Layer CNN (5/6)
  - 6-layer model 3
    - 다른 방법으로 filter를 변경

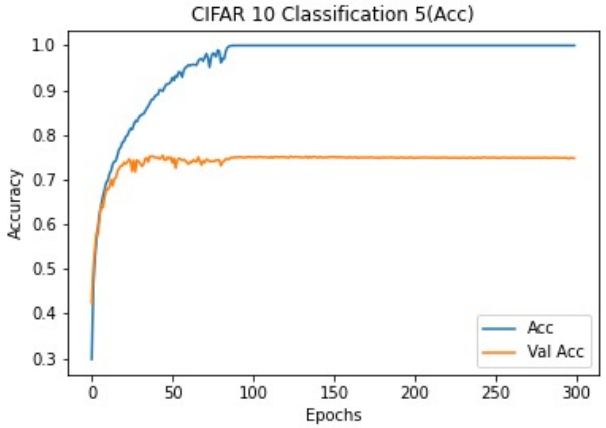
```
input = layers.Input((HEIGHT, WIDTH, CHANNEL), name = 'Input layer')

x = layers.Conv2D(32, (3, 3), padding = 'same', activation = 'relu')(input)
x = layers.MaxPooling2D((3, 3), (2, 2), padding = 'same')(x)

x = layers.Conv2D(64, (3, 3), padding = 'same', activation = 'relu')(x)
x = layers.MaxPooling2D((3, 3), (2, 2), padding = 'same')(x)

x = layers.Conv2D(64, (3, 3), padding = 'same', activation = 'relu')(x)
x = layers.MaxPooling2D((3, 3), (2, 2), padding = 'same')(x)

x = layers.Conv2D(64, (3, 3), padding = 'same', activation = 'relu')(x)
x = layers.MaxPooling2D((3, 3), (2, 2), padding = 'same')(x)

x = layers.Conv2D(128, (3, 3), padding = 'same', activation = 'relu')(x)
x = layers.MaxPooling2D((3, 3), (2, 2), padding = 'same')(x)

x = layers.Flatten()(x)
output = layers.Dense(NUM_CLASS, activation = 'softmax')(x)
```



방법

- 6 Layer CNN (6/6)
  - 6-layer model 3
    - Loss (5.9658)
      - Validation loss가 엄청 증가...
    - Accuracy (74.77%)
      - Accuracy는 이전 모델의 최고 성능보다 1% 증가했지만 우연에 의한 것이라고 생각
      - Train Accuacy는 Epoch 90부터 100% 달성

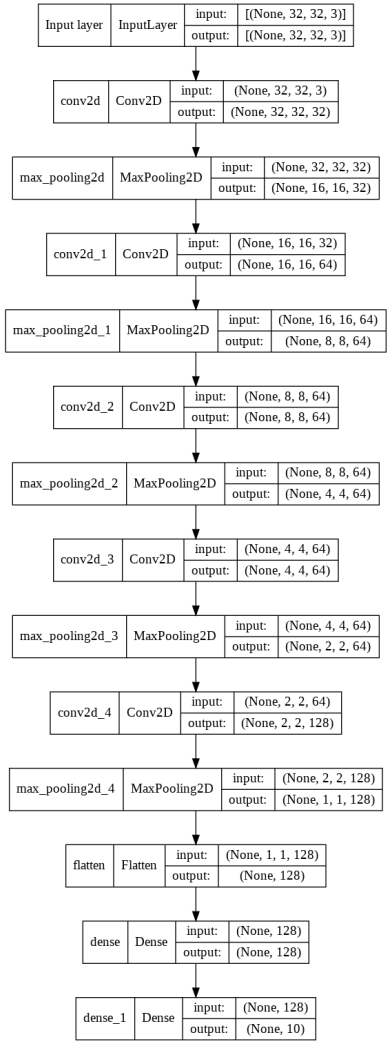




방법

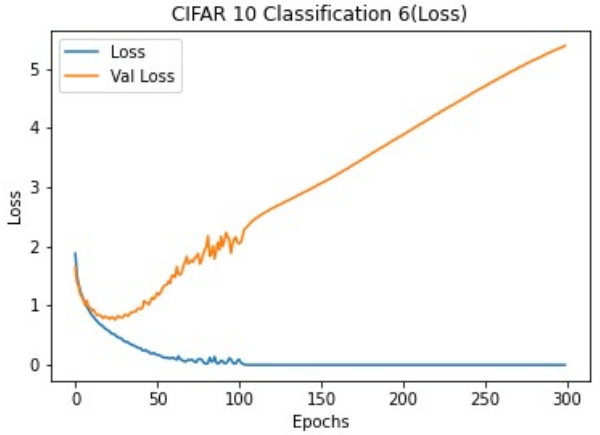
- 7 Layer CNN (1/2)
  - 7-layer model 1 (Basic 7-layer CNN Model)
    - Loss가 이상하게 증가해서 분류기 문제인지 생각하여 분류기를 1 Layer 증가

