과목 명: 시스템프로그래밍

담당 교수 명: 소 정 민

<<Assignment 5>>

**서강대학교 컴퓨터공학과**

**[20151529]**

**[김예찬]**

목 차

**1. 프로그램 개요** 3

**2. 프로그램 설명** 4

2.1 프로그램 흐름도 4

**3. 모듈 정의 및 코드 분석** 5

3.1 모듈 이름 : Paramter tuning part 5

3.1.1 기능 5

3.1.2 사용 변수 5

3.2 모듈 이름: Plot image 6

3.2.1 기능 6

3.2.2 사용 변수 6

3.3 모듈이름: Dataset load 6

3.3.1 기능 6

3.3.2 사용변수 6

3.4 모듈이름: Data Preprocessing 1 (Reshaping the data) 7

3.4.1 기능 7

3.5 모듈이름: Data Preprocessing 2 (Applying one hot encoding) 8

3.5.1 기능 8

3.6 모듈이름: Data Preprocessing 1 (Reshaping the data) 8

3.6.1 기능

3.7 모듈이름: Show data 8

3.7.1 기능

3.8 모듈이름: DNN model creation (Add layer) 8

3.8.1 기능 및 퍼포먼스 개선 노력 9

3.9 모듈이름: Model Visualization 16

3.9.1 기능

3.10 모듈이름: Optimizer 16

3.10.1 기능 및 퍼포먼스 개선 노력 16

3.11 모듈이름: Model Compilation 19

3.11.1 기능 19

3.12 모듈이름: Model Training 20

3.12.1 기능 20

3.13 모듈이름: Model evaluation 21

3.13.1 기능 및 비교 21

3.14 모듈이름: Model prediction 22

3.14.1 기능 23

**4. 코드** 24

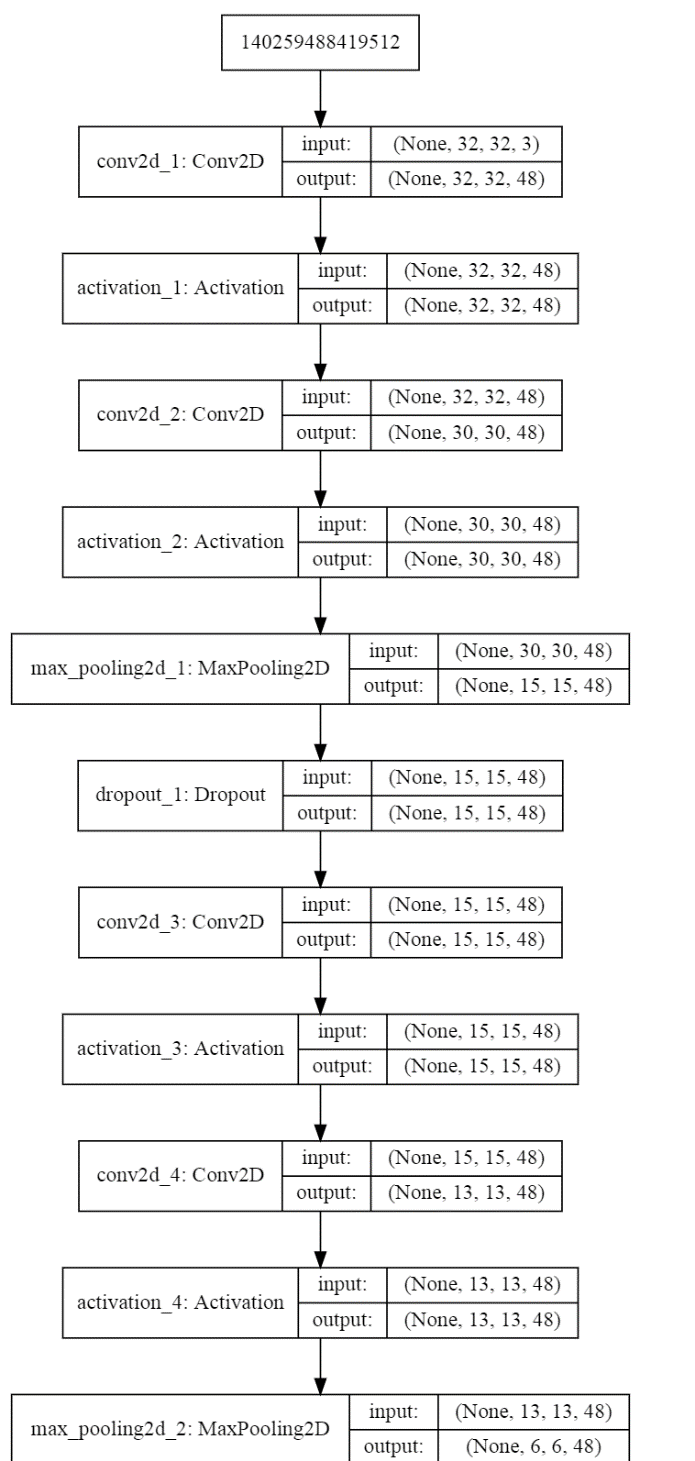
**5. Appendix - Model training record** 28

# 프로그램 개요

Machine Learning을 이해하기 위해 TensorFlow 기반의 Keras 모듈을 이용해 DNN 을 만들어 CIFAR-10 data를 바탕으로 image recognition 모델을 만든다. 적절한 하이퍼 파라미터 설정과 layer 설정, Optimizer 선택에 따라 원하는 수준(75% 이상)의 accuracy를 획득한다.

# 프로그램 설명

## 프로그램 흐름도



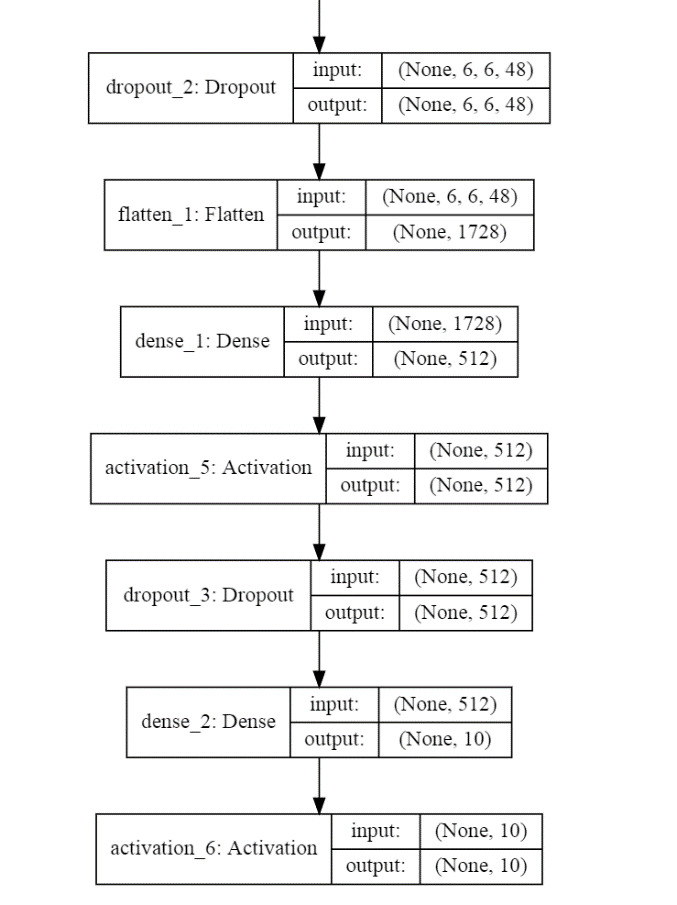


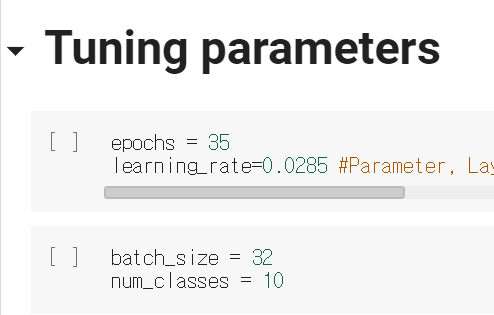
그림 1> DNN model layer

CIFAR-10 data set을 input으로 하는 DNN model에 layer을 쌓아 놓은 것이다. 필터의 크기 3\*3, 개수 48개인 Convolution layer을 이용해, input과 크기가 동일한 output을 필터의 개수 48개만큼, 지역적인 특징들을 뽑아내서 만든다. 이를 Vanishing Gradient problem이 있는 sigmoid function을 방지하기 위해 ReLU activation function을 거친 뒤, 다시 previous convolution layer과 동일한 layer을 거친다. 이렇게 2번 필터링 된 input을 Max Pooling을 이용해 주요한 특징만을 뽑아낸 뒤, Dropout layer을 거쳐 Overfitting 문제와, 특정 뉴런에 특징이 집중되지 않도록 처리해준다.

여기까지의 과정을 1번 더 반복한 뒤, Output layer로의 출력을 위해, 2차원 모델을 1차원 모델로 변환해주는 Flatten layer을 거친다. 다시 이를 512개의 뉴런과 fully connected 된 Dense layer를 ReLU activation function을 거쳐 마지막으로 Dropout을 해준다. Image의 class의 총 개수, 다시 말해 출력 뉴런은 10개가 되어야 하므로, 마지막 hidden layer의 뉴런과 최종 10개의 뉴런을 fully connect해주는 layer을 쌓고, 이미지의 상대적인 확률을 평가하는 softmax activation record를 이용해 Image recognition 모델을 완성한다.

