Journal of Knowledge Information Technology and Systems(JKITS), Vol. 14, No. 5, pp. 553~561, October 2019
https://doi.org/10.34163/jkits.2019.14.5.012

Journal of Knowledge Information Technology and Systems

ISSN 1975-7700

http://www.kkits.or.kr

Effective Implementation for Fast Deep Learning Algorithm

Sangmin Suh*

Samsung Electronics

ABSTRACT

AI (Artificial Intelligence) based on deep learning has been successful in many application areas. Supervised learning such as image classification and object detection has been mainly used for vision and ADAS (Advanced Driver Assistance Systems) / AD (Autonomous Driving). And reinforce learning has been generally utilized for robotics and energy optimization. Therefore, in order to improve the performance, many research papers have focused on optimizing neural networks. However, in practice, FPS (frame per second) is a hidden and critical factor because FPS is also included in the performance measurement. This note show that pre-processing and post-processing are major components affecting FPS. And It is verified that FPS cannot be improved by optimizing the neural network itself because the pre-processing and post-processing are out of the neural networks. In this note, fast pre-processing methods on the basis of DSP (digital signal processing) is suggested. For DSP implementation, binary arithmetic is presented and quantization error due to the conversion from floating point calculation to fixed point calculation is discussed. In addition, major design frameworks for deep learning algorithm implementation are compared and their merit and demerit are also summarized. In the note, implementation is categorized into three, i.e., input data generation with pre-processing, model design of neural network, and performance evaluation. With the selected framework, detailed implementation is also presented.

KEYWORDS: Artificial intelligence, Deep learning, Digital signal processing, Image classification, Keras, Tensorflow

ARTICLE INFO: Received 25 September 2019, Revised 5 October 2019, Accepted 11 October 2019.

*Corresponding author is with the deep learning algorithm part of multimedia team, Samsung electronics, 1-1, Samsungjeonja-ro, Hwaseong-si, Gyeonggi-do, 18448, KOREA.

E-mail address: sangmin.suh@samsung.com

1. 서 론

인공지능은 최근에 가장 연구가 활발히 진행되고 있는 분야 중에 하나이다. 인공지능은 신경망(neural network)을 기반으로 구현이 되는데, 이러한 신경망의 기본 개념은 이미 오래전에 개발되었으며[1], 인공지능의 역사는 상당히 오래된 분야이다[2]. 기본 개념은 여러 개의 입력을 선형 조합으로 가중 합 (weighted sum)을 구한 후에 활성화 함수 (activation function)을 통과함으로써 여러 입력들의 정보를 종합적으로 판단하는 것이다.

초창기에는 단순히 하나의 레이어 (layer)만을 이용하였기 때문에, 단순한 XOR (exclusive OR) 분류 문제조차도 해결하지 못하였다. 여기에, Minsky 교수는 하나의 레이어를 갖는 신경망은 XOR를 해결할 수 없음을 수학적으로 증명까지 하였다[3]. 또한 XOR문제를 해결하기 위하여 은닉층 (hidden layer)을 갖는 다중 레이어 신경망 구조 (MLP: multi-layer perceptron)를 제안하여 XOR 문제를 해결할 수는 있다고 하였으나, 실제로 그 신경망을 이루는 가중치(weight)와 바이어스 (bias)는 계산할 수 없다고 하였다.

그 후, 이러한 가중치 계산에 대한 문제를 해결하기 위하여, Hinton 교수는 연쇄법칙 (chain rule)을 이용한 역전파 (back propagation) 방법을 제안하여 가중치와 바이어스를 계산하는 문제를 해결하였다[4]. 그러나 이러한 이론적 발전에도 불구하고, 다중 레이어 신경망에서 가중치와 바이어스 계산을 위한 연산량이 너무 커서 당시의 계산기(computer)로는 시간이 너무 많이 걸리기 때문에 실제 사용이 거의 불가능하였다.

