**2019학년도 1학기 시스템 프로그래밍 Project 5 보고서**

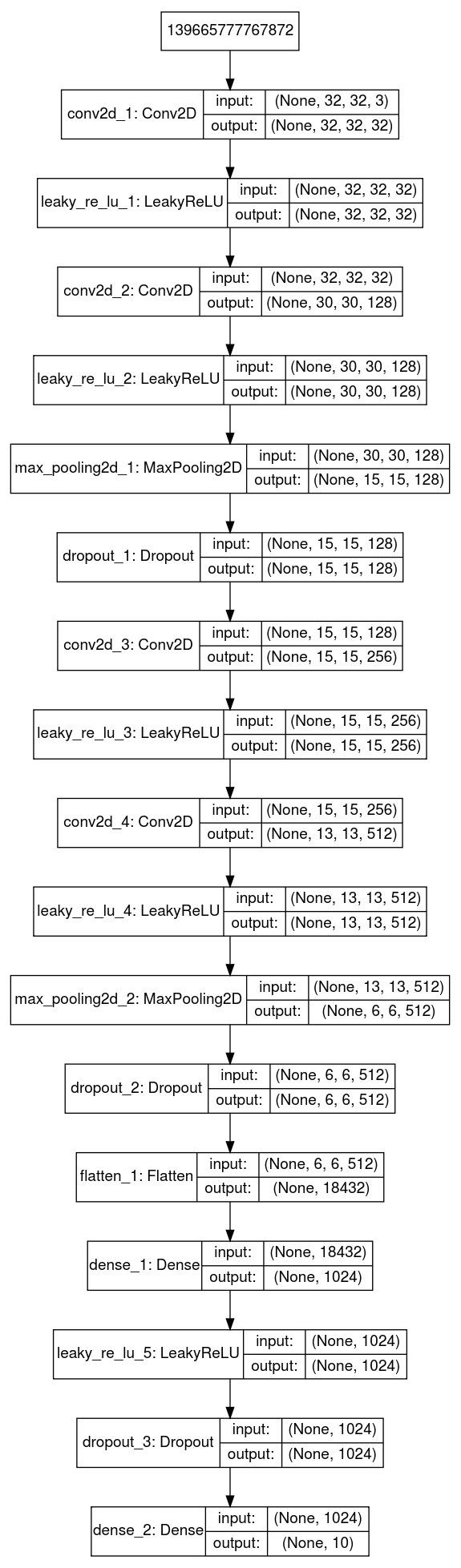
**담당교수: 서강대학교 컴퓨터공학과 소정민**

**1. 프로그램 개요**

이번 프로젝트에서는 기계 학습과 클라우드 서비스 중 하나인 Google Colaboratory와 Keras 라이브러리에 대한 이해도를 높이는 것을 목표로 한다.

Google Colaboratory 환경에서 Keras 라이브러리를 사용하여 기계 학습을 통해 CIFAR-10 dataset에서 주어진 사진들을 10개의 category에 맞게 구분하는 Python 코드의 정확도를 필요 기준치 75% 이상인 약 89.5%까지 개선하는 작업을 진행했다.

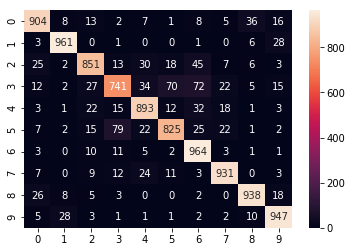
**2. 프로그램 설명**

 기계 학습을 위해 Keras 라이브러리를 import하여 사용했다. Keras는 high-level의 neural networks API이며, Python으로 작성되었고 Google이 개발한 기계학습 플랫폼인 TensorFlow에서 돌아갈 수 있다. Keras 라이브러리는 DNN(Deep Neural Network) Learning에서 필요한 기능들이 모듈화 되어서 제공되는데, 그 중에는 layers, optimizers, models, datasets 등이 이에 해당된다. 이러한 기능들을 프로그램 도입부에서 라이브러리로부터 import 해온다.

CIFAR-10으로부터 training set과 test set에 해당되는 데이터를 load해 오고, 받은 데이터는 데이터의 총 개수, 가로와 세로 픽셀이 있는 2차원 이미지에 관한 데이터, 그리고 각 픽셀마다 0에서 255까지의 RGB 색상 데이터가 차원별로 존재하므로 다차원을 갖는다. 그래서 다차원 데이터를 정규화하기 위해 numpy array인 training set과 test set의 모든 데이터 data type을 floating으로 변환하고 255로 나눈다. 그리고 DNN learning의 input으로 넣었을 때 각 neuron의 output 계산하기 용이하도록 class vector들을 binary class 행렬로 변환한다.