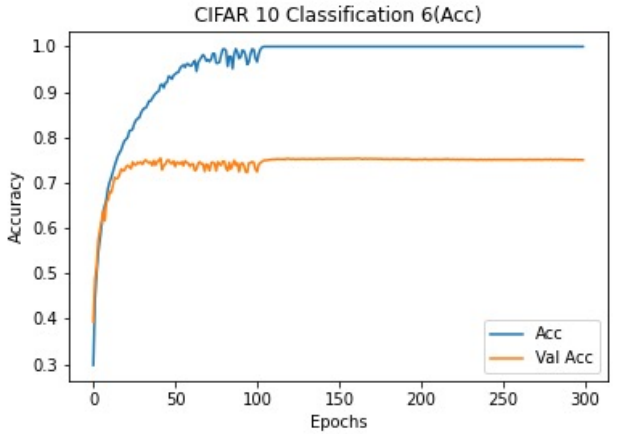
```
input = layers.Input((HEIGHT, WIDTH, CHANNEL), name = 'Input layer')

x = layers.Conv2D(32, (3, 3), padding = 'same', activation = 'relu')(input)
x = layers.MaxPooling2D((3, 3), (2, 2), padding = 'same')(x)

x = layers.Conv2D(64, (3, 3), padding = 'same', activation = 'relu')(x)
x = layers.MaxPooling2D((3, 3), (2, 2), padding = 'same')(x)

x = layers.Conv2D(64, (3, 3), padding = 'same', activation = 'relu')(x)
x = layers.MaxPooling2D((3, 3), (2, 2), padding = 'same')(x)

x = layers.Conv2D(64, (3, 3), padding = 'same', activation = 'relu')(x)
x = layers.MaxPooling2D((3, 3), (2, 2), padding = 'same')(x)

x = layers.Conv2D(128, (3, 3), padding = 'same', activation = 'relu')(x)
x = layers.MaxPooling2D((3, 3), (2, 2), padding = 'same')(x)

x = layers.Flatten()(x)
x = layers.Dense(128, activation = 'relu')(x)

output = layers.Dense(NUM_CLASS, activation = 'softmax')(x)
```



방법

- 7 Layer CNN (2/2)
  - 7-layer model 1 (Basic 7-layer CNN Model)
    - Loss (5.3863)
      - Loss는 이전과 동일
    - Accuracy (75.02%)
      - Accuracy는 이전보다 1% 증가





Contents

방법

- 다양한 기법을 적용한 CNN (1/11)
  - 이 시점부터 layer, filter 증가보단 기법을 적용하는 것이 옳다고 판단

  - 따라서 다양한 성능 향상 기법 적용
    1. Weight initialization
    2. Batch Normalization
    3. Dropout

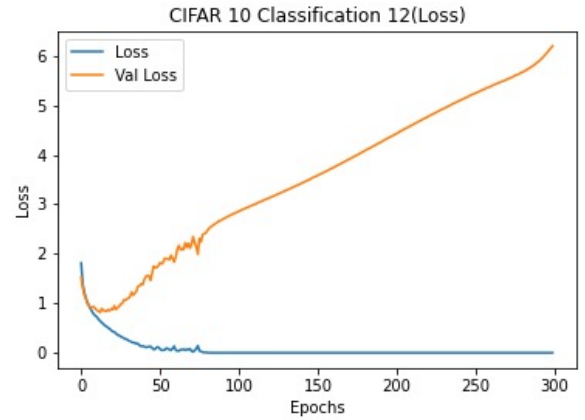
방법

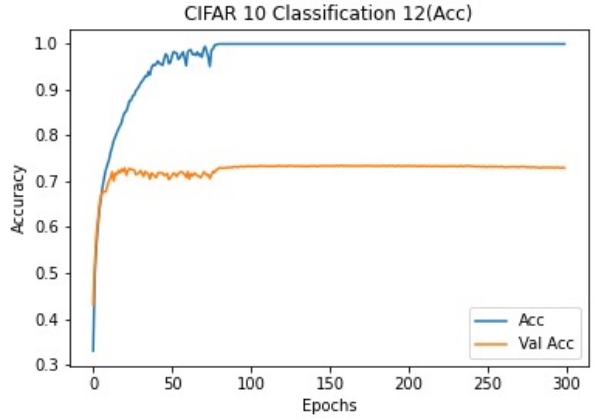
- 다양한 기법을 적용한 CNN (2/11)
  - WeightInit model 1
    - Basic 7-layer model에 weight initialization을 적용한 모델
      - he uniform initialization을 적용

```
input = layers.Input((HEIGHT, WIDTH, CHANNEL), name = 'Input layer')

x = layers.Conv2D(32, (3, 3), padding = 'same', activation = 'relu', kernel_initializer = 'he_uniform')(input)
x = layers.MaxPooling2D((3, 3), (2, 2), padding = 'same')(x)

x = layers.Conv2D(64, (3, 3), padding = 'same', activation = 'relu', kernel_initializer = 'he_uniform')(x)
x = layers.MaxPooling2D((3, 3), (2, 2), padding = 'same')(x)

x = layers.Conv2D(64, (3, 3), padding = 'same', activation = 'relu', kernel_initializer = 'he_uniform')(x)
x = layers.MaxPooling2D((3, 3), (2, 2), padding = 'same')(x)

x = layers.Conv2D(64, (3, 3), padding = 'same', activation = 'relu', kernel_initializer = 'he_uniform')(x)
x = layers.MaxPooling2D((3, 3), (2, 2), padding = 'same')(x)

x = layers.Conv2D(128, (3, 3), padding = 'same', activation = 'relu', kernel_initializer = 'he_uniform')(x)
x = layers.MaxPooling2D((3, 3), (2, 2), padding = 'same')(x)

x = layers.Flatten()(x)
x = layers.Dense(128, activation = 'relu')(x)

output = layers.Dense(NUM_CLASS, activation = 'softmax')(x)
```