실제로 적용이 가능한 신경망은, AlexNet으로 알려진 2012 ILSVRC (ImageNet Large-Scale Visual Recognition)에서 발표된 논문이다[5]. 논문에서는, 학습속도를 향상하기 위하여 활성화 함수로서 ReLU (rectified linear unit) 제안하였고, 학습시에는 성능이 좋으나 실제 테스트에서는 성능이 떨어지는 문제인 과적합 (overfitting) 문제를 줄이기 위한 드롭아웃(dropout)과 같은 여러 가지 기술들이 제안되었다. 그러나 가장 중요한 것은, 기존의 방법에서는 CPU (central processing unit)를 사용한 반면, 이 논문에서는 GPU (graphics processing unit)를 사용하였다는 것이다. 일반적으로 CPU는 SISD (single instruction single data)로 동작하지만, GPU는 SIMD (single instruction multiple data)를 지원하여, 하나의 명령어로 어려개의 동일한 연산을 한꺼번에 할 수 있도록 설계되었다. 이것은 원래 영상처리를 하기 위하여 지원되는 기능이었는데, 이것을 신경망 연산에 사용함으로써 연산 속도를 크게 향상시켜 여러 번의 학습을 가능하게 하여 정밀도 (accuracy) 향상에 도움이 되었다. 최근에서 병렬처리를 위한 MIMD (multiple instruction multiple data)용 프로세서도 개발되었다. 실제 신경망에서에 가장 큰 비중을 차지하는 2D-컨볼루션 (2 dimensional convolution)은 백터 연산으로 이루어져 있기 때문에 GPU가 훨씬 효과적이다. 현재는, 거의 모든 신경망 학습(training)과 실험(inference)에서 GPU 혹은 신경망에 더욱 특화된 NPU (neural processing unit)를 사용한다. 또한, 구글 (google)에서는 TPU (tensor processing unit)를 개발하여 클라우드 서비스에 사용하고 있다.

AlexNet 이후로 수많은 종류의 신경망이 개발되고 있으며, 한정된 메모리 (memory)나 곱셈기 (multiplier) 자원들을 효율적으로 사용하기 위한 노력이 진행 중이다. 이를 위해 현재는 완전 연결층 (fully connected layer)을 최대 풀링 (max pooling)이나 평균 풀링 (average pooling)층으로 변경하는 방향으로 신경망 설계가 이루어지고 있다.

GPU를 이용한 인공지능 알고리즘은 크게 세 가지로 분류되는데, 지도 학습 (supervised learning),

비지도 학습 (unsupervised learning), 그리고 강화 학습 (reinforcement learning)이 있다. 지도 학습은 정보와 그 정보가 무엇인지를 알려주는 정답을 한 쌍으로 입력하여 훈련시킴으로써, 신경망이 새로운 정보를 받아들일 때 그 정보가 무엇인지를 추측(prediction)하는 것이며 현재 가장 연구가 많이 되고 있는 분야이다. 비지도 학습은 주로 차원 축소(dimensionality reduction)와 같이 정보를 압축-추상화하여 정보 추출을 효율적으로 만드는데 사용된다. 강화 학습은 환경에서 받은 정보를 에이전트(신경망)에 인가한 후, 어떤 최적의 동작을 할 것인가 결정하는 문제이다. 강화 학습에서 최적의 동작이라는 것은, 미리 규정된 보상(reward)을 최대로 받도록 하는 행동을 말한다.

이러한 여러 가지 인공 지능 알고리즘은 다양한 응용 분야에서 사용되고 있는데, 데이터 센터 (data center), 스마트 폰과 같은 모바일 응용 (mobile application), ADAS (Advanced Driver Assistance Systems) / AD (Autonomous Driving)[6,7], 제어(control)-로보틱스 (robotics)[8], 의료 영상[9], 그리고 스마트 농장 (smart farm)[10] 등에서 사용되고 있다.

이 논문에서는 신경망 구현을 위한 실제적인 여러 가지 설계 방법 중 더욱 효과적인 방법을 소개한다. 그리고 현재까지는 주로 정밀도만을 고려하여 신경망을 설계하였다면, 그와 달리 이 논문에서는 전처리 과정 (pre-processing)과 후처리 과정 (post- processing)의 중요함을 논한다. 그리고 그 성능향상을 위한 방안으로, 이진 산술법 (binary arithmetic)을 이용한 효과적인 하드웨어 가속기 (hardware accelerator) 설계 방법을 제안하고 DSP (digital signal processing)을 이용하여 구현한다.