DNN 모델은 여러 neuron들로 이루어진 input과 output layer, 그리고 다수의 hidden layer가 존재하며 layer끼리 신경망처럼 연결된 Multi-Layer Perceptron 모델이다. 우선 2차원의 이미지 데이터를 input으로 읽어서 kernel(filter)을 적용하여 모든 영역을 훑으며 feature를 추출하기 위해 input layer이면서 동시에 convolution을 수행하는 layer를 생성한다. 그리고 hidden layer로 2×2 행렬을 filter로 하여 max pooling을 수행하는 layer, 이에 관한 convolution과 max pooling을 수행하는 3개의 layer를 추가한다. 앞에서 convolution layer와 max pooling layer를 반복적으로 거치면서 주요 feature들이 추출된 결과를 10가지 중 하나로 분류하고자 다차원 데이터를 일차원으로 바꾸기 위해 flatten layer와 fully connected dense layer를 hidden layer로 추가한다. 마지막으로 10개의 neuron 중 하나가 가장 높은 output을 가지도록 하여 분류 결과를 도출하기 위해 output layer를 추가한다. 이후 생성한 layer를 DNN 모델로 컴파일 하는데, neural network의 weight를 gradient descent 방법을 사용하여 조정할 때 너무 많은 계산량을 요구하게 되어 부하가 생기는 것을 방지하고자 adam optimizer를 사용한다.

이 프로그램에서는 기계 학습 결과가 training set으로 overfitting 되는 것을 막기 위해 학습 과정에서 training set에 약간의 변화를 준 image들을 학습시키기 위해 ImageDataGenerator 클래스를 import하여 training set을 학습시킨다. 한꺼번에 모든 데이터를 학습시키는 것은 부하가 크기 때문에 batch size가 100인 데이터 set으로 나눠서 이를 500번 시행하여 낸 평균을 적용한다. Learning을 epoch만큼 횟수를 반복하며, learning rate는 lr\_schedule이라는 별도의 함수를 생성하여 반복 횟수가 늘어날수록 더 작은 learning rate 값이 callback 된다. 매번 training set에 따라 학습된 DNN에 test set을 input으로 넣어서 정확도가 얼마나 개선이 되었는지를 화면에 출력한다.



**3. 모듈 정의**

**3.1 Data Augmentation을 통해 Overfitting 방지하기**

Neural Network에 관해 기계 학습을 시킬 때, network가 training set에만 너무 정확하게 맞추려는 경향이 생긴다. 이처럼 학습시키는 데이터의 정확도를 높이는 데만 치중하여 정작 임의의 test data가 주어졌을 때 정작 정확도가 현저히 떨어지는 현상을 overfitting이라고 한다.

이 프로젝트에서 사용하는 CIFAR-10은 5만 개의 data set으로 학습을 시키기에는 현실적으로 데이터의 양이 너무나도 적다. 그래서 임의의 데이터가 들어왔을 때 5만 개의 training set에만 국한되어 학습된 결과에 따라 output을 내기 때문에 현실적으로 정확도를 높이기에 한계가 있다. 이에 대한 해결방안으로 이 프로그램에서는 training set을 학습시킬 때 정해진 training set만을 반복하여 학습시키지 않고, 매번 training set에 약간의 변화를 주어 부풀려진 다양한 데이터에 관해 학습이 이루어져서 새로운 이미지도 잘 분류하도록 한다.

프로그램에서 Keras의 Preprocessing 패키지에 존재하는 ImageDataGenerator 클래스를 import하고 있다. 이 클래스에 있는 함수를 통해 학습 도중에 이미지를 임의로 변형하거나 정규화를 적용할 수 있으며, 변형된 이미지를 batch size 단위로 불러올 수 있는 generator를 생성할 수 있다. 새로운 이미지를 생성할 때는 사용자가 여러 설정을 줄 수 있는데, 예를 들어 rotation\_range는 새로 생성할 이미지 데이터가 기존 이미지로부터 얼마나 회전되어 생성될 수 있는지를 설정할 수 있다. 아래는 기존 이미지로부터 약간의 변화를 준 새로운 데이터를 생성하기 위해 모듈의 parameter로서 직접 조작한 유의미한 설정 값이다. 표에 적힌 내용 외의 프로그램 코드에 작성된 나머지 parameter는 default 값으로 이미 설정이 되어 있는 것을 중복하여 작성하거나 해당 설정 기능을 활성화 하지 않은 것이다.

