GROUP # 5 (CONVOLUTIONAL NETWORKS) Report

EE 210A Winter 2017

Yahya Essa, Xiyuan Jiang, Cong Peng, Shashank Gowda

Abstract: CNN(Convolution Neural Network) is widely used in image processing and image classification. Conventionally, the inputs and outputs in convolution layers are treated as 2D matrices. In this report, the corresponding inputs and outputs are treated as vectors, which broaden its application.

In addition to implementing the CNN algorithm described in the handout, a different approach for back propagation is proposed. This approach copes with problems existing in current back propagation algorithm, such as computation efficiency and padding.

In the experiment part, test results of MNIST handwritten digits and objects classification validate the effectiveness of proposed algorithm.

I. A Different Approach for back propagation through Convolution Layers

The training algorithm presented in Section 27.25.6 of the notes relies on the back propagation of derivatives through the convolutional neural network. These derivatives are then used to calculate the derivative of the cost function (in the case of the note the mean square error) in terms of the weights and biases in the correlation layers and fully connected neural network.

For its earlier part, the back propagation algorithm is similar to its standard counterpart for neural networks. The derivatives calculations however, become a bit more technical and involved when we come to the correlation layers. This is mostly caused since correlation layers rely on operations (such as partitioning and pooling) which at first glance, may look complicated to back propagate through.

In this section, we try to simplify the forward algorithm through the convolution layer and then show how this simplification can help make the back propagation algorithm a bit more intuitive to follow. Additionally in this section, we also explicitly incorporate the padding operation operation which - in our opinion - was not explicitly handled in the notes (Although the framework described in the notes can be modified to incorporate it in the training algorithm in Section 27.25.6).

A. Feed Forward Algorithm Revisited

Here we try to revisit the operations in the forward algorithm and rewrite them all in linear format to get a mapping between the outputs of layer (c) and the output of layer $(c+1)$ that look similar to the structure in a neural network between layers ℓ and $\ell+1$.

As described in the notes, at the end of layer c, the signal $t^{(d,c)}$ is prepared for the next layer through the partitioning operation as follows

$$\left\{h_{n,p}^{(d,c)}\right\}_{p=1}^{P_{c+1}} = \text{partition}\left(t_n^{(d,c)}, P_{c+1}\right)$$

However in applications such as image classification, it is customary to pad the output of the previous layer with zeros before passing/partitioning it for the next layer, similar to Fig. 1. Thus we define the new operation

$$t_{n,pad}^{(d,c)} = \text{padding}\left(t_n^{(d,c)}, c+1\right),$$

and the partitioning is then made on $t_{n,pad}^{(d,c)}$ as

$$\left\{h_{n,p}^{(d,c)}\right\}_{p=1}^{P_{c+1}} = \text{partition}\left(t_{n,pad}^{(d,c)}\ , P_{c+1}\right).$$

Both the padding and partition operation, can be performed by linear operations. As the notes describe, padding can be performed through the partition matrix V_c. Thus we can stack the $h_{n,p}^{(d,c)}$ on top of each other to get

$$\text{col}\left\{h_{n,1}^{(d,c)}, \ldots, h_{n,P_{c+1}}^{(d,c)}\right\} = V_c\ t_{n,pad}^{(d,c)}. \tag{1}$$

For the padding operation, we can also define a padding matrix Q_c such

$$t_{n,pad}^{(d,c)} = Q_c \; t_n^{(d,c)}. \tag{2}$$

For the padding example showed in Fig 1 (assuming the elements of $t_n^{(d,c)}$ and $t_{n,pad}^{(d,c)}$ are indexed column-wise) we have that

$$t_{n,pad}^{(d,c)} = \begin{bmatrix} & \mathbf{0}_{4\times 4} & & \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ & \mathbf{0}_{2\times 4} & & \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ & \mathbf{0}_{4\times 4} & & \end{bmatrix} t_n^{(d,c)}.$$

It is not hard to see that the padding matrix Q_c is just an identity matrix with all-zero rows introduced where we need to pad zeros.

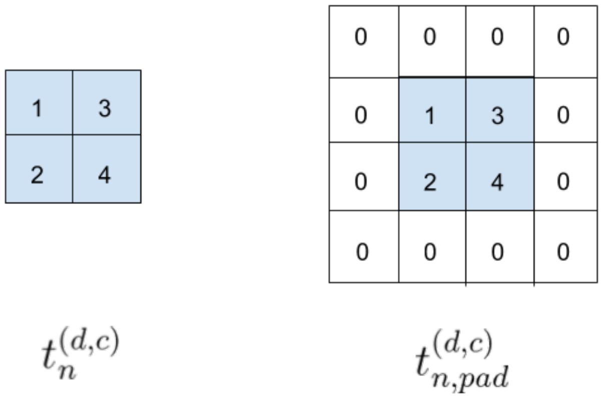

Fig. 1: Padding a 2×2 image with 1-level pad in all directions.