논문의 구성은 다음과 같다. 다음 장에서는 현재 사용되고 있는 여러 가지 신경망 프레임 워크 (framework)을 비교 설명하며, 그 중에 가장 효율적인 방법인 텐서플로우-케라스를 이용하여 각 레이어를 어떻게 구현하여 모델을 생성하는지 보인다. 그 후 설계된 모델을 이용한 실험결과를 보이고 고속 전처리를 위한 DSP 설계 방안을 제시한다.

2. 신경망 설계

이 장에서는 신경망 설계를 위하여 각 은닉층을 포함한 실제 각 레이어들의 설계와 구현방법을 보인다.

2.1 프레임워크 선정

신경망 설계를 위한 프레임워크는 대표적으로 텐서플로우 (Tensorflow), 케라스 (Keras), 파이토치 (Pytorch), 카페 (Caffe)가 있으며 모두 오픈 소스이다[11].

텐서플로우[12]는 구글에서 만든 것으로 현재 가장 많이 사용되고 있고 그만큼 수많은 신경망들이 텐서플로우로 이미 구현되어있어서 사용자들이 참고할만한 정보가 풍부한 장점이 있다. 그런데 이 텐서플로우는 노드(nod: 레이어 동작에 해당)와 엣지(edge: 텐서 데이터 흐름에 해당)를 이용하여 그래프를 먼저 만들고, 그 그래프를 따로 실행해야 한다. 그러나 이러한 설계 방식은 직관적이지 못하다는 단점이 있다. 이러한 단점을 해결하기 위하여 한 소프트웨어 엔지니어가 케라스[13]를 개발하였는데, 신경망 설계가 텐서플로우보다 훨씬 쉽다. 파이토치는 페이스북[14]이 개발한 프레임워크로 케라스와 같이 신경망 설계가 용이하다는 장점이 있다. 마지막으로 카페[15]는 대학교에서 만든 것으로, 소개하는 프레임워크 중에 유일하게 C++로 신경망을 설계하여야 한다. 최근에, 구글이 텐서플로우가 사용하기가 복잡하다고 생각하여 케라스와 텐서플로우를 통합한 방법을 제안하였다. 이것은

새로운 프레임워크는 아니고, 기존의 텐서플로우에 케라스를 서브모듈(sub- module)로 포함시켜 각각의 장점을 하나로 합친 것이다.

저자가 생각하는 각 프레임워크의 장단점을 <표 1>에 나타내었고, 텐서플로우-케라스 (TF-Keras)가 설계에 있어서 가장 용이하다고 생각하여, 이 논문에서는 텐서플로우-케라스를 사용하여 이미지 분류기 (image classification)용 신경망을 구현한다.

표 1. 여러 가지 인공지능 프레임워크
Table 1. Various frameworks for deep learning

Name	Design flexibility	Usage	Language
Tensorflow	low	high	Python
Keras	high	mid	Python
PyTorch	high	high	Python
Caffe	mid	low	C++
TF-Keras	high	high	Python

2.2 신경망 구현

텐서플로우-케라스를 이용하여 신경망을 구현하는 방법은 크게 두 가지로 나뉜다. 하나는 sequential API (application programming interface)를 이용하여 구현하는 것이고, 다른 하나는 functional API를 이용하여 구현하는 것이다. 먼저 간단한 신경망 설계를 생각한다. 입력의 개수는 100개이고 두 개의 완전 연결층으로 구성되어 있으며, 첫 번째 레이어는 64개의 뉴런 노드가 있고 두 번째는 10개의 뉴런 노드가 있는 신경망을 고려해 보자. 그러면, 식 1과 같이 매핑(mapping) 관계로 표현이 가능하다.

$$x \xrightarrow{L_0} h \xrightarrow{L_1} y \qquad (1)$$

여기서, $x \in R^{100}$, $h \in R^{64}$, $y \in R^{10}$이다. 그리고, L_i는 각 레이어를 나타내며, L_0의 활성화 함수는 ReLU이고 L_1의 활성화 함수는 softmax이다. 그리고 이것을 도식적으로 표현하면, <그림 1>과 같이 표현할 수 있다.