|  |  |  |
| --- | --- | --- |
| **Parameter 이름** | **설정한 Value** | **설명** |
| **width\_shift\_range** | 0.1 | 기존 이미지 데이터에서 수평 방향으로 얼마만큼 이미지 이동이 가능하여 임의의 새로운 이미지 데이터로 생성될 수 있는지를 의미하며, 0.1은 가로 방향으로 최대 10%까지 이동할 수 있다는 뜻이다. |
| **height\_shift\_range** | 0.1 | 기존 이미지 데이터에서 수직 방향으로 얼마만큼 이미지 이동이 가능하여 임의의 새로운 이미지 데이터로 생성될 수 있는지를 의미하며, 0.1은 세로 방향으로 최대 10%까지 이동할 수 있다는 뜻이다. |
| **fill\_mode** | nearest | 이미지를 회전하거나 이동 또는 축소시킬 때 생기는 빈 공간을 어떻게 채울 것인지를 정하며, ‘nearest’는 가까운 영역의 데이터로 빈 공간을 채우는 것을 의미한다. |
| **horizontal\_flip** | True | 기존 이미지를 수평 방향으로 뒤집어서 임의의 새로운 이미지를 생성할지 설정하는 것이며, True로 설정하면 50% 확률로 이미지를 수평 방향으로 뒤집는다. 이 프로젝트에서 수행하는 CIFAR-10은 좌우가 비대칭인 경우가 다수이고, 좌우가 바뀌어도 이미지 식별 결과가 달라지지는 않기 때문에 True로 설정했다. |

Data Augmentation을 통해 부풀려진 데이터를 학습시키는 것도 정확도를 높이는 방법이지만, 단순히 기하학적인 변형으로는 이미지의 특성이 크게 달라지지 않으면서 layer model이 갖는 전체적인 데이터의 양은 변함이 없기 때문에 분류 작업에 저해되는 특징들도 갖고 있을 수 있다. 그래서 모델의 complexity를 줄이는 방법도 같이 사용했다.

**3.2 Dropout 기법을 사용하여 Overfitting 방지하기**

DNN 모델이 input으로 주어진 이미지를 분류하는 데 필요한 특징적인 데이터도 가지고 있으나, training set을 통해서만 학습된 나머지 불필요하게 가지고 있는 데이터도 존재한다. 그래서 학습 과정의 batch sample마다 어떤 한 layer가 이전 layer로부터 같은 입력을 두 번 이상 받을 가능성을 현저히 줄이기 위해 매번 일시적으로 neural network에 있는 일부 요소를 삭제하여 gradient vector를 구하는 기법을 사용했다. 이는 프로그램 코드에서 DNN 모델의 layer를 생성할 때 convolution layer 사이를 제외한 각 layer 사이마다 Dropout(0.25) 또는 Dropout(0.5)를 삽입하여 구현했다. 여기서 Dropout 함수의 paramenter 값은 layer 사이에 얼마나 많은 neuron과 weight로 연결된 network를 삭제할지를 percentage로 나타낸 것이다. 결과적으로 학습 데이터에만 overfitting 되는 현상을 방지하면서 결과적으로 핵심적인 특징에 대한 가중치로 향해 갈 수 있다.

**3.3 Layer의 Activation Function을 Sigmoid에서 Leaky ReLU로 변경하기**

DNN 모델 layer의 어떤 한 neuron에서 이전 layer의 neuron들의 output과 neuron들과 연결된 net의 weight를 곱하여 모두 더한 값을 activation function의 input으로 넣는다. 그러면 neuron은 activation function을 통해 나온 output을 가지며, 학습 중간 과정에서 DNN 모델의 해당 layer가 input data의 feature에 관한 데이터를 갖도록 한다. 다시 말해서, activation function은 어떤 layer의 한 neuron에 이전 layer에서 온 여러 신호가 들어오면 다음 layer의 neuron에 보낼 신호의 강도를 결정하게 된다. 대표적인 activation function에는 sigmoid function이 있으며, 이번 프로젝트에서 주어진 코드에서는 layer의 activation function으로 sigmoid function을 사용했다. 이 함수는 모든 input에 관하여 0과 1 사이의 output을 갖으며, output이 1에 가까우면 다음 layer에 완전히 값을 전달하고 0에 가까우면 그렇지 않는다는 특징이 있어서 인체의 neuron과 유사하다는 특성이 있다. 식과 그래프는 아래와 같다.