# 모듈 정의 및 코드 분석

## 모듈 이름 : Parameter tuning part



### 기능

Machine learning의 환경설정 hyper parameter인 epoch, learning rate와 batch\_size와 최종 image recognition class의 개수인 num\_classes 변수를 설정한다. 적절한 parameter 초기값 설정은 굉장히 중요한데, 이는 Overfitting 문제 및 local minimum에서 탈출하지 못하는 문제를 발생시킬 수도 있고, 결과적으로 적절히 train된 Model을 생성해내지 못할 수도 있다.

### 사용 변수

1. Epoch : 한 data set을 training할 횟수로, model에 외부적으로 이용되는 hyper parameter이다. 이번 프로젝트에서 epoch의 크기는 learning rate에 맞추어 적절하게 적게 설정하였는데, epoch을 늘리면 training sample에 대한 accuracy가 높아지지만 실제 test set에서 accuracy가 정체되는 overfitting이 발생하였다. Overfitting을 해결하기 위해 Dropout 방식, Convolutional layer의 뉴런의 개수를 줄이는 방법 등을 이용하였고 자세한 설명은 해당 모듈 부분 참조.
2. Learning rate : Machine Learning의 Learning rate로, weight, bias update에 이용되는 수치이다. Learning rate가 너무 클 경우, learning 속도는 빠르지만 weight의 최적값을 찾기가 힘들고, 반대로 너무 작을 경우 최적값을 찾을 확률은 높지만, learning 속도가 너무 느리기에 적절한 값 설정이 중요한 변수, hyper parameter이다.

이번 프로젝트에서 Optimizer가 SGD일 때 0.03보다 큰 값으로 설정하였을 때 Learning speed는 빨랐지만, model의 accuracy가 더 이상 향상되지 않아 overfitting이 발생하였고, 0.026 근사치로 설정하였을 경우, Learning speed가 너무 느렸기에 그 중간인 0.285로 설정하게 되었다.

Optimizer로 Adam을 선택했을 때 권장 default값인 0.001로 설정하였다.

1. batch\_size : 한 번에 data input을 몇 개나 넣을지 정하는 변수로, 하나의 data씩 모델에 넣어 cost function을 이용해 weight, bias를 조정하기엔 시간적 부담이 크기에, 동시에 여러 input을 넣어 weight와 bias를 조정하기 위한 방법론을 지원하는 변수이다.
2. num\_classes : Image recognition의 최종 결과 class들의 개수이다.

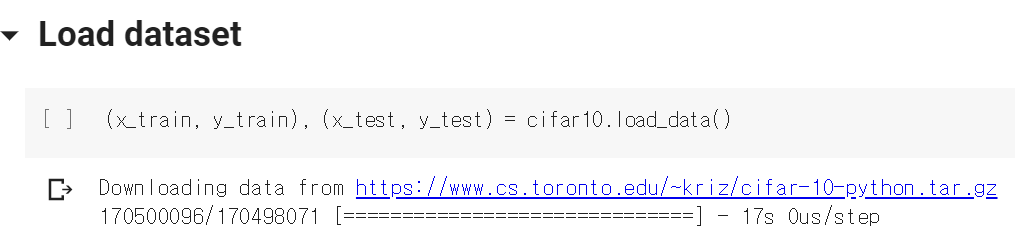
## 모듈 이름: Plot image



### 기능

CIFAR-10의 image data를 load한 것을 5\*5매트릭스 형태로 이미지와, 이 이미지가 속한 class들의 data label값을 출력해준다.

## 모듈 이름: Dataset load



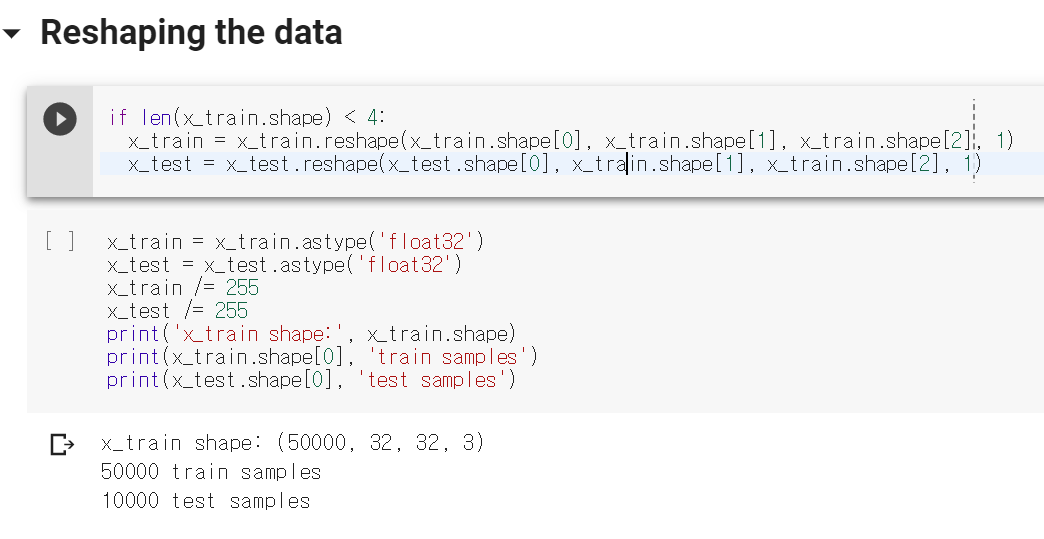
### 기능

Model Training을 위한 CIFAR-10 image data set과 이 data set의 label 값, 그리고 Model test용 input과 label 값을 load하는 모듈이다. 50,000개의 training sample과 10,000개의 test sample이 있다.

### 사용변수

1. x\_train, y\_train : Training sample의 input, ouput
2. x\_test, y\_test : Test sample의 input, output

## 모듈이름: Data Preprocessing 1 (Reshaping the data)



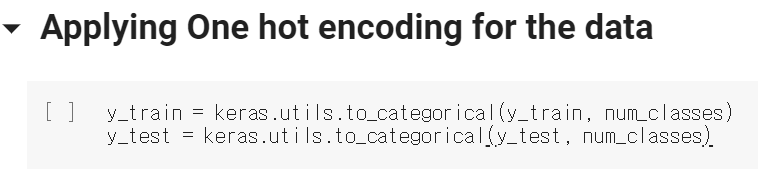
### 기능

Model training 이전 Data preprocessing 과정을 담당하는 모듈이다.

Training sample과 test sample의 shape property의 길이가 4보다 작을 경우 (= (data 개수, 이미지 픽셀 행, 이미지 픽셀 열, color))을 맞춰 연산을 가능하게 해주기 위해 조건문을 설정하였다.

이 후, data처리를 위해 Feature Scaling이 필요하다. x\_train의 각 element는 0~255사이의 값을 가지고 있는데, Overfitting 과 Cost function의 빠른 수렴을 위해 0~1로 값을 Scaling해준다. 또한 나눗셈 연산이 들어가 float형으로 변환도 해주어야한다. 현재 모델에서는 255로만 나누어도 Feature Scaling이 가능하다.

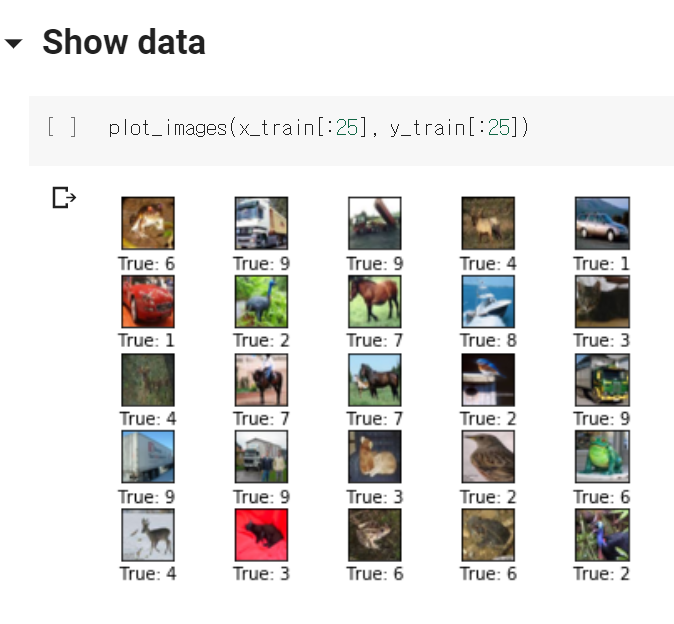
## 모듈 이름: Data Preprocessing 2 (Applying one hot encoding)



#### 기능

Data preprocessing을 위해, Label의 categorical값을 one-hot 형태로 변환해 0, 1로 label data로 재설정해준다.