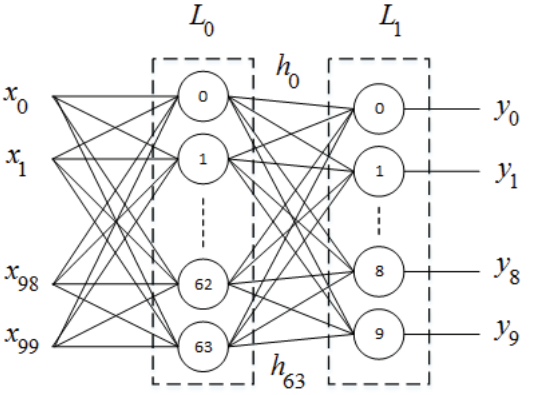

그림 1. 두 개의 완전 연결층
Figure 1. two fully connected layers

이것을 먼저 sequential API를 이용하여 구현하면 <그림 2>와 같다.

```
# Base model definition
model = tf.keras.models.Sequential()

# L0: fully connected layer with 64 nodes
# and ReLU activation function
model.add(tf.keras.layers.Dense(units=64, activation='relu',
                                input_dim=100))

# L1: fully connected layer with 10 nodes
# and softmax activation
model.add(tf.keras.layers.Dense(units=10,
                                activation='softmax'))
```

그림 2. Sequential API를 이용한 구현
Figure 2. Sequential API implementation

이 방법은, 초기에 모델을 정의하고 각 레이어들을 순차적으로 그대로 명기하기만 하면 되기 때문에, 연속적으로 이루어진 mobileNet[16]과 같은 직렬형 신경망 설계에서는 사용하기가 용이하다. 그

러나 노드들이 분리되었다가 다시 합쳐지는 shuffleNet[17]이나 inception V3[18] 모델의 경우 사용하기가 복잡해진다.

이러한 경우, functional API를 이용한 구현이 보다 명확하며, 이를 <그림 3>에 보인다.

```
# input data definition
input_data = tf.keras.layers.Input(shape =
                                   (img_h, img_w, img_c))

# L0: fully connected layer with 64 nodes
# and ReLU activation
L0 = tf.keras.layers.Dense(units=64, activation='relu',
                           input_dim=100)(input_data)

# L1: fully connected layer with 10 nodes
# and softmax activation
L1 = tf.keras.layers.Dense(units=10, activation='softmax')(L0)
```

그림 3. Functional API를 이용한 구현
Figure 3. Functional API implementation

여기서는 sequential API를 이용한 것과 레이어의 구현 방법은 거의 일치하지만, 연속적으로 레이어를 쌓는 것이 아니고, 입력을 별도로 지정해줄 수 있는 방법이다. 그러므로 일반적인 신경망 설계에서는 이 방법이 더 유용하며, 이 논문에서도 functional API를 이용하여 이미지 분류기를 구현할 것이다.

표 2. 신경망 구조
Table 2. Neural network structure

Layer	Layer type	Output Shape
input	InputLayer	(32, 32, 3)
conv2D_1	Conv2D	(32, 32, 32)
dropout_1	Dropout	(32, 32, 32)
max_pooling2d_1	MaxPooling2	(16,16,32)
flatten_1	Flatten	(8192)
fully_conn_1	Dense	(10)

이 논문에선 <표 2>와 같은 이미지 분류기를 설계한다. 훈련과 실험을 위한 데이터로는 10개의 이미지를 구분하는 CIFAR-10 이미지를 이용하였고, 이미지 크기는 높이×폭×채널 = 32×32×3이다. 드롭아웃(Dropout) 레이어는, 과적합을 방지하기 위하여 추가된 것이며 훈련(training) 시에만 사용된다.