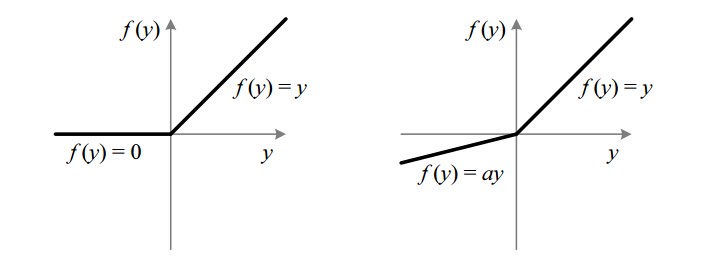


[출처: https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6]



그러나 학습 과정에서 weight를 조정하기 위해 각 network의 weight마다 gradient descent를 구해야 하는데, 이를 위해 back-propagation 알고리즘을 사용한다. 앞 layer의 weight를 조정하기 위해 cost function을 해당 weight로 편미분한 값을 구해야 하지만, 이를 직접 구할 수는 없기 때문에 chain rule을 사용하여 뒤의 layer의 neuron에서 계산되는 activation function의 output을 activation function에 넣기 전의 input으로 편미분한 값을 같이 계산해야 한다. 이는 뒤의 layer의 neuron에서의 activation function을 미분한 함수에 activation function의 output을 넣은 값을 계산하는 것과 동일한데, 여기서 sigmoid function의 문제가 발생한다. Sigmoid function은 미분하기 전의 output이 0 또는 1에 가까운 구간을 미분하면 0에 가까워지고 미분한 최댓값은 4분의 1이다. 다시 말해서, 초기에 임의로 weight 값이 설정될 때 0 또는 1에 근접하게 되면, weight 값이 학습 과정에서 크게 변하지 않고 매우 느리게 된다는 사실을 알 수 있다. 또한 back-propagation 알고리즘 사용하는 과정에서 gradient descent를 계산할 때 hidden layer의 개수가 많거나 epoch이 늘어나서 weight가 0 또는 1에 수렴하게 되면, 앞 layer network의 weight를 계산할 때 뒤의 layer에서 계산된 미분한 output이 0에 수렴하므로 weight 값이 초기에 비해 크게 변하지 않고 학습이 더디어진다. 그래서 DNN 모델의 정확도도 줄어들게 된다. 이를 vanishing gradient problem이라고 한다.

그래서 sigmoid function 대신 Leaky ReLU(rectified linear unit)라는 새로운 함수를 output layer를 제외한 layer들의 activation function으로 사용했다. Relu function은 아래 그래프를 통해 알 수 있다시피 input이 양수이면 미분한 값이 1로 일정하므로 gradient가 0으로 수렴해서 더 이상 특정 neuron에 대해 학습이 이루어지지 않는 현상을 막을 수 있다. 또한 미분한 값의 구간이 0보다 크고 최대 4분의 1 이하였던 기존의 sigmoid function에 비해 미분한 값이 상대적으로 커서 정확한 결과로 수렴하는 속도가 빠르다. 그러나 relu function은 input이 음수일 때 미분한 값이 0이어서 gradient descent가 0이 되는 현상이 발생할 수 있는데, 이는 학습 과정에서 한 번 neuron의 weight가 0이 되면 다시 neuron이 발현되기 어려운 문제가 발생한다. 이를 dying relu problem이라고 하며, 이러한 문제를 해결하기 위해 relu function의 음수 영역을 보완한 새로운 activation function인 leaky relu function을 사용했다. 아래는 Relu function과 leaky relu function에 대한 식과 그래프이다.