방법

- 다양한 기법을 적용한 CNN (3/11)
  - WeightInit model 1
    - Loss (6.1982)
    - Accuracy (72.94%)
    - 전반적으로 이전 모델보다 성능이 안좋아짐
      - Conv2D layer가 기본적으로 glorot_uniform으로 초기화함
      - 따라서 성능 향상이 없는 것으로 생각





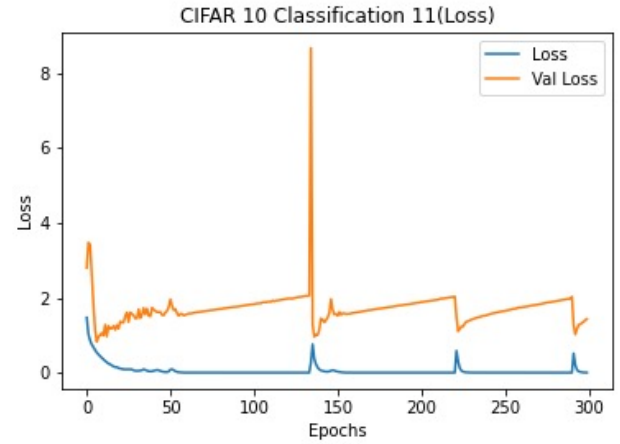
방법

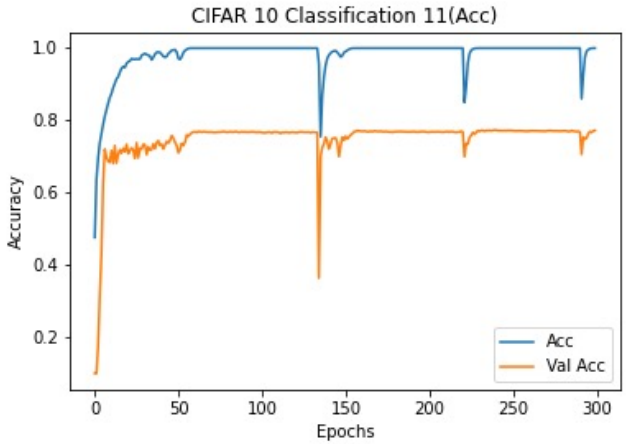
- 다양한 기법을 적용한 CNN (4/11)
  - BatchNorm model 1
    - Basic 7-layer model에 Batch Normalization을 적용한 모델

```
input = layers.Input((HEIGHT, WIDTH, CHANNEL), name = 'Input layer')

x = layers.Conv2D(32, (3, 3), padding = 'same', activation = 'relu')(input)
x = layers.BatchNormalization()(x)
x = layers.MaxPooling2D((3, 3), (2, 2), padding = 'same')(x)

x = layers.Conv2D(64, (3, 3), padding = 'same', activation = 'relu')(x)
x = layers.BatchNormalization()(x)
x = layers.MaxPooling2D((3, 3), (2, 2), padding = 'same')(x)

x = layers.Conv2D(64, (3, 3), padding = 'same', activation = 'relu')(x)
x = layers.BatchNormalization()(x)
x = layers.MaxPooling2D((3, 3), (2, 2), padding = 'same')(x)

x = layers.Conv2D(64, (3, 3), padding = 'same', activation = 'relu')(x)
x = layers.BatchNormalization()(x)
x = layers.MaxPooling2D((3, 3), (2, 2), padding = 'same')(x)

x = layers.Conv2D(128, (3, 3), padding = 'same', activation = 'relu')(x)
x = layers.BatchNormalization()(x)
x = layers.MaxPooling2D((3, 3), (2, 2), padding = 'same')(x)

x = layers.Flatten()(x)
x = layers.Dense(128, activation = 'relu')(x)
x = layers.BatchNormalization()(x)

output = layers.Dense(NUM_CLASS, activation = 'softmax')(x)
```

방법

- 다양한 기법을 적용한 CNN (5/11)
  - BatchNorm model 1
    - Loss (1.4304)
    - Accuracy (77.27%)
    - 기존 모델보다 성능이 좋아짐을 확인 가능
    - 하지만 여전히 overfitting은 해결하지 못했으며,
중간 중간 성능이 굉장히 떨어지는 계곡 같은 구간이 존재