신경망의 구성은 크게 세부분으로 나뉜다. 첫 번째는 입력 데이터를 받는 부분으로, 입력 영상들의 크기가 각각 다르므로 그 크기를 같도록 하여야 한다. 이는 설계자가 임의로 신경망을 설계할 수 있다고 하더라도, 일단 신경망이 결정되면 읽어 들일 수 있는 영상의 크기가 정해져 있기 때문이다.

```
# image data load
(x_train, y_train), (x_test, y_test) =
                        tf.keras.datasets.cifar10.load_data()

# int8 to float32 conversion
x_train = np.float32(x_train)
x_test = np.float32(x_test)

# image normalization
x_train = x_train / 255
x_test = x_test / 255

# label
y_train = tf.keras.utils.to_categorical(y_train, num_classes)
y_test = tf.keras.utils.to_categorical(y_test, num_classes)

# image dimension
img_h = x_train.shape[1]     # image height
img_w = x_train.shape[2]     # image width
img_c = x_train.shape[3]     # image channel

# input data definition
input_data = tf.keras.layers.Input(shape =
                                   (img_h, img_w, img_c))
```

그림 4. 입력 데이터 처리부
Figure 4. Input data processing

<그림 4>는 입력 데이터 처리부를 나타내고 있다. 8비트 정수로 되어있는 데이터를 32비트 부동소수점 형태로 변경한 후 정규화(normalization)을 하여 입력으로 사용하였다.

두 번째는 신경망 모델의 구현이다. 이미지 분류기의 신경망은 여러 가지 방법으로 구현할 수 있지만, 이미지를 추상화(abstraction)할 때는 주로 2D 컨볼루션을 사용하여 피처맵 (feature map)의 채널

수를 늘리며, 최대 풀링 (maximum pooling)과 평균 풀링 (average pooling)으로 각 레이어의 크기를 줄이면서 이루어진다. 그리고 각 레이어들의 출력에 활성화 함수를 추가하면 하나의 레이어가 완성된다. 이렇게 이루어진 레이어들을 계속 이어 나가면, 최종적으로는 채널 수는 깊고 피처맵의 크기는 작은 출력 피처맵이 얻어진다. 그리고 그 출력을 소프트맥스 (softmax) 레이어에 통과 시키면, 주어진 입력 영상에 대하여 여러 가지 피사체가 있는 확률들이 계산되고, 그 중에 제일 큰 확률을 가진 클래스가 선정된다. <그림 5>은 <표 2>에서 정의된 신경망을 실제로 구현한 것이다.

```
conv2d_1 = tf.keras.layers.Conv2D(input_shape=in_shape,
  padding='same', kernel_initializer='glorot_normal',
  bias_initializer='zeros', filters=32, kernel_size=(3,3),
  activation='relu')(input_data)
dropout_1 = tf.keras.layers.Dropout(rate=0.4)(conv2d_1)
maxpool2d_1 = tf.keras.layers.MaxPooling2D(pool_size=(2,2))
                (dropout_1)
flat_1 = tf.keras.layers.Flatten()(maxpool2d_1)
softmax = tf.keras.layers.Dense(units=num_classes,
            kernel_initializer='glorot_normal',
            activation='softmax')(flat_1)
output_data = softmax

model = tf.keras.models.Model(inputs=[input_data],
                              outputs=[output_data])
model.compile(optimizer=tf.keras.optimizers.Adam(lr=0.001),
            loss=tf.keras.losses.categorical_crossentropy,
            metrics=['accuracy'])
```

그림 5. 신경망 모델 설계
Figure 5. Neural network design

세 번째는 훈련된 모델의 성능을 분석하여 타당한 결과가 나오는지를 확인해야 하며, 그렇지 않을 경우는 모델을 재설계하여야 한다.

<그림 6>은 <그림 5>에서 설계된 신경망을 학습하고 그 결과를 보여주는 프로그램이고 이를 통하여 설계된 신경망의 성능을 평가 할 수가 있다. 최고의 성능을 얻기 위해서는 드롭아웃 계수 (dropout_rate), 에폭(epoch), 그리고 훈련 계수 (learning rate) 등과 같은 하이퍼 파라미터 (hyper-parameter)를 조절해야 한다. 여기서는 dropout_rate=0.4, epoch=64, learning_rate=0.001을 이용하여 정밀도(Top-1)는 81.34%의 결과를 얻었다.

```
model.fit(x=x_train, y=y_train, epochs=epochs,
          batch_size=batch_size, callbacks=[tensorboard])

train_loss,train_accuracy=model.evaluate(x=x_train, y=y_train)
print('train data loss:{:2.4f}'.format(train_loss))
print('train accuracy:{:2.4f}'.format(train_accuracy))

test_loss, test_accuracy = model.evaluate(x=x_test, y=y_test)
print('test data loss:{:2.4f}'.format(test_loss))
print('test accuracy:{:2.4f}'.format(test_accuracy))
```