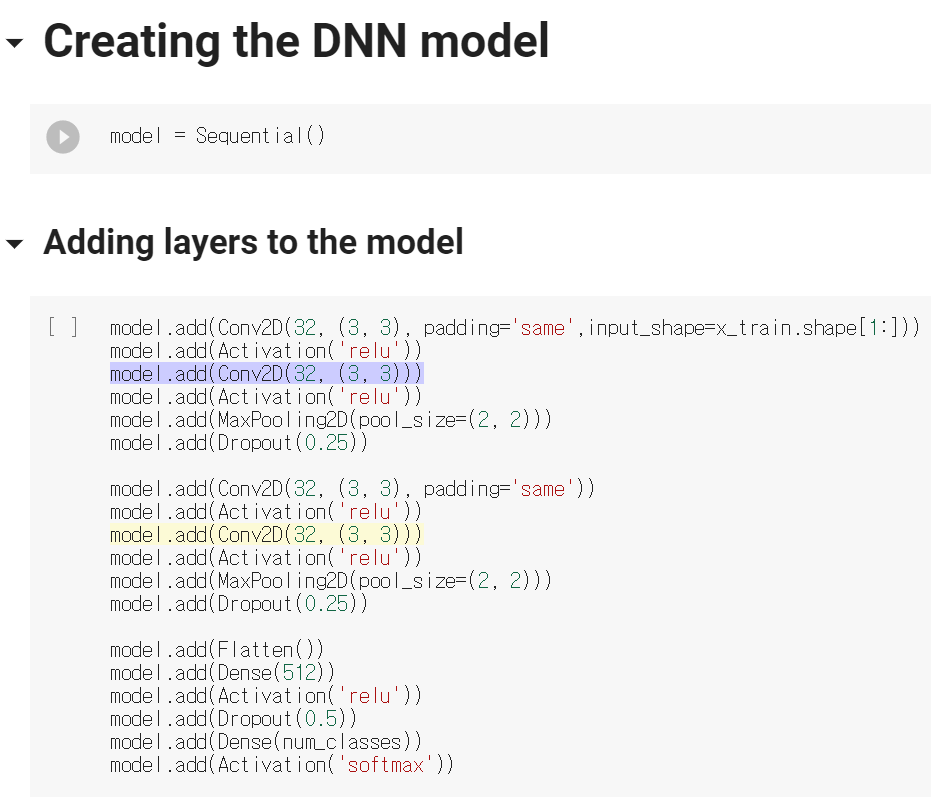
## 모듈 이름: Show data



### 기능

Training sample data의 이미지와, Label 값을 출력해준다 (plot\_image 모듈 이용).

## 모듈 이름: DNN model creation (Add layer)



### 기능 및 퍼포먼스 개선 노력

Image classification Model을 만드는 부분으로 이번 프로젝트 Machine Learning의 핵심부분이다.

1. Sequentual() : 메소드를 이용하여 model에 대해 layer을 순차적으로 쌓을 수 있는 공간을 만들었다고 간주하면 된다.
2. .add() 메소드를 이용하여 layer을 쌓아 갈 것이다.

기존 Multi Layer perceptron 모델에서는 3가지 문제점 (가까운 픽셀 또는 먼 픽셀 간 무의미한 차이, Fully connected된 Dense layer라 Scalibility의 문제, 조금이라도 다른 이미지일 경우 classification이 힘든 문제)가 발생하여 이를 해결하기 위해 CNN(Convolutional Neural Network)가 대두되었다. CNN에는 전통 MLP 모델의 문제점을 해결하기 위해, Convolution layer, pooling layer을 추가했는데 이 둘의 대표적인 특징은 data에서 Featrue을 추출한다는 점이다. 이를 통해 Flexible한 model을 형성할 수 가 있는 것이다.

* **Layer Stack**

2-1) 첫 번째 Layer



Convolutional layer을 모델에 추가할 것인데, Convolution layer란, 가중치가 동일한(shared weight) 필터를 Local하게 적용하여 Feature mapping 한다.

Conv2D의 parameter의 48는 필터의 개수, 즉 Mapping된 Feature, 뉴런의 개수이다.

(3,3)은 필터의 매트릭스 shape, "padding”은 경계를 처리하는 방법으로, value값으로 'same’이 들어가면 출력 이미지 사이즈가 입력 이미지 사이즈와 동일할 수 있게 필요하다면 virtual한 빈 data를 추가하여 필터를 적용한다.

이번 프로젝트에서 뉴런의 개수를 48개로 제한하였는데, 기존 Keras document에서는 64개였으나 Overfitting이 지속적으로 발생함에 따라 parameter 수를 줄이기 위해 뉴런의 개수를 줄였다.

모델에 layer을 쌓을 때, 첫 layer의 parameter로 들어올 input shape을 모델에 알려주어야 한다.

2-2) 첫 번째 Activation function



Layer을 쌓았다면, weight와 bias를 input을 이용해 구성한 Activation function으로 output을 도출해야한다. ‘ReLU’라는 이름의 activation function을 이용할 것인데, 이는 기존 sigmoid function에서 발생하는 문제를 해결하기 위해 나타난 대안이다.

Sigmoid function은 뉴런이 fire되는 유무에 따라 0또는 1이 되는 step function을 미분하기위해 이와 유사한 값을 가지는 function이다. 그런데 Sigmoid function을 activation function으로 이용할 경우, Vanishing Gradient Problem이 발생할 수 있다. 이는 Sigmoid function의 미분값으로 gradient가 결정되는데, 미분값의 최대값(1/4)으로 반복될 경우, gradient가 0에 가까워져 weight와 bias의 변화가 잘 발생하지 않아 Learning의 속도가 급격히 하락해 Accuracy역시 증가하지 않을 수 있다.

이를 해결하기 위해 ReLU(Rectifier Linear Units) function을 대안으로 이용한 것이다.

2-3) 두 번째 Layer, Activation function



위 2-1), 2-2)를 다시 반복하여 Feature을 더 추출한다.

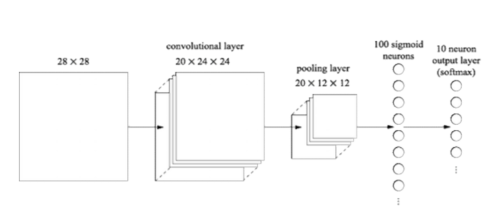
2-4) 세 번째 Layer



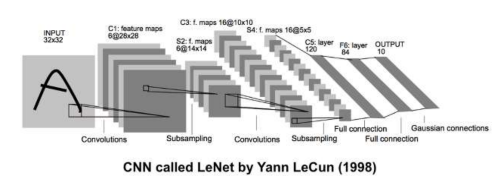
이번엔 Max Pooling layer을 모델에 추가할 것이다. Pool size는 (2,2)로 이는 input shape에 대한 비율이다. Max Pooling은 특정 local area에서 주요 값만을 뽑아 크기가 작은 출력을 만드는 것으로, Local 내 사소한 변화가 영향을 미치지 않고 Feature에 집중하는 것을 가능하게 해준다. 이를 통해 다시 모델의 Flexibility를 높일 수 있는 것이다.

이렇게 2-1) ~ 2-4) 과정을 거치는 것을 간단한 CNN architecture라고 할 수 있다.

퍼포먼스를 더 향상시키는 방법들이 있는데, 첫 째, Fully-connected layer(Dense layer)를 추가해주거나(이후 코드에 추가 예정),



둘 째, Convolution layer을 추가하거나



셋 째, Activation function으로 sigmoid 대신 ReLU를 사용하거나(이미 코드에서 적용)

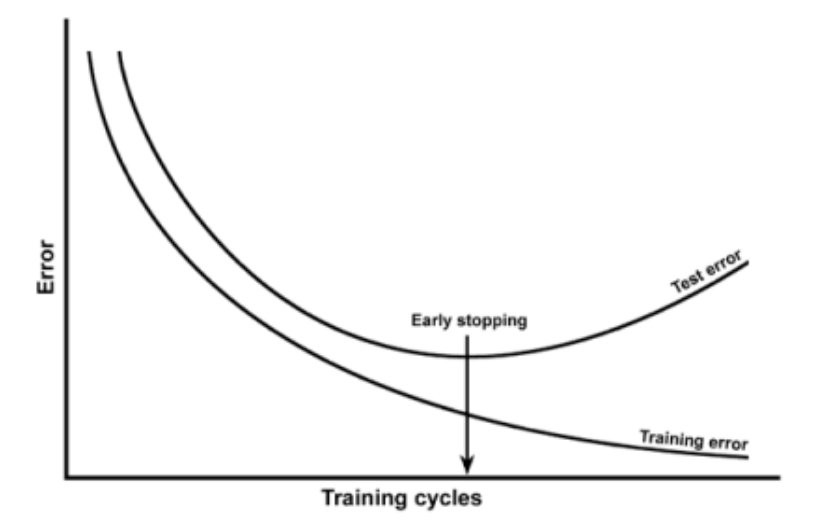
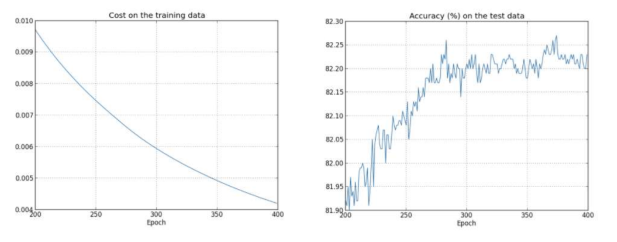
넷 째, Data augmentation, Dropout, Ensemble 하는 방법들이 존재한다.

2-5) Overfitting 방지를 위한 Dropout

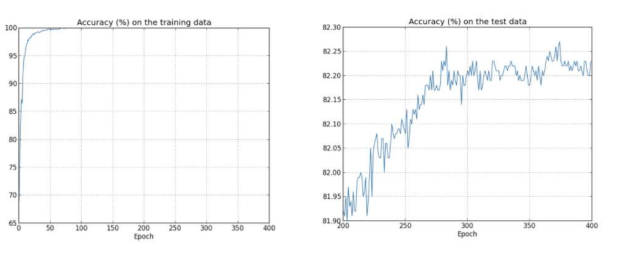


Dropout 이란 Overfitting 문제를 해결하기 위해 모델의 뉴런을 일시적으로 지우고, 매 batch sample마다 뉴런의 집합을 새로 골라 training을 진행하는 방법이다.

이에 앞서, **Overfitting이란**, 특정 data set에 너무 학습이 잘되어 추가적인 data나 machine learning의 input으로 적절한 data가 들어가도 제대로 fitting을 하지 못하는 현상을 말한다. 학습 데이터에 대해서는 accuracy가 높아지지만, 실제 데이터에 대해서는 accuracy가 증가하지 않는다.