[출처: https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6]

DRW0000606074c5



이 프로그램에서는 layer의 activation function으로 leaky relu function을 사용하기 위해 output layer를 제외한 나머지 layer에 이를 적용했다. model의 멤버 함수인 add의 parameter인 activation으로는 linear relu function 사용을 제공하지 않기 때문에 직접 개선된 activation function 클래스(advanced\_activations)를 keras 라이브러리로부터 import 해오는 방법을 사용했다. 그리고 필요한 layer 사이에 leaky relu function 역할을 담당하는 layer를 추가했다. LeakyReLU 멤버 함수의 parameter로서 alpha는 음수 영역의 기울기를 얼마만큼으로 설정할지를 의미하는 것이며, 여기서는 기울기로 0.01을 적용했다.

**3.4 학습되는 Weight를 Adam Optimizer로 최적화하기**

DNN의 weight를 조정하는 과정에서 앞서 말한 바처럼 gradient descent 방법을 사용한다. DNN 모델이 내놓은 output과 실제로 정확한 output 사이의 차이를 정의하는 cost function을 최소화하기 위해 cost function이 나타내는 공간 위에 위치해 있는 기울기의 음수를 learning rate와 곱한 것을 기존 weight에 더하면서 정해진 epoch마다 이를 반복 시행하는 것이 weight를 전반적으로 수정해가는 과정이다. 그런데 매 epoch마다 전체 training set을 사용하여 weight를 최적화해 나가는 것은 계산량이 너무 많아서 부하를 줄 수 있으므로 cost function을 구할 때 전체 training set 대신 이를 임의의 작은 data set으로 나눈 것을 여러 개 가지고 학습시키는 방법을 SGD(Stochastic Gradient Descent)라고 한다. 이 프로그램에서도 SGD와 유사한 방식을 사용하기 위해 batch size와 steps per epoch 값을 설정하여 batch size만큼 작은 set으로 쪼갠 것을 steps per epoch만큼 한 epoch에 실행했다. 프로젝트에서 주어진 프로그램에서는 단순히 SGD 방법을 사용했지만, SGD는 전체 training set을 한꺼번에 학습시키는 것에 비해 정확도가 떨어질 수 있으며, 새롭게 변형하여 개발된 다른 방법들보다 정확한 output으로 향해가는 속도가 현저히 느리고 local minimum으로 수렴할 수 있는 단점이 있다.

이러한 단점을 보완하고자 이 프로그램에서는 SGD를 보완한 Adam(Adaptive Moment Estimation) 알고리즘 방법을 사용하여 학습되는 weight를 최적화했다. Adam 방식은 SGD의 변형 알고리즘들 중 하나인 Momentum과 RMSProp 방식을 합친 것과 유사하다.

Momentum 방식은 gradient descent를 구하여 cost function의 최적의 minimum으로 이동하는 과정에 일종의 관성을 주는 방법이다. Momentum 방법은 학습해 나가는 과정에서 자주 이동하는 방향으로 관성이 걸리게 되어서 SGD보다 더욱 빠르게 최적의 minimum으로 이동할 수 있으며, local minimum 주위로 들어간다 하더라도 기존에 움직이던 관성이 있기 때문에 SGD와 달리 local minimum에 빠지지 않고 나올 수 있다는 장점이 있다.

RMSProp은 weight들을 조정해 가는 과정에서 필요한 변수마다 step size를 다르게 설정해서 이동하는 방법이다. 기본적인 핵심 아이디어는 학습 과정에서 지금까지 많이 변화하지 않는 변수들의 step size는 크게 하고, 지금까지 많이 변화했던 변수들의 step size는 작게 하는 것이다. 자주 변화한 변수들은 최적의 경우에 근접하게 있을 확률이 높기 때문에 작은 크기로 이동하면서 세밀하게 값을 변화시키고, 자주 변화하지 않은 변수들은 최적의 경우로 도달하기 위해 많이 이동해야 할 확률이 높아서 크게 변화시키는 것이다.

Adam 방법에서는 이 두 가지 방법에다가 한 가지 최적화를 추가했다. 처음에 학습을 진행할 때 필요한 변수들이 0으로 초기화되어 있으므로 0과 가깝게 편중되어 있다고 판단하여 이를 편중되지 않게 만들어주는 작업을 거친다.