그림 6. 신경망 훈련과 평가
Figure 6. Training and evaluation of the designed neural network

3. 고속 신경망을 위한 DSP 구현

신경망에서 일반적인 성능은 입력 이미지를 얼마나 잘 분류하느냐를 말한다. 그리고 이 평가지표를 정밀도 (accuracy)라고 한다. 그런데, 간과하기 쉬운 또 하나의 평가 지수가 있는데, 그것은 FPS (frame per second)이다. 이것은 정밀도와는 다르게, 들어오는 입력 이미지에 대하여 얼마나 빠른 시간 안에 분류를 하는지를 나타낸다. 이것은 주로 빠른 동영상에서 이미지를 분류하는 작업에서 평가 지표로 사용된다. 그리고 이것은 주로 신경망 자체보다는 주변 하드웨어의 성능에 의하여 영향을 받는다. <그림 5>에서는 전처리 과정으로 데이터를 정규화를 보여주고 있는데, 실제 응용에서는 그보다 많은 전처리가 필요하다. 예를 들어, 카메라나 캠코더를 통하여 들어오는 이미지의 크기는 실제 신경망의 입력크기와 다르기 때문에 입력영상의 크기를 조절하여야만 한다. 양선형 보간법 (bilinear interpolation)이나 입방 스플라인 보간 (cubic spline interpolation)등의 방법이 사용될 수

있는데, 이 작업은 연산시간을 많이 필요로 하기 때문에 FPS에 많은 영향을 준다.

그러므로 빠른 전처리를 위하여 디지털 신호 처리기 (DSP processor: digital signal processing processor)를 이용하거나 verilog HDL (hardware description language)를 이용하여 보다 직접적인 하드웨어 가속기 (hardware accelerator) 사용을 제안한다. 입력 이미지는 RGB 세개의 채널로 구성이 되어 있으며 고정 소수점으로 표현된다. 이 원본 이미지를 양선형 보간법과 정규화에 적용을 하려면 DSP 프로세서에서 부동 소수점 연산을 수행해야만 하고, 이를 위해서는 이진 산술 (binary arithmetic) [19] 연산 방법을 사용해야만 한다.

이 장에서는 여러 가지 연산법 중에서 가장 중요한 곱셈기 구현 방법을 보인다. 사용가능한 하드웨어는 8비트이고 $C = A(0.1234) \times B(0.5678)$를 계산한다고 가정한다. 부동 소수점 결과는 0.070067이다. 두 개의 피연산자를 정수로 표현하면 a=round(A·2^7)=16, b=round(B·2^7)=73이다. 이것을 정수 곱셈기에 적용하면 결과는 16·73=1168이 되고, 이것은 2^7이 두 번 곱해지는 형태가 되므로 2^7로 나누어야 정확한 결과, floor(1168/2^7)=9가 계산된다. 그리고 이 값을 부동 소수점으로 표현하면 9/2^7=0.070313이 된다.

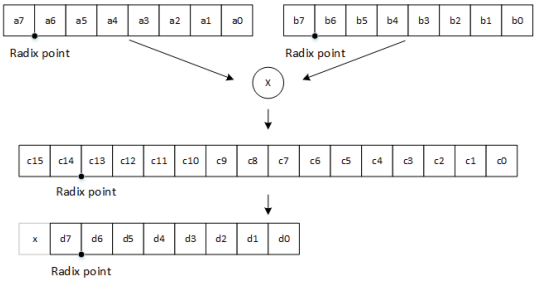

그림 7. 고정 소수점 표현법
Figure 7. Fixed point representation

그러므로 8비트 고정소수점 연산에서 곱셈할 때 양자화 오차(quantization error)는 0.070067−0.070313=−0.00025이고, 이러한 동작 원리를 <그림 7>에 보이고 있다. 그리고 만약 8비트가 아니고 16비트로 구성이 된다면 위의 수식을 통하여 양자화 잡음이 훨씬 줄어들어 성능이 덜 민감할 것이다.