<Overfitting graph : Cost는 감소하지만 Accuracy는 정체 – 제대로 학습이 되지 않고 있다>



<Overfitting graph 2 : Training set의 accuracy는 매우 높지만, Test set의 accuracy는 낮다 – 학습한 것이 제대로 된 것(일반적인 것)이 아니다>

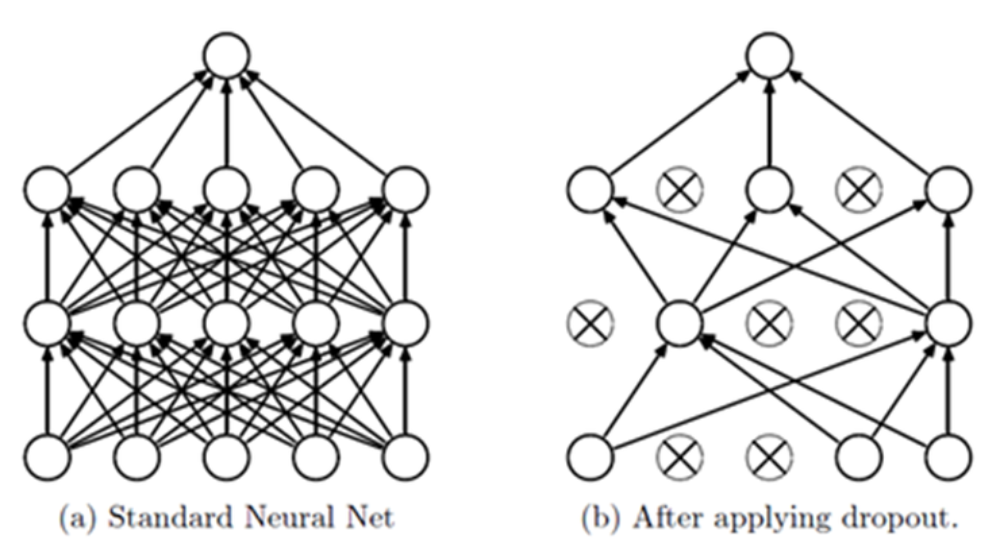
이러한 Overfitting을 해결하기 위한 방법으로는

1. Data augmentation을 이용한 Training data set의 확장
2. Dropout을 이용해 특정 뉴런에 의존성이 커지는 것을 방지해 다양한 뉴런의 robust feature을 학습
3. 초기 parameter 값을 달리 train한 모델들로 투표한 결과를 선택하는 Ensemble 방법

등이 존재한다.

그런데 Data augmentation과 같이 결국 데이터의 양을 늘리는 방법은 Learning cost부담이 있기에, - "Occam's Razor” (사고 절약의 원리) 같은 현상을 설명하는 2개의 주장이 있다면 간단한 쪽을 선택하는 – 방법론에 따라 등장한 Normalization, Dropout 등 다양한 해결책들 중

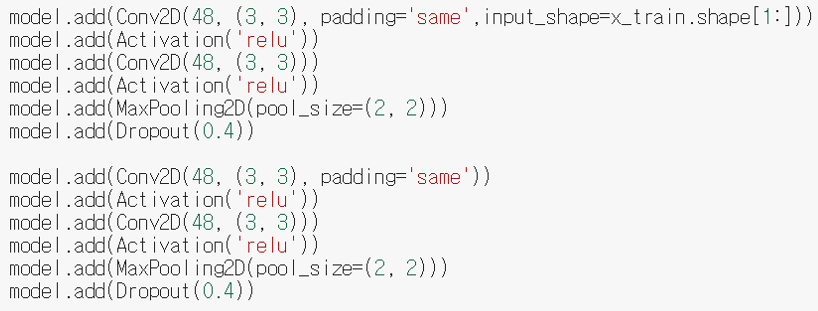
이번 프로젝트에서는 Dropout 방법을 이용하였다.



Dropout을 이용해 무작위로 줄어든 다양한 Net에서 각각 train이 되면 결과적으로 평균 효과를 얻을 수 있다. 또한, 특정 뉴런의 bias나 weight가 큰 값을 가지면 그 영향이 커져 다른 뉴런들의 학습 속도가 느려지거나 제대로 학습이 안되는데, 특정 뉴런의 영향을 없애 co-adapt되는 것을 방지, 특정한 data set에 영향을 받지 않는 robust한 net을 구성할 수 있다.

Dropout의 parameter로 들어가는 실수 값은 Drop할 뉴런의 비율을 의미한다. 일반적으로 Dropout의 비율은 0.25 내외로 유지되는데, 뉴런의 개수가 32개일 때 4분의 1을 drop out한 것과, 뉴런의 개수가 48개일 때 5분의 2를 dropout한 것에서 accuracy 차이가 발생하여 일반적인 값보다 큰 0.4를 drop한다.

2-6) 이전 Layer stack과 동일하게 stack



이렇게 2-1) ~ 2-5)까지의 과정을 한 번 더 반복해, 위에서 언급한 Convolution Neural Network의 퍼포먼스를 향상시켰다.

2-7) Fully-connected layer연결을 위한 Data processing



Flatten Layer는 CNN architecture model을 거친 뒤 추출된 주요한 특징들을 연산하기 위해 기존 2차원 data에서 1차원 data로 convert해주기 위한 것이다. 이렇게 1차원 data로 변환해줌으로 써 Fully connected layer (Dense layer)의 input으로 적용될 수 있다.

Flatten은 이전 layer의 output을 자동으로 input으로 설정하고, output 역시 input에 따라 자동으로 설정된다. 이전의 layer의 output은 6 \* 6 의 shape으로 32 개 였기에, 이를 input으로 받은 Flatten layer은 6\*6\*32=1152 개의 data로 출력해준다.

2-8) Fully-connected layer 쌓기



이전 Flatten layer에서 1차원으로 변환된 data sample은 Fully connected layer (Dense layer)의 input으로 들어간다.

Dense layer의 parameter는 출력 뉴런의 개수로 여기서는 512개로, 또한 다음 코드에서 Activation function을 ReLU로 설정해 Layer에서 처리를 해준다.

앞서 추출된 주요 Feature들이 Dense layer와 연결되어 CNN architecture의 퍼포먼스를 향상되는 특징도 존재한다.

2-9) Overfitting 방지를 위한 Dropout



이 후, Overfitting을 방지하기 위해 다시 한 번 더 Dropout layer을 추가하는데, 이번에는 뉴런의 반을 Drop 한다.

2-10) Output layer

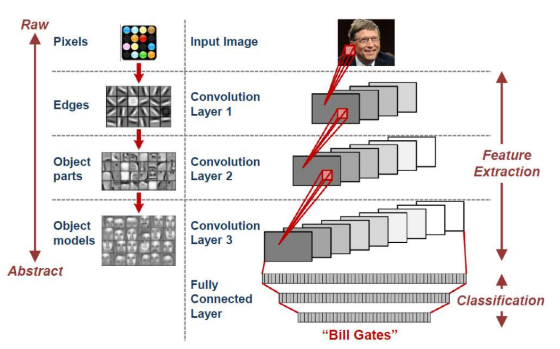


모든 Hidden layer들을 거쳐 이제 마지막으로 output layer로 Input data set을 Image classify하기 위해 앞서 설정해주었던 num\_classes (클래스의 개수, 10개) 만큼 뉴런을 만들어 fully-connect 한다.

마지막 output layer에서 사용되는 activation record는 ReLU가 아닌, softmax이다.

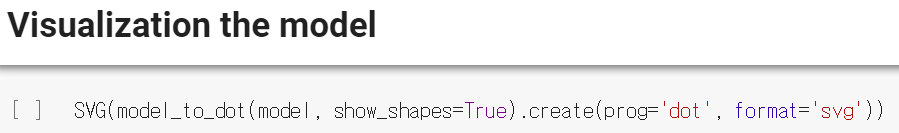
Softmax는 output layer에 자주 사용되는 activation function으로, 모든 activation 결과값의 합이 1이고, 각 결과값들은 이 합으로 나뉘어 상대적인 확률 분배 및 가치를 지니게 된다.

이렇게 모든 과정을 거쳐 layer들을 쌓아 만든 CNN architecture model은



위 그림과 같이 Raw한 low-level feature에서부터 추상적인 feature들을 추출해 내어, Fully connected layer을 이용해 image classification을 수행한다.

## 모듈 이름: Model Visualization



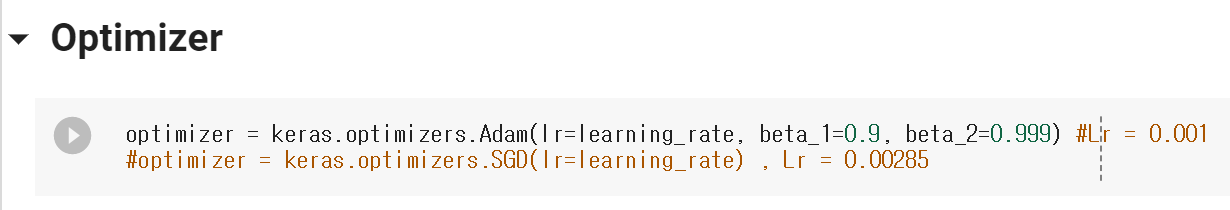
### 기능

Layer들을 쌓아 만든 model을 input과 output, 그리고 자세한 정보를 출력해주는 모듈이다. 아래로 순차적으로 내려갈수록 다음 layer들을 뜻한다.