**3.5 Learning Rate를 Epoch가 커질수록 줄여 나가기**

Learning Rate는 cost function의 최적의 minimum로 가는 weight를 구하기 위해 현재 cost function에서의 weight로 편미분한 기울기의 반대 방향으로 얼마나 이동한 값을 더할 것인지를 결정짓는 요소이다. Learning Rate가 크면 기울기의 반대 방향으로 더 많이 이동하게 되고, 작으면 더 적게 이동한다. 그런데 learning rate가 너무 크면 cost function이 최적의 minimum에 근접했을 때 이를 지나칠 가능성이 있어서 진동하여 발산할 수 있고, learning rate가 작으면 학습하는 데 시간이 너무 오래 걸린다는 단점이 있다. 그래서 처음에 임의로 weight가 설정되고 cost function을 줄여 나가는 과정에서는 learning rate가 minimum으로 빠르게 수렴하도록 learning rate를 어느 정도 크게 주고, cost function이 웬만큼 줄어들었을 때는 발산 없이 최적의 minimum으로 수렴하도록 learning rate를 작게 설정하는 것이 바람직하다.

이 프로그램에서는 epoch 실행 횟수에 따라 learning rate가 조정되도록 lr\_schedule 함수를 별도로 작성했다. 초기에는 learning rate가 0.001이고, epoch가 50을 넘으면 0.0005로, 80을 넘으면 0.0001로, 100을 넘으면 0.000001로 설정된다. model의 멤버 함수인 fit\_generator의 parameter 중 하나인 callbacks로 lr\_schedule 함수를 넘겼으며, 이는 학습이 진행될 때 epoch 수에 따라 lr\_schedule 함수가 callback 되면서 함수에서 미리 정의된 바에 따라 learning rate가 조정된다.

**3.6 Epoch를 130으로 늘려서 training set에 관해 학습을 충분히 하기**

프로젝트에서 주어진 기존의 10만큼의 epoch를 130으로 늘려서 training set에 관해 DNN의 weight가 최적 상태로 조정될 수 있을 만큼 충분히 학습이 되도록 한다. 그러나 너무 epoch가 커지면 기존의 training set에만 정확한 값을 출력하도록 학습이 되어서 DNN 모델이 overfitting될 수 있다. 그래서 epoch를 무한정 많이 늘리는 것은 오히려 test set에 관해 정확도를 떨어뜨리는 부작용을 낳을 수 있다.

최적의 epoch 수는 DNN의 layer나 weight 수 등 다양한 변수에 영향을 받으므로 명확히 정해진 바가 없으나, 이번 프로그램을 여러 번 runtime하여 실행하면서 가장 정확도가 높이 나오는 epoch인 130으로 결정했다. 그리고 ‘3.1’에서 서술한 data augmentation과 ‘3.2’에서의 dropout 방법을 통해 주어진 training set에만 국한되어 학습되지 않고 부풀려진 data set을 학습할 수 있도록 구현했으므로 130만큼의 epoch로 학습이 된다고 하더라도 DNN이 overfitting 될 가능성은 낮다고 판단했다.

**3.7 Convolution Layer와 Max Pooling Layer 추가하고 Neuron 개수 늘리기**

‘3.3’에서 서술한 vanishing gradient problem에 의하면 너무 많은 layer의 추가는 높은 정확도를 보장하지 못하고 오히려 일부 neuron의 활성을 무효화해서 정확도를 떨어뜨리는 부작용을 낳는다. 그러나 적당량의 hidden layer를 추가하는 것은 정확도를 향상시키는 데 도움이 되며, 기존의 프로그램에서 각각 두 개의 convolution layer와 max pooling layer를 추가했다. Convolution layer는 학습시키는 input 데이터를 가지고 filter에 의해 pattern matching하여 feature의 존재에 관한 정보를 저장하는 데 유용하다. 특히, filter의 개수가 많을수록 training set에 관해 다양한 feature를 파악할 수 있으므로 일정량 늘리는 것이 좋다고 알려져 있다. Max pooling layer는 기존의 많은 데이터 중에서 분류에 저해가 되는 데이터는 제외하고 가장 유의미한 데이터만 뽑으면서 동시에 전체 DNN의 parameter 개수를 줄일 수 있기 때문에 정확도 개선에 도움이 된다.

이 프로그램에서는 DNN의 hidden layer로 추가하기 위해 Conv2D와 MaxPooling2D 함수를 사용했다. Convolution layer에서는 여러 다른 feature들을 학습할 수 있도록 3×3 크기의 filter의 개수를 128개, 256개, 512개로 설정했으며, max pooling layer에서는 2×2 크기의 filter에 의해서 데이터의 각 2×2 영역마다 maximum한 데이터 값을 추출하도록 했다.