일반 SISD계열의 CPU와 SIMD 계열의 DSP사이의 연산속도를 비교하여 이것이 FPS에 어떤 영향을 주는지 알아본다. H×W×C 크기의 이미지를 세로로, 가로를 정수 n, m만큼 양선형 보간법을 이용하여 크기를 변경한다고 할 때, 전체 이미지 픽셀(pixel)수는 (nH)×(mW)×C가 된다. 그러므로 이 과정에서 계산해야 할 픽셀의 수는 (n−1)×H×(m−1)×W×C가 된다. 보간법 계산에서는 덥셈기(adder)과 나눗셈기(divider)가 사용되는데, 일단 n과 m이 결정되면 정수 나눗셈기를 사용하지 않고 쉬프트 연산자들로만 나눗셈기를 만들 수가 있다. 그러므로 여기서는 간단한 예시를 위하여 덧셈기만 고려한다.

DSP를 설계할 때, 한 번에 몇 개의 연산을 동시에 할 수 있는 지는 설계자가 결정한다. 만약 16개의 연산을 한 번에 할 수 있는 DSP를 사용한다면, 위의 계산은 (n−1)×H×(m−1)×W×C/16으로 연산량이 줄어든다.

인공지능은 전처리와 실제 예측(inference)을 하는 신경망, 그리고 후처리로 구성이 되어 있고, 퍼센트 수행 시간을 각각 α, β, γ라고 한다면 한 장의 이미지를 처리하는 시간은 α+β+γ가 된다. 그러므로 DSP를 사용할 경우 아래와 같은 FPS 개선률을 얻게 된다.

$$\left(1 - \frac{(\alpha+\gamma)/N + \beta}{\alpha+\beta+\gamma}\right) \times 100\,[\%] \qquad (2)$$

여기서 N은 DSP가 한번 처리할 수 있는 벡터 연산의 수이다. 딥러닝의 또 다른 분야인 객체 탐지

(object detection)와 추적(object tracking)의 경우는 비최대값 억제 (NMS: non maximum suppression)라고 하는 후처리과정에 많은 연산이 요구되어 γ값도 추가적으로 크다. 이 경우에는 DSP를 이용한 방법이 더욱 효과적이다.

4. 결 론

이 논문은 인공지능 알고리즘을 빠르게 동작 할 수 있는 방법을 제시하였다. 우선 인공지능의 연구 방향에 대하여 정리하였고, 왜 GPU를 이용한 연산이 중요한 가에 관하여 논하였다. 대표적인 인공지능 설계 프레임 워크들의 장단점을 비교하였고, 신경망을 구현하는 자세한 방법을 제시하였다. 다른 논문들과 다르게, 전처리와 후처리 과정의 중요성을 보였고, 기존의 단점을 개선하기 위한 방안으로 디지털 신호처리를 이용한 방법을 제안하였다.

References

[1] F. Rosenblatt, *The perceptron: a probabilistic model for information storage and organization in the brain,* Cornell Aeronautical Laboratory, Psychological Review, Vol. 65, No. 6, pp. 386-408, 1958.

[2] M. Z. Alom, T. M. Taha, C. Yakopcic, S. Westberg, P. Sidike, M. S. Nasrin, B. C V. Esesn, A. S. Awwal, and V. K. Asari, *The history began from AlexNet: A comprehensive survey on deep learning approaches,* https://https://arxiv.org/abs/1803.01164, 2018.

[3] M. Minsky, and S. A. Papert, *Perceptrons: an introduction to computational geometry,* The MIT Press, 1969.

[4] D. H. Ackley, G. E. Hinton, and T. J. Sejnowski, *A learning algorithm for Boltzmann machines,* Cognitive science, Vol. 9, No. 1, pp.147-169, 1985.

[5] A. Krizhevsky, I. Sutskever, and G. E. Hinton, *Imagenet classification with deep convolutional neural networks,* https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf, Sep. 2019.