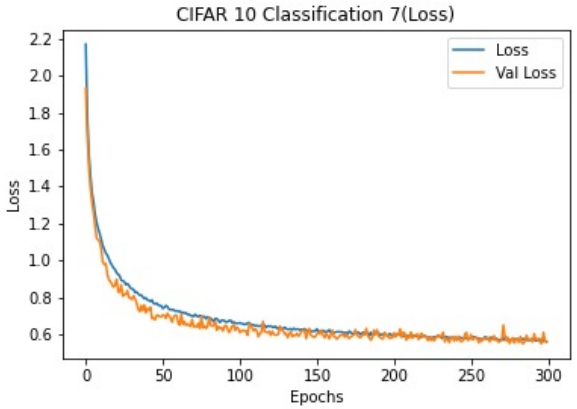
방법

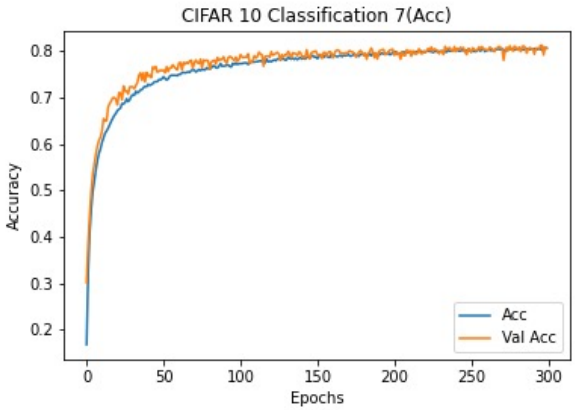
- 다양한 기법을 적용한 CNN (6/11)
  - Dropout model 1
    - Basic 7-layer model에 Dropout을 적용한 모델
      - Dropout rate는 0.3을 선택

```
input = layers.Input((HEIGHT, WIDTH, CHANNEL), name = 'Input layer')

x = layers.Conv2D(32, (3, 3), padding = 'same', activation = 'relu')(input)
x = layers.MaxPooling2D((3, 3), (2, 2), padding = 'same')(x)
x = layers.Dropout(0.3)(x)

x = layers.Conv2D(64, (3, 3), padding = 'same', activation = 'relu')(x)
x = layers.MaxPooling2D((3, 3), (2, 2), padding = 'same')(x)
x = layers.Dropout(0.3)(x)

x = layers.Conv2D(64, (3, 3), padding = 'same', activation = 'relu')(x)
x = layers.MaxPooling2D((3, 3), (2, 2), padding = 'same')(x)
x = layers.Dropout(0.3)(x)

x = layers.Conv2D(64, (3, 3), padding = 'same', activation = 'relu')(x)
x = layers.MaxPooling2D((3, 3), (2, 2), padding = 'same')(x)
x = layers.Dropout(0.3)(x)

x = layers.Conv2D(128, (3, 3), padding = 'same', activation = 'relu')(x)
x = layers.MaxPooling2D((3, 3), (2, 2), padding = 'same')(x)
x = layers.Dropout(0.3)(x)

x = layers.Flatten()(x)
x = layers.Dense(128, activation = 'relu')(x)
x = layers.Dropout(0.3)(x)

output = layers.Dense(NUM_CLASS, activation = 'softmax')(x)
```

방법

- 다양한 기법을 적용한 CNN (7/11)
  - Dropout model 1
    - Loss (0.5568)
      - Loss가 train loss와 굉장히 비슷해짐
        - Overfitting이 해결
    - Accuracy (80.67%)
      - Test accuracy 또한 약 5% 증가
      - Dropout으로 overfitting이 해결됨이 동시에 Acc도 올라감





방법

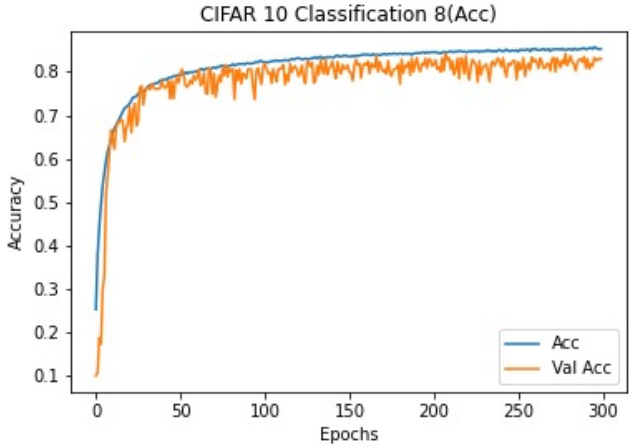
- 다양한 기법을 적용한 CNN (8/11)
  - Complex model 1, 2
    - Dropout을 적용한 모델에 BatchNorm, WeightInit을 각각 적용한 모델