Thus we can combine (1) and (2) into

$$h_{n,col}^{(d,c)} = \mathrm{col}\left\{h_{n,1}^{(d,c)}, \ldots, h_{n,P_{c+1}}^{(d,c)}\right\} = V_c \; Q_c \; t_n^{(d,c)}. \tag{3}$$

Equation (3) captures the preparation that layer c has to do to the pooled output $t_n^{(d,c)}$ before sending information to layer $c+1$. As described in the Algorithm in (47.1216) in the notes, the calculation of $z_n^{(d,c+1)}$ in layer $c+1$ is done as follows

$$z_n^{(d,c+1)} = \sum_{j=1}^{D_c} \left(\left[H_n^{(j,c)}\right]^T w_{j,n-1}^{(d,c)} - \theta^{(d,c+1)}(j)\mathbb{1} \right), \tag{4}$$

where

$$H_n^{(j,c)} = \left[h_{n,1}^{(j,c)}, \; h_{n,2}^{(j,c)}, \; \ldots, h_{n,P_{c+1}}^{(j,c)}\right].$$

Here we actually propose a rewriting of this as

$$z_n^{(d,c+1)} = \sum_{j=1}^{D_c} \left(\left[I_{P_{c+1}} \otimes w_{j,n-1}^{(d,c+1)}\right]^T h_{n,col}^{(j,c)} - \theta^{(d,c+1)}(j)\mathbb{1} \right). \tag{5}$$

It is not hard to see that every row in the matrix $\left[I_{P_{c+1}} \otimes w_{j,n-1}^{(d,c+1)}\right]^T$ multiplies the section of $h_{n,col}^{(j,c)}$ that corresponds to only one vector $h_{n,i}^{(j,c)}$ by the coefficients in $w_{j,n-1}^{(d,c)}$. In other words, every element in $z_n^{(d,c)}$ through this calculation is only affected by one partition from depth j in the previous layer c. If we now introduce (3) into (5) we get a simple linear mapping between the vectors $t_n^{(j,c)}$ and $z_n^{(d,c+1)}$ as

$$\begin{aligned} z_n^{(d,c+1)} &= \sum_{j=1}^{D_c} \Bigg(\underbrace{\left[I_{P_{c+1}} \otimes w_j^{(d,c+1)}\right]^T V_c \; Q_c}_{\widehat{W}_{c,n-1}^{(j,d)}} \; t_n^{(j,c)} - \theta^{(d,c+1)}(j)\mathbb{1} \Bigg) \\ &= \sum_{j=1}^{D_c} \left(\widehat{W}_{c,n-1}^{(j,d)} \; t_n^{(j,c)} - \theta^{(d,c+1)}(j)\mathbb{1} \right). \end{aligned} \tag{6}$$

Remark 1: At this point it is interesting to note $\widehat{W}_{c,n-1}^{(j,d)}$ is a sparse matrix. This is due to the fact that each of its comprising components $[I_{P_{c+1}} \otimes w_j^{(d,c+1)}]$, V_c and Q_c are sparse matrices themselves. Thus $\widehat{W}_c^{(j,d)}$ can be computed and stored efficiently. This will play a big role when we discuss how this new rewritten formulation can reduce the running time and efficiency of the algorithm in Section 27.25.6 as we see in the following subsection.

B. Computing Sensitivity Factors for Correlation Layers

In this section, we show how to use the back propagation to calculate the sensitivity factors $\delta_n^{(d,c)}$ for correlation layer c and $d \in \{1, 2, \ldots, D_c\}$ given that we have the factors for the next layer $\delta_n^{(d',c+1)}$, $d' \in \{1, 2, \ldots, D_{c+1}\}$. Note that $\delta_n^{d,c}$ is defined as

$$\delta_n^{(d,c)} = \frac{\partial J(W,\theta)}{\partial z_n^{(d,c)}}.$$

As an initial step, we are actually interested in deriving the sensitivity factor in terms of $t_n^{(d,c)}$, we can derive that as follows

$$\begin{aligned}
\frac{\partial J(W,\theta)}{\partial t_n^{(d,c)}(i)} &\overset{(i)}{=} \sum_{d'=1}^{D_{c+1}} \sum_{k=1}^{P_{c+1}} \frac{\partial J(W,\theta)}{\partial z_n^{(d',c+1)}(k)} \frac{\partial z_n^{(d',c+1)}(k)}{\partial t_n^{(d,c)}(i)} \\
&\overset{(ii)}{=} \sum_{d'=1}^{D_{c+1}} \sum_{k=1}^{P_{c+1}} \frac{\partial J(W,\theta)}{\partial z_n^{(d',c+1)}(k)} [\widehat{W}_c^{(d,d')}]_{ki} \\
&= \sum_{d'=1}^{D_{c+1}} \sum_{k=1}^{P_{c+1}} \delta_n^{(d',c+1)}(k) [\widehat{W}_{c,n-1}^{(d,d')}]_{ki}, \qquad (7)
\end{aligned}$$

where: (i) follows from chain rule and (ii) follows from the expression in (6).