### 사용변수

1. x\_train, y\_train : Training sample의 input, ouput
2. x\_test, y\_test : Test sample의 input, output

## 모듈 이름: Optimizer



### 기능 및 퍼포먼스 개선 노력

Optimizer란 Machine Learning에서 적절한parameter(weight, bias 등)를 찾아 cost function을 최적화시키는 과정을 진행한다. Cost function이 0에 가까워 지는 것이 목표로, optimization으로 이용할 수 있는 방법으로는 SGD (Stochastic gradient descent optimizer), RMSprop, Momentum, Adagrad, Adadelta, 그리고 Adam 등이 있다.

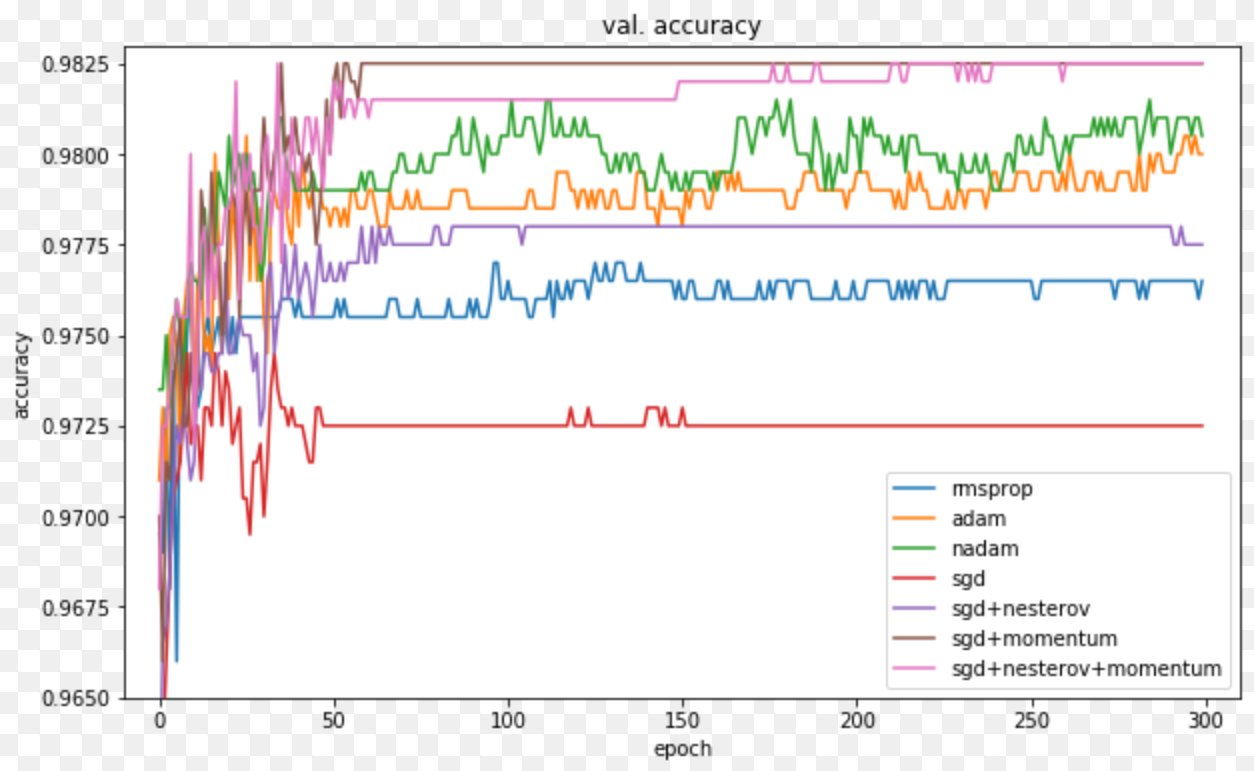
* **Optimizer의 종류 및 특징**

1. **SGD**: 기존 Gradient descent 방식은 data set이 많아지면 batch 사이즈가 커져 training 시간이 오래 걸린다. 또한 random sample이 포함된 data set에 중복 데이터가 포함되어 중복의 가능성도 존재한다. 이러한 문제점을 작은 batch size – 적은 계산 –으로 근사한 결과값을 얻는 Stochastic gradient descent 방식으로 해결할 수 있다. Data set에서 sample을 random하게 선택해 평균값을 추정하는 방법론을 확장해, 반복 당 훨씬 크기가 작은 batch(크기가 1)를 사용하고, 비록 노이즈가 심할 수 있지만, 충분한 반복을 통해 효과를 얻을 수 있다.
2. **RMSprop**는 gradient descent의 퍼포먼스를 향상시킬 수 있는 또다른 방법으로, Root Mean Squared Propagation의 약자이다. Exponentially weighted average를 이용하는 방법으로, 모델의 각 parameter에 개별 learning rate 조정이 가능하다. RMSprop은 gradient가 큰 parameter에서 learning rate가 감소하는 속도가 빠르고, gradient가 작은 부분에서는 속도가 느리다. 이를 통해, 노이즈와 oscillation(진동)을 방지할 수 있으나, 결적으로 learning rate가 너무 작아져 model이 완전히 멈출 수 있는 단점도 존재한다.

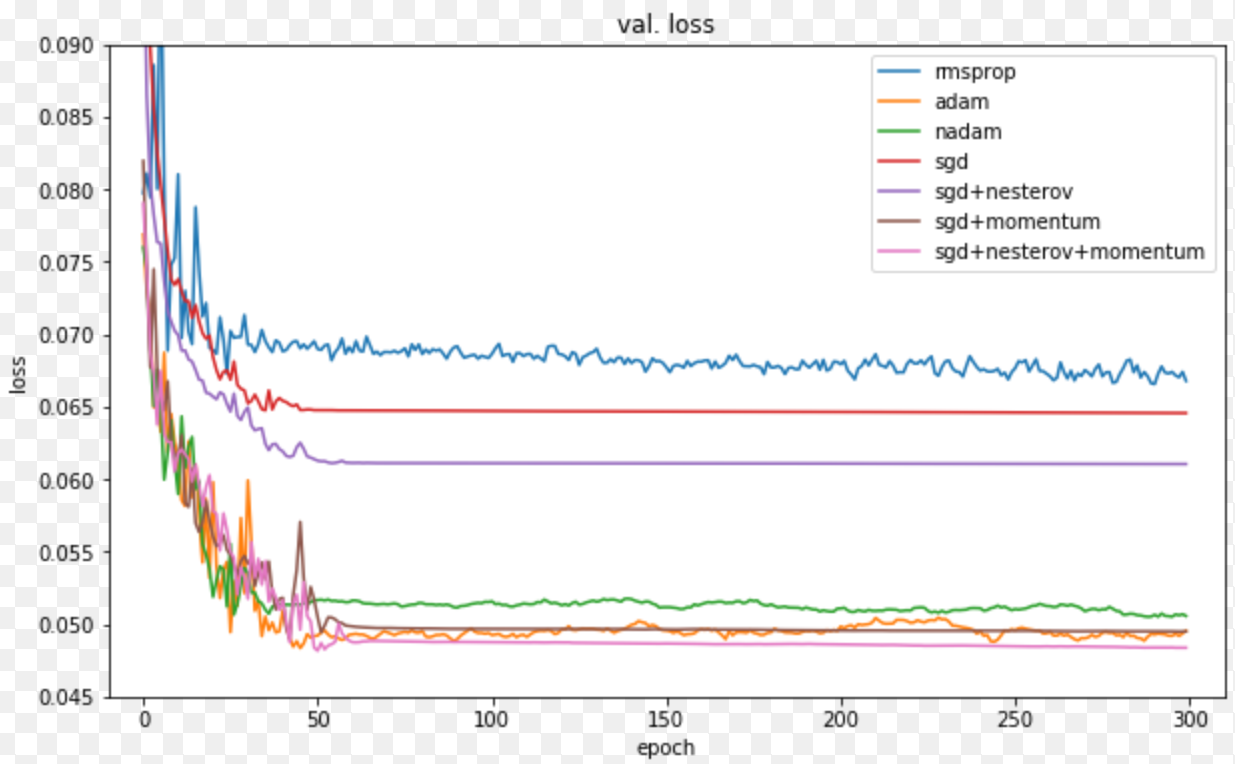
RMSprop은 Non-sttionary한 data를 이용해 train할 때 많이 사용한다.

1. **Momentum**은 SGD에서 계산된 gradient의 한 step 전 gradient를 일정한 비율만큼 반영하여 사용함으로써, 새로운 gradient에 관성 모멘트와 같은 효과를 주어 local minimum을 빨리 빠져나올 수 있는 방법이다.
2. **Adam**(Adaptive Moment Estimation)은 RMSprop의 장점과 Momentum optimization의 아이디어를 결합한 방법이다. 이는 빠르고 효과적인 optimization을 가능하게 해주는데, 효과가 증가함에 따라 계산의 복잡성도 커진다.
3. **Adagard** 는 learning rate를 normalization하여 서서히 감소시키는 방법이다. Adagard는 training동안 parameter가 얼마나 자주 update되는지에 상대적으로 적용되어, parmeter가 더 자주 update될수록, learning rate가 더 작아진다.
4. **Adadelta**는 Rmsprop과 Adagrad를 보정한 방법으로, Adagard의 더욱 robust한 확장판이다. Adadelta는 update가 자주 발생하여도 learning rate를 유지하고, 초기 learning rate를 따로 설정하지 않아도 된다.