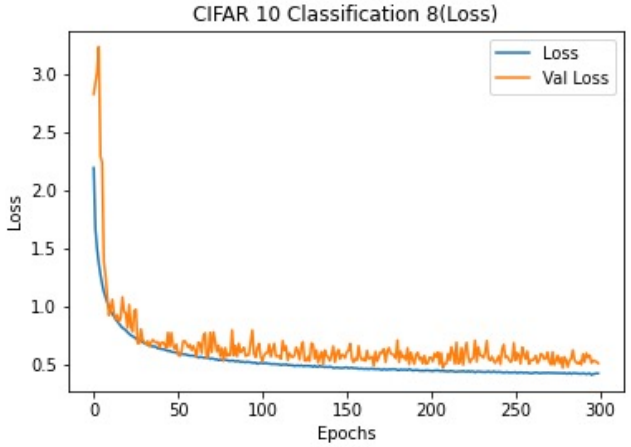
```
input = layers.Input((HEIGHT, WIDTH, CHANNEL), name = 'Input layer')

x = layers.Conv2D(32, (3, 3), padding = 'same', activation = 'relu')(input)
x = layers.BatchNormalization()(x)
x = layers.MaxPooling2D((3, 3), (2, 2), padding = 'same')(x)
x = layers.Dropout(0.3)(x)

x = layers.Conv2D(64, (3, 3), padding = 'same', activation = 'relu')(x)
x = layers.BatchNormalization()(x)
x = layers.MaxPooling2D((3, 3), (2, 2), padding = 'same')(x)
x = layers.Dropout(0.3)(x)

x = layers.Conv2D(64, (3, 3), padding = 'same', activation = 'relu')(x)
x = layers.BatchNormalization()(x)
x = layers.MaxPooling2D((3, 3), (2, 2), padding = 'same')(x)
x = layers.Dropout(0.3)(x)

x = layers.Conv2D(64, (3, 3), padding = 'same', activation = 'relu')(x)
x = layers.BatchNormalization()(x)
x = layers.MaxPooling2D((3, 3), (2, 2), padding = 'same')(x)
x = layers.Dropout(0.3)(x)

x = layers.Conv2D(128, (3, 3), padding = 'same', activation = 'relu')(x)
x = layers.BatchNormalization()(x)
x = layers.MaxPooling2D((3, 3), (2, 2), padding = 'same')(x)
x = layers.Dropout(0.3)(x)

x = layers.Flatten()(x)
x = layers.Dense(128, activation = 'relu')(x)
x = layers.BatchNormalization()(x)
x = layers.Dropout(0.3)(x)

output = layers.Dense(NUM_CLASS, activation = 'softmax')(x)
```

```
input = layers.Input((HEIGHT, WIDTH, CHANNEL), name = 'Input layer')

x = layers.Conv2D(32, (3, 3), padding = 'same', activation = 'relu', kernel_initializer = 'he_uniform')(input)
x = layers.BatchNormalization()(x)
x = layers.MaxPooling2D((3, 3), (2, 2), padding = 'same')(x)
x = layers.Dropout(0.3)(x)

x = layers.Conv2D(64, (3, 3), padding = 'same', activation = 'relu', kernel_initializer = 'he_uniform')(x)
x = layers.BatchNormalization()(x)
x = layers.MaxPooling2D((3, 3), (2, 2), padding = 'same')(x)
x = layers.Dropout(0.3)(x)

x = layers.Conv2D(64, (3, 3), padding = 'same', activation = 'relu', kernel_initializer = 'he_uniform')(x)
x = layers.BatchNormalization()(x)
x = layers.MaxPooling2D((3, 3), (2, 2), padding = 'same')(x)
x = layers.Dropout(0.3)(x)

x = layers.Conv2D(64, (3, 3), padding = 'same', activation = 'relu', kernel_initializer = 'he_uniform')(x)
x = layers.BatchNormalization()(x)
x = layers.MaxPooling2D((3, 3), (2, 2), padding = 'same')(x)
x = layers.Dropout(0.3)(x)

x = layers.Conv2D(128, (3, 3), padding = 'same', activation = 'relu', kernel_initializer = 'he_uniform')(x)
x = layers.BatchNormalization()(x)
x = layers.MaxPooling2D((3, 3), (2, 2), padding = 'same')(x)
x = layers.Dropout(0.3)(x)

x = layers.Flatten()(x)
x = layers.Dense(128, activation = 'relu')(x)
x = layers.BatchNormalization()(x)
x = layers.Dropout(0.3)(x)

output = layers.Dense(NUM_CLASS, activation = 'softmax')(x)
```

방법

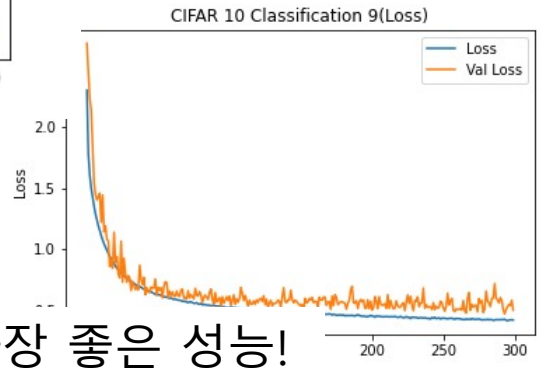
- 다양한 기법을 적용한 CNN (9/11)
  - Complex model 1, 2
    - 왼쪽은 BatchNorm을 적용한, 오른쪽은 BatchNorm + WeightInit을 적용한 모델

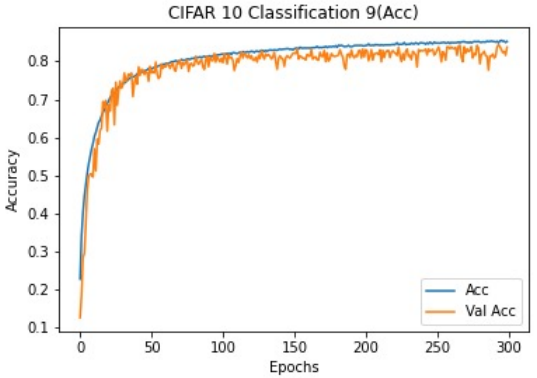




Loss: 0.5087, Acc: 82.98%

Loss: 0.5005, Acc: 83.75%





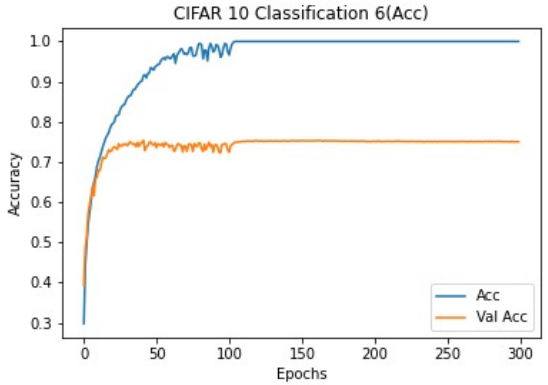
- Dropout, BatchNorm, WeightInit을 모두 적용한 모델이 가장 좋은 성능!