Hidden layer의 neuron 개수를 늘리는 것이 항상 정확도가 높아짐을 보장하지는 않지만, DNN 모델의 representational capacity를 늘려주는 역할을 한다. 만약 정확도 저하가 representational capacity에 의한 것이라면 neuron의 개수를 늘리는 것은 정확도 개선에 도움이 된다. 이 프로그램에서는 convolution과 max pooling layer를 거치고 나서 이를 일차원 데이터로 바꾼 후에 연결되는 layer의 neuron의 개수를 1024개로 설정했다. Dropout 기법을 사용했으므로 DNN 모델이 training set에 overfitting 되는 문제는 발생할 가능성이 낮다고 판단했다.

**4. 전역 변수 정의**

프로젝트 보고서 작성 안내 사항에 의하여 생략.

**5. 코드**

import keras

from keras.datasets import cifar10

from keras.models import Sequential

from keras.preprocessing.image import ImageDataGenerator

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

from keras.layers.advanced\_activations import LeakyReLU

from keras.layers import Conv2D, MaxPooling2D

from keras import backend as K

from keras.callbacks import LearningRateScheduler

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

def lr\_schedule(epoch):

lrate=0.001

if epoch > 50:

lrate=0.0005

if epoch > 80:

lrate=0.0001

if epoch > 100:

lrate=0.00001

return lrate

epochs = 110

learning\_rate=0.0001

batch\_size = 100

num\_classes = 10

data\_augmentation = True

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()

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

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')

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

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

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

model = Sequential()

model.add(Conv2D(32, kernel\_size=(3, 3), padding='same', activation='linear',

input\_shape=x\_train.shape[1:]))

model.add(LeakyReLU(alpha=.01))

model.add(Conv2D(128, (3, 3), activation='linear'))

model.add(LeakyReLU(alpha=.001))

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

model.add(Dropout(0.25))

model.add(Conv2D(256, (3, 3), padding='same', activation='linear'))

model.add(LeakyReLU(alpha=.01))

model.add(Conv2D(512, (3, 3), activation='linear'))

model.add(LeakyReLU(alpha=.01))

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

model.add(Dropout(0.25))

model.add(Flatten())

model.add(Dense(1024, activation='linear'))

model.add(LeakyReLU(alpha=.01))

model.add(Dropout(0.5))

model.add(Dense(num\_classes, activation='softmax'))

if data\_augmentation:

datagen = ImageDataGenerator(

# set input mean to 0 over the dataset

featurewise\_center=False,

# set each sample mean to 0

samplewise\_center=False,

# divide inputs by std of dataset

featurewise\_std\_normalization=False,

# divide each input by its std

samplewise\_std\_normalization=False,

# apply ZCA whitening

zca\_whitening=False,

# epsilon for ZCA whitening

zca\_epsilon=1e-06,

# randomly rotate images in the range (deg 0 to 180)

rotation\_range=0,

# randomly shift images horizontally

width\_shift\_range=0.1,

# randomly shift images vertically

height\_shift\_range=0.1,

# set range for random shear

shear\_range=0.,

# set range for random zoom

zoom\_range=0.,

# set range for random channel shifts

channel\_shift\_range=0.,

# set mode for filling points outside the input boundaries

fill\_mode='nearest',

# value used for fill\_mode = "constant"

cval=0.,

# randomly flip images

horizontal\_flip=True,

# randomly flip images

vertical\_flip=False,

# set rescaling factor (applied before any other transformation)

rescale=None,

# set function that will be applied on each input

preprocessing\_function=None,

# image data format, either "channels\_first" or "channels\_last"

data\_format=None,

# fraction of images reserved for validation (strictly between 0 and 1)

validation\_split=0.0)

datagen.fit(x\_train)

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

optimizer = keras.optimizers.Adam(lr=learning\_rate,decay=1e-6)

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

optimizer=optimizer,

metrics=['accuracy'])

if not data\_augmentation:

model.fit(x\_train, y\_train, batch\_size=batch\_size, epochs=epochs, verbose=1, validation\_data=(x\_test, y\_test), shuffle = True)

else:

model.fit\_generator(datagen.flow(x\_train, y\_train, batch\_size=batch\_size),

epochs=epochs,

steps\_per\_epoch = x\_train.shape[0] // batch\_size,

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

callbacks=[LearningRateScheduler(lr\_schedule)])

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

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

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

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')