이 외에도 다양한 Optimizer가 존재하고 각각의 특징들이 존재해 상황에 따라 적절한 것을 이용할 수 있다. 이번 프로젝트에서는 Data set의 크기가 크다고 고려하여 적은 batch size로도 일반 Gradient descent 모델과 근사한 결과를 도출해 낼 수 있는 SGD를 이용하였다. 그러나 SGD에서 더 나아가 Optimization의 방법은 지속적으로 발전해왔고, 비교적 최신인 RMSprop 또는 Adam을 이용하였다면, 더욱 빠르고 효과적인 Optimization이 가능했을 것이다.



<Optimizer에 따른 accuracy변화 graph>

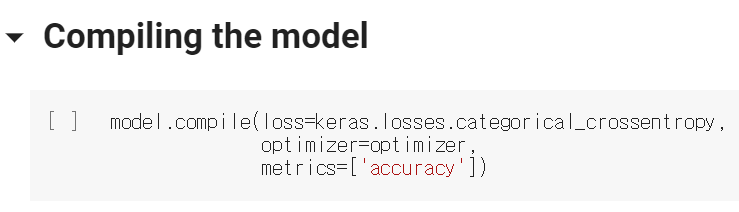


<Optimizer에 따른 loss의 변화 graph>

위의 그래프들을 참조한다면, Accuracy : SGD < RMSprop < Adam, Loss : Adam < SGD < RMSprop 임을 알 수 있다. 이에 따라 근 2년 전부터 Adam이 유행하는 이유 – 퍼포먼스 – 를 확인할 수 있고, SGD를 개선한 다면 Adam에 근접한 퍼포먼스를 보일 수 있다는 것도 알 수 있다.

따라서 이번 프로젝트에서는 SGD와 Adam을 이용하여 동일 epoch과 batch size 조건 하에 두 Optimizer의 퍼포먼스를 비교해 볼 것이다. (Adam의 Learning rate가 SGD의 learning rate보다 훨씬 작고, Tutorial 레벨의 모듈 이용에 따라 non-specific, rough한 comparison)

## 모듈 이름: Model Compilation

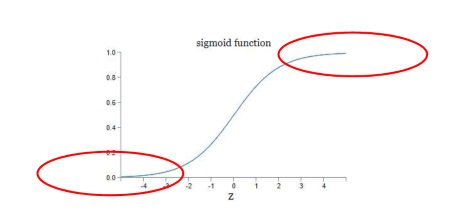
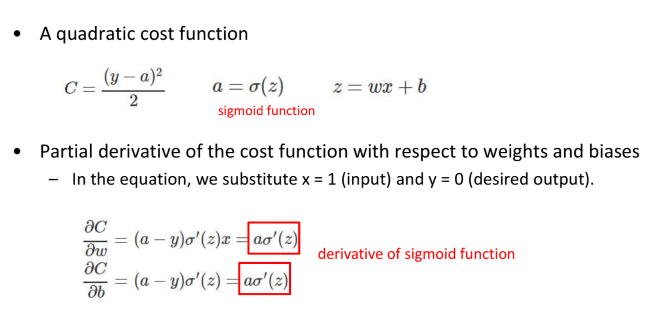


### 기능

Model을 train하기 전에 optimizer, loss function (or cost function), metrics 등의 환경 설정을 compile하는 모듈이다.

Loss function은 single training의 error을 나타낸다면 cost function은 전체 training 간 loss function의 평균으로 loss function과 synonym처럼 쓰이기도 한다. Loss function으로 가능한 방법은 Mean Squared error, Mean absolute error, hinge, categoriacal crossentropy등이 있다. 이번 프로젝트에서는 categorical crossentropy를 이용할 것인데, 이는 이전에 quadratic cost function을 이용해 발생한 문제점을 해결하기 위해서이다.

흔히 사용되는 MSE, Quadratic cost function을 이용할 경우, 초기 parameter의 설정값에 따라 learning이 굉장히 느려질 수 있다. 이는 cost function의 미분값이 sigmoid function의 미분값을 이용하기 때문인데(Activation function으로 sigmoid function을 사용할 경우 cost function과 특성이 결합), sigmoid function의 미분값은 input의 크기가 0 또는 1에 가까울수록 작아져 cost function 의 미분값(변화량) 역시 작아지기 때문이다.



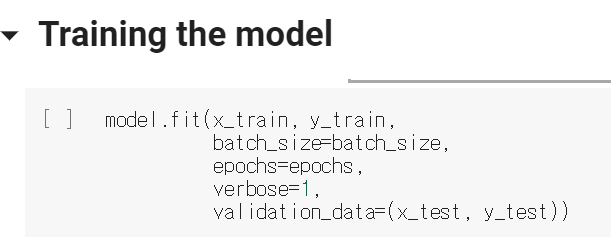
<cost function과 미분 식> <sigmoid function의 미분graph>

이렇게 Learning이 느려지는 것을 방지하기 위해 Label output과 model output의 차이에 기반해서 learning speed를 결정하는 "Cross-Entropy” cost function을 설정하였다. 이 cost function의 변화량은 weight를 바탕으로 이루어지기에, slow learning을 방지할 수 있는 것이다.

Learning speed 측면에서는 Cross Entropy 방식이 훨씬 빠르나, Learning outcome은 MSE나 Cross Entropy나 큰 차이가 없다.

학습 결과에 차이가 없지만 속도에서의 차이가 분명하기에 이번 프로젝트에서는 Cross Entropy 방식을 이용한다.

## 모듈 이름: Model Training

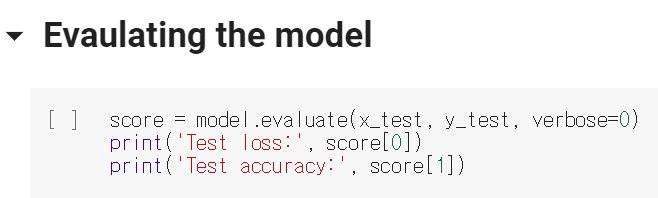


### 기능

.fit() 메소드는 주어진 epoch 횟수 만큼 model을 train한다. 간단히 parameter에 대한 설명을 하자면, Training sample data와 이에 대응하는 label data, batch크기, data set 반복 횟수(epoch), progress bar가 나오는 지 Verbosity mode 설정, epoch 끝날 때마다 loss와 metrics를 평가하기 위한 test sample data로 train 되진 않는 것들이 있다.

이전에 만들어 놓은 CNN model 과 설정한 optimizer를 이용하여 model train을 진행한다.

## 모듈 이름: Model evaluation



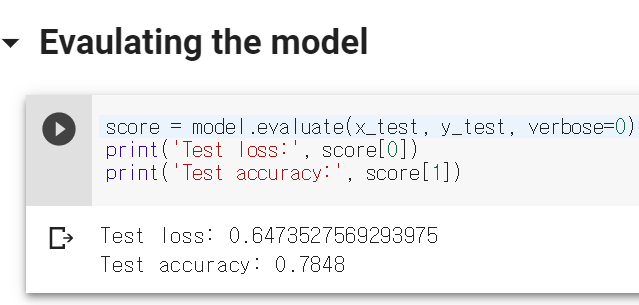
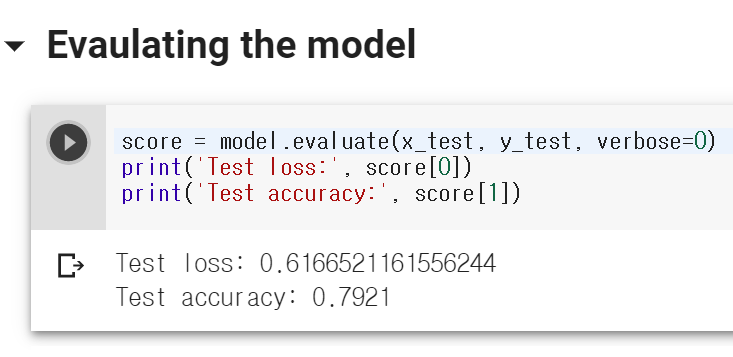
### 기능 및 비교

.evaluate()는 Model의 loss와 metrics 값을 return해주는 모듈로, Training이 완료된 모델의 퍼포먼스를 평가하기 위해 이용한다. 이번 프로젝트에서는 loss와 accuracy가 평가척도이다.

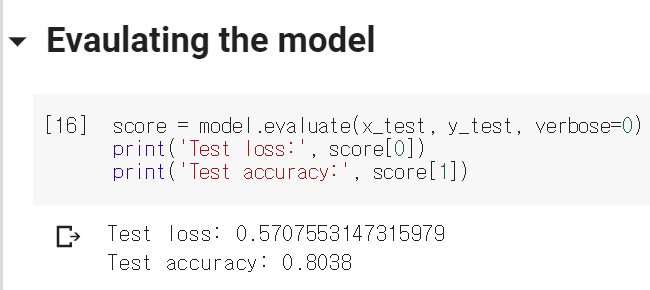
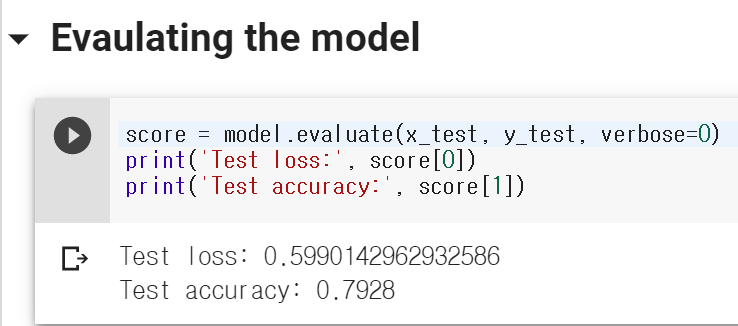
위의 Optimizer module 부분에서 언급한 것처럼, 두 Optimizer Adam과 SGD의 퍼포먼스를 비교해 더 나은 accuracy를 얻을 수 있는 것을 선택하고자 하였다.