[6] R. Okuda. Y. Kajiwara, K. Terashima, *A survey of technical trend of ADAS and autonomous driving, Proceedings of Technical Program,* International Symposium on VLSI Technology, Systems and Application, 2014.

[7] A. Moujahid, M. E. Tantaoui, M. D. Hina, A. Soukane, A. Ortalda, A. ElKhadimi, and A. R. Cherif, *Machine learning techniques in ADAS: a review*, International Conference on Advances in Computing and Communication Engineering, 2018.

[8] H-K. Joung, and W-G. Oh, *A study on system identification using deep learning,* Journal of Knowledge Information Technology and Systems(JKITS), Vol. 14, No. 4, pp. 359-368, 2019.

[9] S. K. Zhou, H. Greenspan, and D. Shen, *Deep learning for medical image analysis,* Academic Press, 2017.

[10] S-H. Jeong, M-h. Lee, and H. Yoe, *Fruit classification system using deep learning*, Journal of Knowledge Information Technology and Systems(JKITS), Vol. 13, No. 5, pp. 589-595, 2018.

[11] J. Hale, *Deep learning framework power scores,* https://towardsdatascience.com/deep-learning-framework-power-scores-2018-23607ddf297a, Sep. 2019.

[12] Tensorflow, https://www.tensorflow.org, Sep. 2019.

[13] Keras, https://keras.io/, Sep. 2019.

[14] PyTorch, https://pytorch.org/, Sep. 2019.

[15] Caffe, https://caffe.berkeleyvision.org/, Sep. 2019.

[16] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L-C. Chen, *MobileNetV2: inverted residuals and linear bottlenecks,* https://arxiv.org/abs/1801.04381, 2018.

[17] X. Zhang, X. Zhou, M. Lin, and J. Sun, *ShuffleNet: an extremely efficient convolutional neural network for mobile devices*, https://arxiv.org/abs/1707.01083, 2017.

[18] C. Szegedy, V. Vanhoucke, S Ioffe, J. Shlens, and Z. Wojna, *Rethinking the inception architecture for computer vision,* https://arxiv.org/abs/1512.00567, 2015.

[19] P. Lapsley, J. Bier, A. Shoham, and E. A. Lee, *DSP processor fundamentals: architectures and features,* IEEE Press, 1997.

고속 딥러닝 알고리즘의 효과적인 구현

서상민

삼성전자 멀티미디어팀 딥러닝 알고리즘 수석 연구원

요 약

딥러닝 기반의 인공지능은 많은 응용 분야에서 성공적이었다. 이미지 분류나 물체 감지와 같은 지도 학습은 주로 비전과 자율주행 분야에 사용된다. 그리고 강화 학습은 로봇이나 에너지 최적화에 사용되고 있다. 따라서, 그 성능을 높이기 위하여 많은 연구 논문들이 신경망의 최적화에 집중하고 있다. 그러나 초당 프레임수는 겉으로는 드러나지 않은 또 하나의 중요한 성능지표이다. 이 노트는 전처리와 후처리가 초당 프레임수에 영향을 끼치는 중요한 인자임을 보이고, 신경망을 최적화하는 것으로는 전처리와 후처리를 개선할 수 없음을 보인다. 이것은 전처리와 후처리가 신경망 밖에 구성이 되어 있기 때문이다. 그래서 이 노트에서는 디지털 신호 처리 기술을 이용하여 전처리의 성능을 가속시키는 방법을 제안한다. 디지털 신호를 처리할 때, 양자화 잡음과 연산자들의 비트 길이와의 관계도 논한다. 실제 신경망의 구현은 세가지 단계로 나뉘어지는데, 전처리를 포함한 입력단, 신경망 구성부, 그리고 성능평가부로 나뉘며, 각 구성에 대하여 자세히 보여준다.

Sangmin Suh received the B.S., M.S. and Ph.D. degree in electronics engineering from Hanyang University of Seoul Korea in 1991, 1994, and 2003 respectively. From 1994 to 1999, he has been with Daewoo telecom. Since 2003, he is currently with Samsung electronics. His current research interests include deep learning based vision, reinforcement learning, robotics, and convex optimization.

E-mail address: sangmin.suh@samsung.com