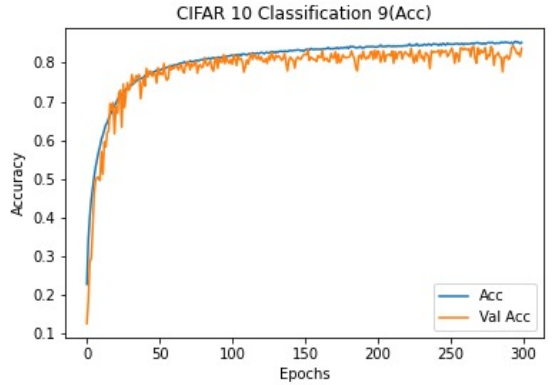
방법

- 다양한 기법을 적용한 CNN (10/11)
  - Dropout, batch normalization, weight initialization을 다양한 방식으로 이용

  - 기존의 7-layer model의 성능 75.02%에서
  - 위의 기법을 모두 적용해 83.75%로 성능 향상

  - 약 8.73%의 acc 증가를 이뤄내고 overfitting을 해결





방법

- 다양한 기법을 적용한 CNN (11/11)
  - 이 외에 다양한 실험을 진행하였지만, 주목할만한 결과만 언급
    - e.g., BatchNorm layer뒤에 activation function 적용, Dropout + WeightInit 만 적용
  - 자세한 실험 결과는 아래 링크 참조
    - https://github.com/Taejin1221/CIFAR10_Classification

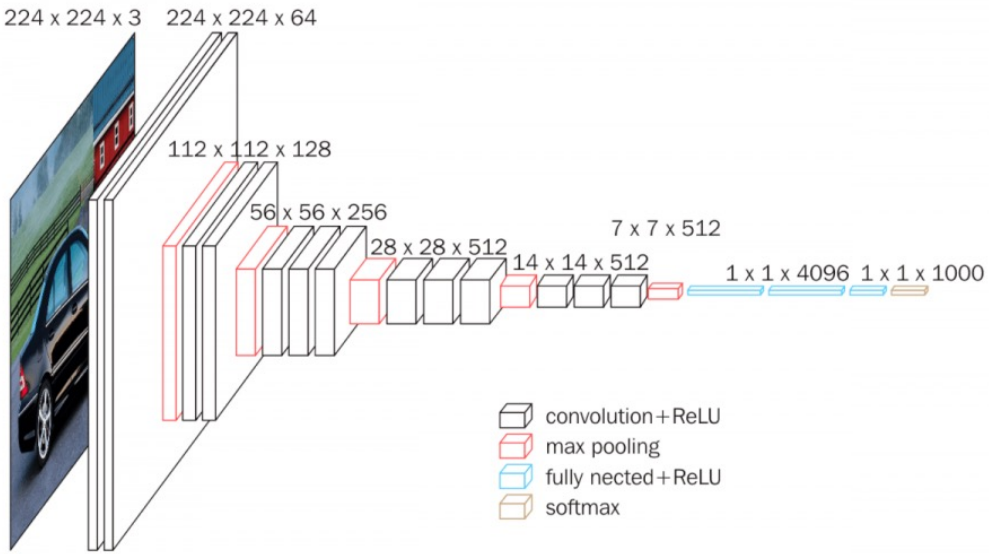
Contents

방법

- 성능 더 높이기 (1/7)
  - 여기서 만족하지 않고 3가지 실험을 더 진행

1. Data Augmentation
   - 회전, flip 등을 이용하여 데이터의 개수를 증가

1. VGG like Model
   - 오른쪽 그림처럼 C-C-P-C-C-P... 으로 신경망 구성



방법

- 성능 더 높이기 (2/7)
  - Data augmentation
    - keras의 ImageDataGenerator를 이용해 augmentation을 진행

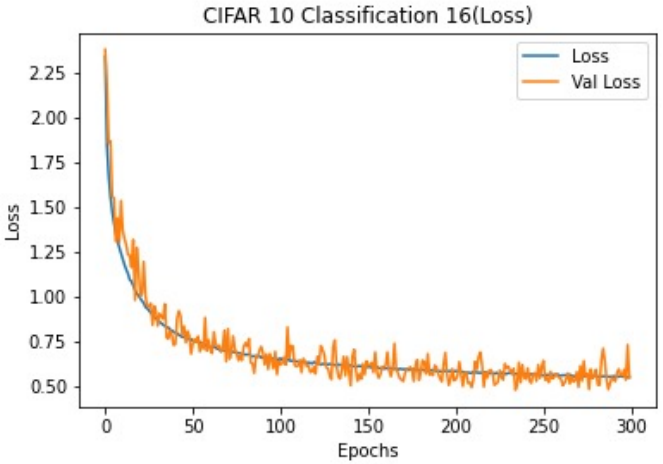
```
datagen = keras.preprocessing.image.ImageDataGenerator(width_shift_range = 0.1, height_shift_range = 0.1, horizontal_flip = True)
iter_train = datagen.flow(x_train, y_train, batch_size = BATCH_SIZE)
```

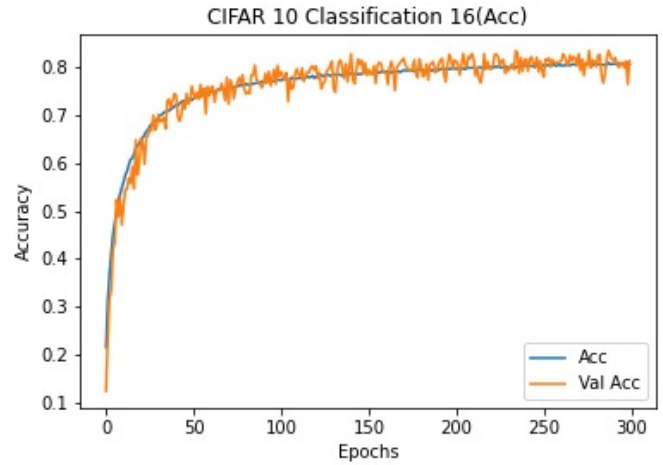
...

```
steps = int(x_train.shape[0] / BATCH_SIZE)
history = model.fit(iter_train, steps_per_epoch = steps, epochs = EPOCHS, validation_data = (x_test, y_test))
```