비교에 앞서 설정한 조건은, Epoch : 35, batch size : 32로 동일하고, Learning rate는 SGD는 0.0285, Adam은 0.001로 설정하였다.

Model train을 완료한 뒤 loss와 accuracy를 비교해보았을 때, SGD와 Adam의 accuracy 차이는 유의미한 수준이 아니었고, loss는 Adam이 조금 더 발생한 것으로 보였다.



<Adam Train 1> <Adam Train 2>

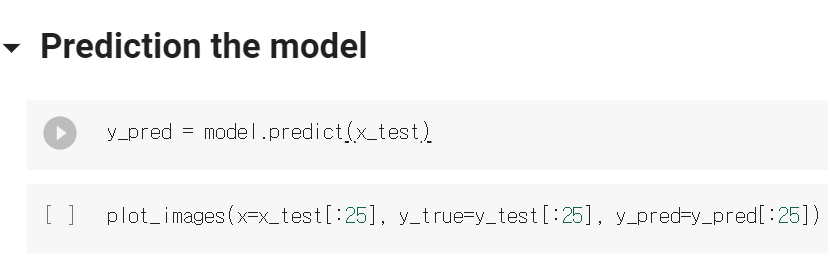


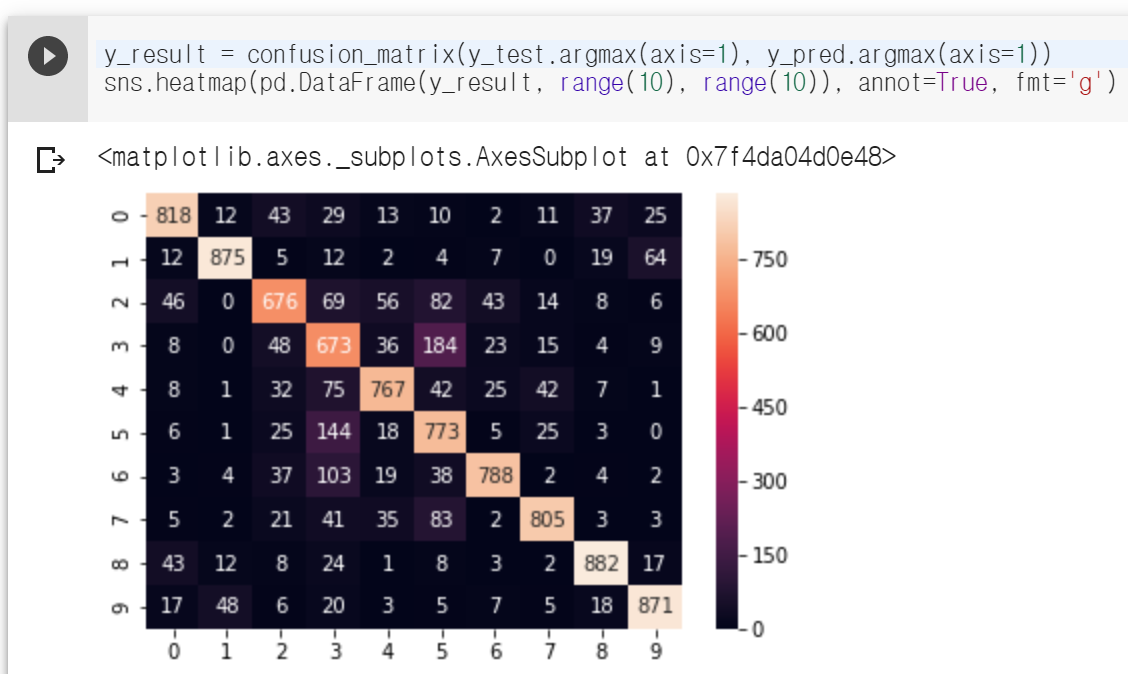
<SGD Train 1> <SGD Train 2>

조사한 위의 Optimizer module 부분에 따르면 Adam이 일반 SGD보다 퍼포먼스가 더 좋다는데, 왜 이번 프로젝트 조건 상에서는 둘의 차이가 거의 없는 것인지 의문점이 들었다. 보고서 뒤에 첨부한 Appendix - Model training record를 보면, SGD는 목표 Accuracy 75%에 도달하는데 20 epoch 정도가 걸렸고, Adam은 평균적으로 15 epoch 정도가 걸렸다. Learning rate가 SGD보다 훨씬 적음에도 불구하고 Adam이 더 적은 epoch 횟수에서도 목표 Accuracy를 빨리 도달한 것이다. 결과론 적으로 35 epoch을 설정하고 Model training을 했을 때 최종 loss와 accuracy는 둘이 큰 차이가 없었지만 Optimizer module 부분의 그래프를 보아도 epoch 수가 50은 넘어야 optimizer들 간 큰 차이가 발생함을 알 수 있다. 또한 Adam의 알고리즘이 훨씬 complex함에 따라 각 epoch별 training speed에서 2 sec 정도 오래 걸림을 확인할 수 있다.

개인적으로 이번 프로젝트에서 중점적으로 생각했던 부분은 많지 않은 epoch 수에서 (일반적으로 Optimizer들 간 퍼포먼스 차이가 분명한 최소 epoch 수 50~100) 빠른 learning speed를 갖고 목표 Accuracy에 도달하는 것이기 때문에, SGD를 선택하였다. 만약 Training시간에 여유를 갖고 epoch수를 늘렸다면 Adam optimizer를 이용해 최적화를 했을 것이다.

## 모듈 이름: Model Prediction





### 기능

Model에 대해 주어진 input sample에 대한 output prediction을 생성해내는 모듈이다. 이렇게 생성한 output prediction과 실제 test output을 사진과 함께 출력해준다(plot\_images 모듈). 이후, output prediction값과 실제 test output을 heatmap을 이용하여 data sample이 얼마나 제대로 Image recognition/classification이 되었는지를 확인할 수 있다.

# 코드

# -\*- coding: utf-8 -\*-

"""05.CIFAR10.ipynb의 사본

Automatically generated by Colaboratory.

Original file is located at

https://colab.research.google.com/drive/1j2i27aEUQcYHSwc3vu-4qq3xz9Hw\_fWI

# Import modules

"""

import keras

from keras.datasets import cifar10

from keras.models import Sequential

from keras.layers import Dense, Dropout, Flatten, Activation

from keras.layers import Conv2D, MaxPooling2D

from keras import backend as K

from keras.utils.vis\_utils import model\_to\_dot

from IPython.display import SVG

# %matplotlib inline

import matplotlib.pyplot as plt

from sklearn.metrics import confusion\_matrix

import pandas as pd

import seaborn as sns

"""# Tuning parameters"""

epochs = 35

learning\_rate=0.0285 #Parameter, Layer, 등을 바꿔본다든지. Keras.io/examples/cifar10\_cnn/ 의 문서를 잘 읽어보고 비슷하게 따라하면 된다. 프로젝트는 코드가 비슷할테니, '분석'을 얼마만큼 잘 할 것인지. 코드는 정확성이 75프로 이상이면 만점. 보고서에서 모델에 대한 이해를 얼마만큼했느냐!

batch\_size = 32

num\_classes = 10

"""# Data

## Plot image

"""

def plot\_images(x, y\_true, y\_pred=None, size=(5, 5)):

assert len(x) == len(y\_true) == size[0] \* size[1]

fig, axes = plt.subplots(size[0], size[1])

fig.subplots\_adjust(hspace=0.5, wspace=0.1)

for i, ax in enumerate(axes.flat):

if x[i].shape[-1] == 1:

ax.imshow(x[i].reshape(x[i].shape[0], x[i].shape[1]))

else:

ax.imshow(x[i])

if y\_pred is None:

xlabel = "True: {0}".format(y\_true[i].argmax())

else:

xlabel = "True: {0}, Pred: {1}".format(y\_true[i].argmax(),

y\_pred[i].argmax())

ax.set\_xlabel(xlabel)

ax.set\_xticks([])

ax.set\_yticks([])

plt.show()

"""## Load dataset"""

(x\_train, y\_train), (x\_test, y\_test) = cifar10.load\_data()

"""## Reshaping the data"""

if len(x\_train.shape) < 4:

x\_train = x\_train.reshape(x\_train.shape[0], x\_train.shape[1], x\_train.shape[2], 1)

x\_test = x\_test.reshape(x\_test.shape[0], x\_train.shape[1], x\_train.shape[2], 1)

x\_train = x\_train.astype('float32')

x\_test = x\_test.astype('float32')

x\_train /= 255

x\_test /= 255

print('x\_train shape:', x\_train.shape)

print(x\_train.shape[0], 'train samples')

print(x\_test.shape[0], 'test samples')

"""## Applying One hot encoding for the data"""

y\_train = keras.utils.to\_categorical(y\_train, num\_classes)

y\_test = keras.utils.to\_categorical(y\_test, num\_classes)

"""## Show data"""

plot\_images(x\_train[:25], y\_train[:25])

"""# Creating the DNN model"""

model = Sequential()

"""## Adding layers to the model"""

model.add(Conv2D(48, (3, 3), padding='same',input\_shape=x\_train.shape[1:]))

model.add(Activation('relu'))

model.add(Conv2D(48, (3, 3)))

model.add(Activation('relu'))

model.add(MaxPooling2D(pool\_size=(2, 2)))

model.add(Dropout(0.4))

model.add(Conv2D(48, (3, 3), padding='same'))

model.add(Activation('relu'))

model.add(Conv2D(48, (3, 3)))

model.add(Activation('relu'))

model.add(MaxPooling2D(pool\_size=(2, 2)))

model.add(Dropout(0.4))

model.add(Flatten())

model.add(Dense(512))

model.add(Activation('relu'))

model.add(Dropout(0.5))

model.add(Dense(num\_classes))

model.add(Activation('softmax'))