방법

- 성능 더 높이기 (3/7)
  - Data augmentation
    - Loss (0.5521)
    - Accuracy (81.40%)
    - 오히려 떨어진 성능을 확인할 수 있음





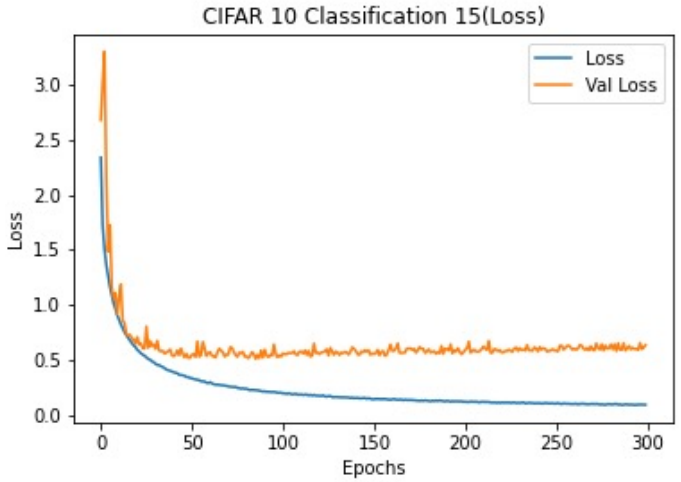
방법

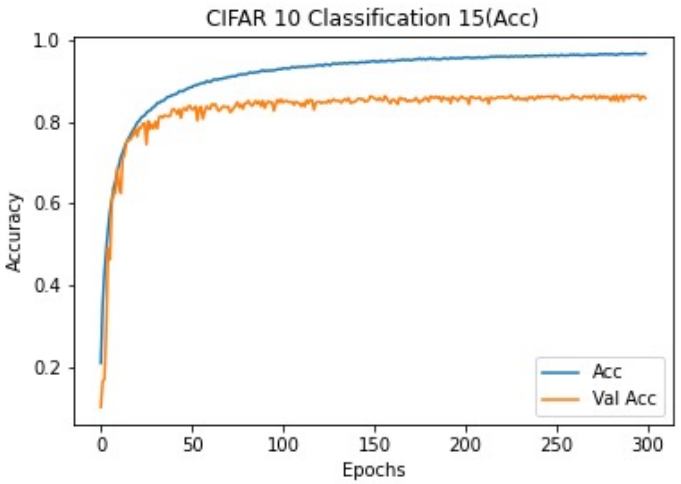
- 성능 더 높이기 (4/7)
  - VGG like Model
    - 기존의 Conv2D layer 뒤에
    - 똑같은 Conv2D layer 추가

```
input = layers.Input((HEIGHT, WIDTH, CHANNEL), name = 'Input layer')

x = layers.Conv2D(32, (3, 3), padding = 'same', activation = 'relu', kernel_initializer = 'he_uniform')(input)
x = layers.Conv2D(32, (3, 3), padding = 'same', activation = 'relu', kernel_initializer = 'he_uniform')(x)
x = layers.BatchNormalization()(x)
x = layers.MaxPooling2D((3, 3), (2, 2), padding = 'same')(x)
x = layers.Dropout(0.3)(x)

x = layers.Conv2D(64, (3, 3), padding = 'same', activation = 'relu', kernel_initializer = 'he_uniform')(x)
x = layers.Conv2D(64, (3, 3), padding = 'same', activation = 'relu', kernel_initializer = 'he_uniform')(x)
x = layers.BatchNormalization()(x)
x = layers.MaxPooling2D((3, 3), (2, 2), padding = 'same')(x)
x = layers.Dropout(0.3)(x)

x = layers.Conv2D(64, (3, 3), padding = 'same', activation = 'relu', kernel_initializer = 'he_uniform')(x)
x = layers.Conv2D(64, (3, 3), padding = 'same', activation = 'relu', kernel_initializer = 'he_uniform')(x)
x = layers.BatchNormalization()(x)
x = layers.MaxPooling2D((3, 3), (2, 2), padding = 'same')(x)
x = layers.Dropout(0.3)(x)

x = layers.Conv2D(64, (3, 3), padding = 'same', activation = 'relu', kernel_initializer = 'he_uniform')(x)
x = layers.Conv2D(64, (3, 3), padding = 'same', activation = 'relu', kernel_initializer = 'he_uniform')(x)
x = layers.BatchNormalization()(x)
x = layers.MaxPooling2D((3, 3), (2, 2), padding = 'same')(x)
x = layers.Dropout(0.3)(x)

x = layers.Conv2D(128, (3, 3), padding = 'same', activation = 'relu', kernel_initializer = 'he_uniform')(x)
x = layers.Conv2D(128, (3, 3), padding = 'same', activation = 'relu', kernel_initializer = 'he_uniform')(x)
x = layers.BatchNormalization()(x)
x = layers.MaxPooling2D((3, 3), (2, 2), padding = 'same')(x)
x = layers.Dropout(0.3)(x)

x = layers.Flatten()(x)
x = layers.Dense(128, activation = 'relu')(x)
x = layers.BatchNormalization()(x)
x = layers.Dropout(0.3)(x)

output = layers.Dense(NUM_CLASS, activation = 'softmax')(x)
```