"""## Visualization the model"""

SVG(model\_to\_dot(model, show\_shapes=True).create(prog='dot', format='svg'))

"""## Optimizer"""

#optimizer = keras.optimizers.Adam(lr=learning\_rate, beta\_1=0.9, beta\_2=0.999) #Lr = 0.001

optimizer = keras.optimizers.SGD(lr=learning\_rate) #, Lr = 0.00285

"""## Compiling the model"""

model.compile(loss=keras.losses.categorical\_crossentropy,

optimizer=optimizer,

metrics=['accuracy'])

"""## Training the model"""

model.fit(x\_train, y\_train,

batch\_size=batch\_size,

epochs=epochs,

verbose=1,

validation\_data=(x\_test, y\_test))

"""## Evaulating the model"""

score = model.evaluate(x\_test, y\_test, verbose=0)

print('Test loss:', score[0])

print('Test accuracy:', score[1])

"""## Prediction the model"""

y\_pred = model.predict(x\_test)

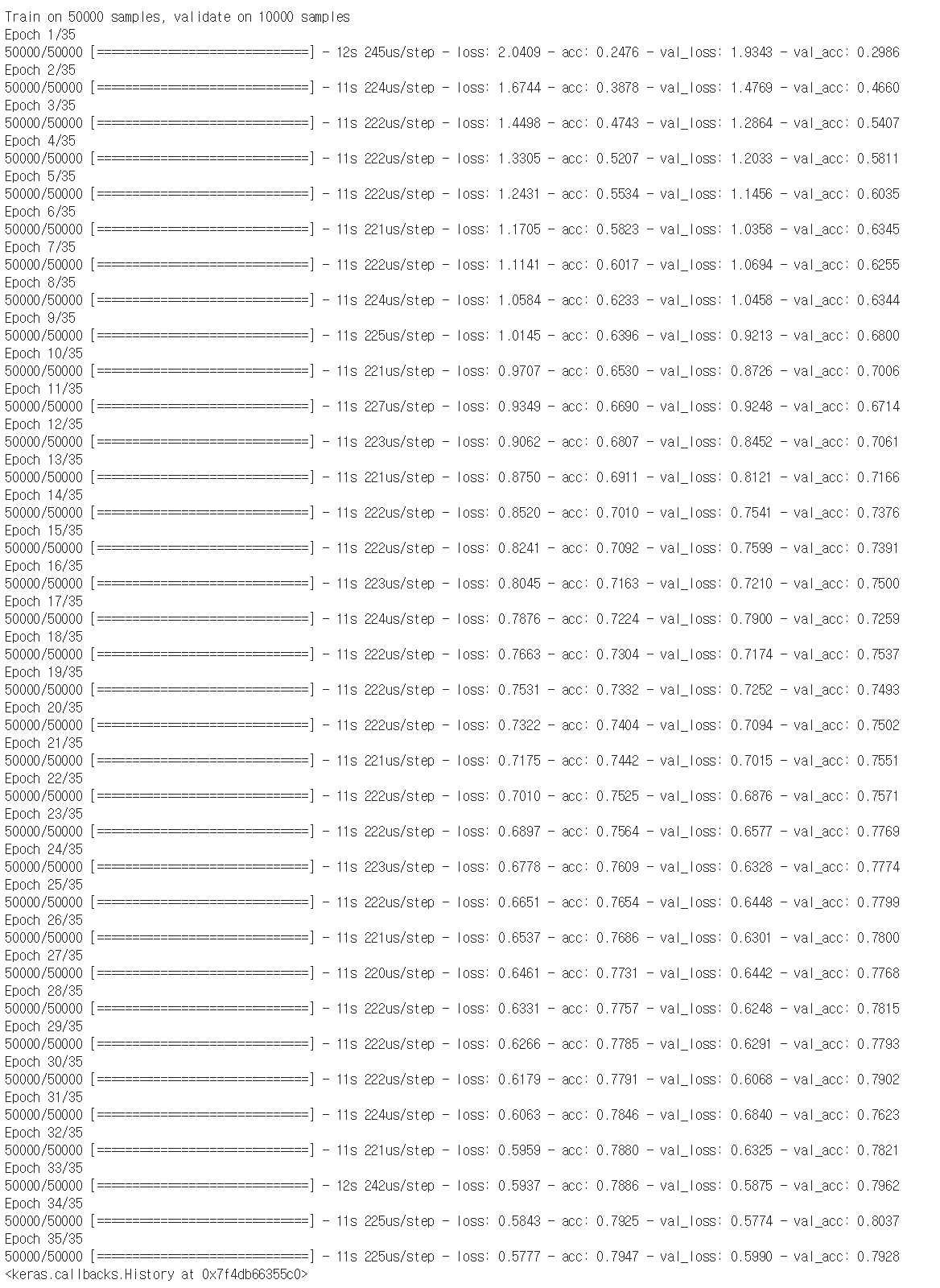
plot\_images(x=x\_test[:25], y\_true=y\_test[:25], y\_pred=y\_pred[:25])

y\_result = confusion\_matrix(y\_test.argmax(axis=1), y\_pred.argmax(axis=1))

sns.heatmap(pd.DataFrame(y\_result, range(10), range(10)), annot=True, fmt='g')

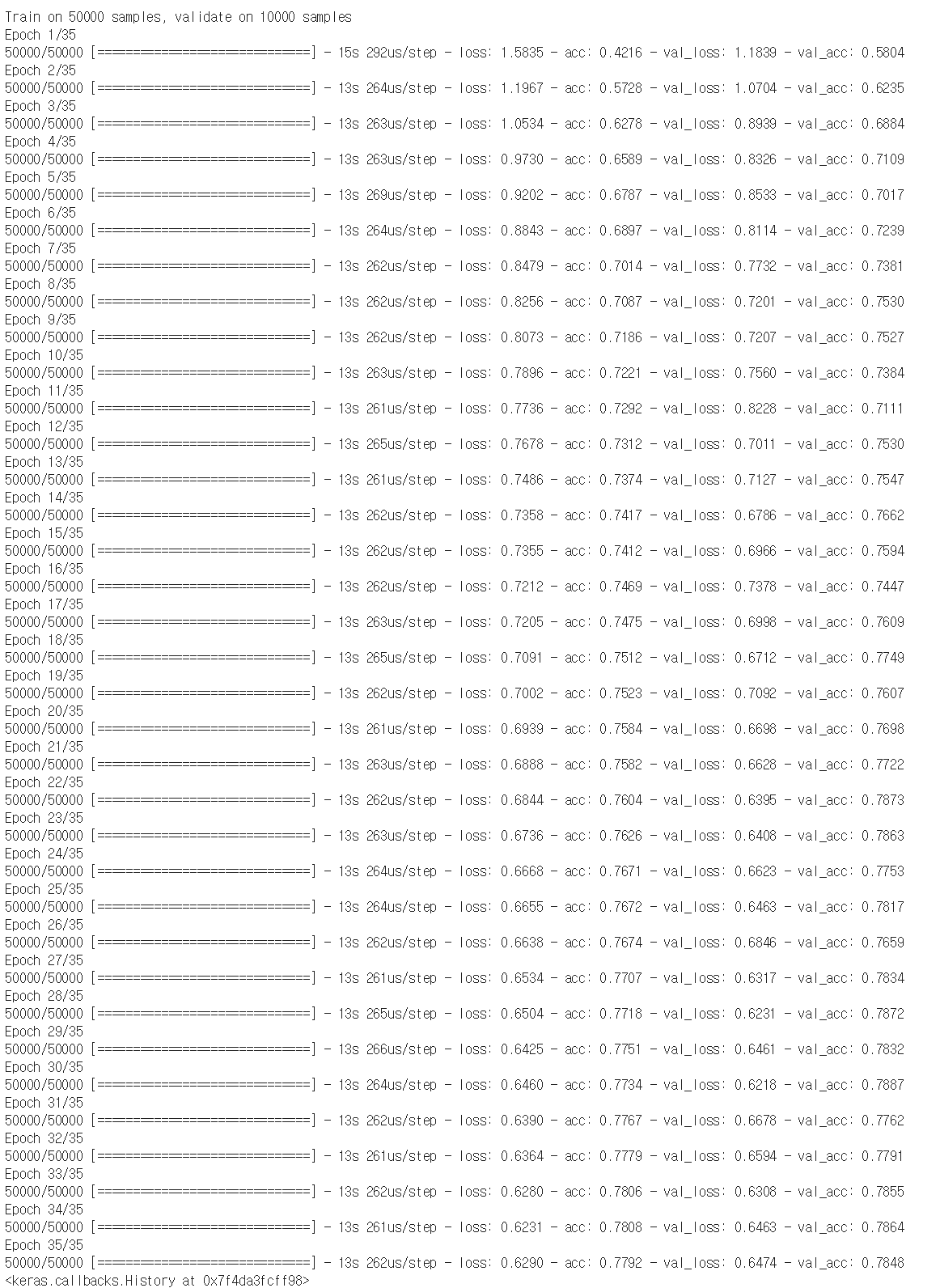
# Appendix – Model Training record

1. SGD training record



<SGD training record – Learning rate : 0.0285, Epoch : 35, batch\_size : 32>

1. Adam training record



<Adam training record – Learning rate : 0.001, Epoch : 35, batch\_size : 32>