방법

- 성능 더 높이기 (5/7)
  - VGG like Model
    - Loss (0.6348)
    - Accuracy (85.87%)
    - Loss는 이전보다 높아졌지만 가장 높은 acc 달성
    - 하지만 약간의 overfitting이 보임





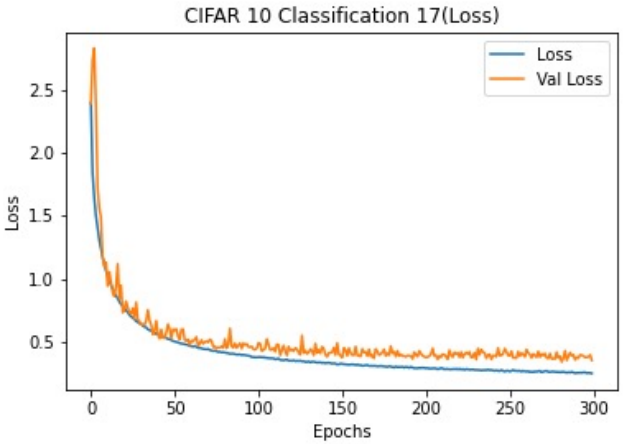
방법

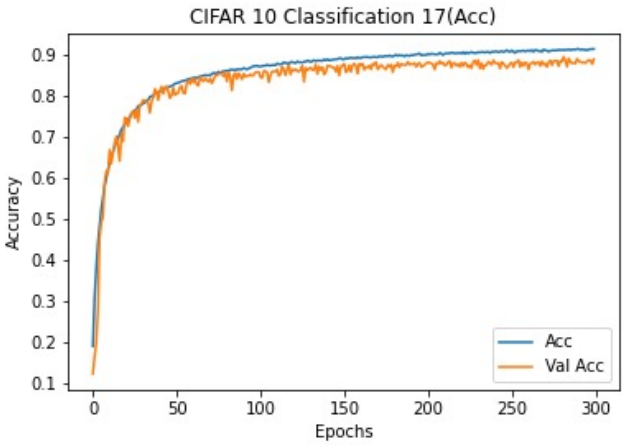
- 성능 더 높이기 (6/7)
  - VGG like model with data augmentation
    - VGG like model에 augment된 data로 학습을 진행

    - 모든 코드는 이전과 동일

방법

- 성능 더 높이기 (7/7)
  - VGG like model with data augmentation
    - Loss (0.3480)
    - Accuracy (89.02%)
    - 가장 낮은 loss를 달성하며 가장 높은, 90%에 가까운 acc 달성
    - 더불어 이전 VGG like model보다 overfitting이 해결된 모습





Contents

결론

- CIFAR 10 Classification
  - Sequential한 4 ~ 12개의 layer를 가진 CNN으로
  - 최저 63%에서 최고 89%의 성능을 가진 분류기를 학습시킴
  - 단순 CNN으로도 90%에 가까운 성능을 낼 수 있다는 것을 보임
  - 다양한 기법들을 직접 적용해보며 어떤 결과가 나오는지를 보였음
  - 그에 따른 성능 지표 변화와 다양한 조합을 해가며 어떤 결과를 보이는지를 보임

결론

- 중요 기법
  - Data normalization
    - Overfitting은 제거하지 못했지만, 성능 향상
  - Dropout
    - Overfitting 해결 + 약 5% 성능 향상
    - + Batch normalization + weight initialization을 한다면 총 8~9% 정도의 성능 향상
  - Data augmentation + layer 증가
    - 단순 data augmentation이나 layer 증가로는 주목할만한 성능 향상 X
    - 하지만 이 둘을 결합한다면 layer 증가의 overfitting 문제를 data augmentation이 잡아주며 성능 향상까지 불러오는 결과 초래

Contents

한계 및 추후 연구

- 한계
  - 89%까지 성능을 끌어올렸지만 90%를 넘기지 못해 쓸만하다고 생각 X
  - 사용한 기법들이 왜 성능을 높이는지를 설명하지 못함
    - 따로 따로는 엄청난 성능을 내지 못하였지만 결합하면 이전을 뛰어넘는 성능을 가지는지
      - e.g., 단순 Weight Init으로는 오히려 성능이 떨어졌지만
      Dropout + BatchNorm과 결합하면 어떤 조합보다 뛰어난 성능을 보임
  - 학습 시간 측면은 완전 배제하였음

한계 및 추후 연구

- 한계
  - DNN부터 복잡한 CNN (Inception V3, ResNet 등)의 성능은 보여주지 못하여 다른 네트워크보다 과연 뛰어난 것인지를 확인 불가

  - Hyper parameter에 따른 성능 비교를 하지 못하였음

한계 및 추후 연구

- 추후 연구
  - 성능 향상
    - 90%를 뛰어 넘어 실제 사용할만한 성능을 가지도록 네트워크 구현
    - 더 적은 Epoch로 더 좋은 성능을 낼 수 있도록 좋은 네트워크 고안
    - 훨씬 좋은 성능이 증명된 다른 네트워크 구조 또한 사용
  - 상관 관계 연구
    - 더 다양한 조합으로 어떤 결과를 낼 수 있는지 연구

감사합니다!