As a resut, we can write the computation in vector form as

$$\frac{\partial J(W,\theta)}{\partial t_n^{(d,c)}} = \sum_{d'=1}^{D_{c+1}} \left(\widehat{W}_{c,n-1}^{(d,d')}\right)^T \delta_n^{(d',c+1)}. \qquad (8)$$

What follows after this is very similar to the computation in the notes, where we can compute $\delta_n^{(d,c)}$ through upsampling and inverse permutation of $\partial J(W,\theta)/\partial t_n^{(d,c)}$

$$\delta_n^{(d,c)} = f'(z_n^{(d,c)}) \odot \text{permute}^{\#} \left(\text{upv} \left(\frac{\partial J(W,\theta)}{\partial t_n^{(d,c)}} \right) \right)$$

$$= f'(z_n^{(d,c)}) \odot \text{permute}^{\#}\left(\text{upv}\left(\sum_{d'=1}^{D_{c+1}}\left(\widehat{W}_{c,n-1}^{(d,d')}\right)^T \delta_n^{(d',c+1)}\right)\right). \tag{9}$$

Note that since $\text{permute}^{\#}$ and $\text{upv}()$ are linear operators, then we can also write $\delta_n^{(d,c)}$ as

$$\delta_n^{(d,c)} = \sum_{d'=1}^{D_{c+1}} f'(z_n^{(d,c)}) \odot \text{permute}^{\#}\left(\text{upv}\left(\left(\widehat{W}_{c,n-1}^{(d,d')}\right)^T \delta_n^{(d',c+1)}\right)\right),$$

however, we found through evaluation that the expression in (9) performs a lower number of matrices operations and is therefore more efficient to compute.

C. Why do we believe this is better ?

We start this section by mentioning that this back propagation formulation is more intuitive that what is presented in Section 27.25.6 of the notes. We highlight this through the following three points

1) **It is more intuitive:** The main reason, we start our argument reformulating the Feed Forward Algorithm (5) is to make the chain rule from $z_n^{(d',c+1)}$ to $t_n^{(d,c)}$ more explicit. If we ignore the padding operation, then in reality the matrix $W_{c,n-1}^{(d,d')}$ is the same as the matrix $V_{n-1}^{(d,d',c+1)}$ described in Section 27.25.6 of the notes. All we do in this formulation is to make its appearance in the chain rule more explicit.
2) **Computation and Storage Efficiency:** The main motivation for this formulation is that by following the Algorithm (47.1276) in the notes, we can quickly run into storage limitation when constructing matrices. Consider for example, if we want to construct the matrix $V_{n-1}^{(d,d',c+1)}$, when the output of layer c is a 256×256 2D grid image in each depth. If our image filter has a stride of 1, then the matrix $\left[\mathbb{1}_{P'_c}\mathbb{1}_{P_{c+1}}^T\right]$ needed would have a size of $256^2 \times 256^2$, which easily takes a lot of memory and slows down future computation in the construction of $V_{n-1}^{(d,d',c+1)}$. Note however ,that the matrix $V_{n-1}^{(d,d',c+1)}$ is sparse, therefore this generation of a big non-sparse matrix

in $256^2 \times 256^2$ could be saved using our construction where

$$V_{n-1}^{(d,d',c+1)} = W_{c,n-1}^{(d,d')} = [I_{P_{c+1}} \otimes w_{j,n-1}^{(d,c+1)}]^T \; V_c \; Q_c,$$

can be computed and stored efficiently since each of the component matrices are sparse. When evaluated, this constructed was seen to save on average 76% of the time needed for the computation proposed in (47.1276).

3) **Dealing with Padding in Back Propagation:** The back propagation method mentioned in the handouts does not address the issue of padding, which limits the usage of such algorithm. The formulation proposed deals with padding in back propagation via Eq. (9).

II. Experiment Results - MNIST handwritten digit database

In this chapter, we implement our CNN algorithm to MNIST handwritten digits to validate the effectiveness of proposed method.

A. Data Description

The database is consist of handwritten digits from 0 to 9, and the size of each image is 32 pixels * 32 pixels. There are 60000 digits in the training dataset and 10000 digits in the test dataset.

5 fold cross validation is used to choose optimal hyper-parameters such as number of layers, mask size and so on. After hyper-parameters are chosen, we calculate the test accuracy in the test dataset.

B. Use Cross Validation to Select Hyperparameters

In order to reduce running time we only use 10000 data images in the training stage. The network structure is:

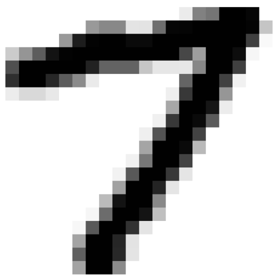

Fig. 2: example of MNIST handwritten digit

- step size $\mu = 0.01$;
- regularization term $\rho_1 = \rho_2 = 0.01$;
- 2 convolution layers of depth 5 and 5;
- For each convolution layer, the mask is 3 pixel by 3 pixel, padding 1, stride 2 and 1 respectively;
- For each convolution layer, the pooling mask is 2 pixel by 2 pixel;
- The activation function for the convolution layer is $y = tanh(z)$;
- 1 hidden fully connected layer of 200 neurons;
- The activation function for the hidden layer is $y = 1.7159\ tanh(\frac{2}{3}z)$
- Cross entropy loss is used as cost in the output layer.

- The training runs for epoch = 100 times;

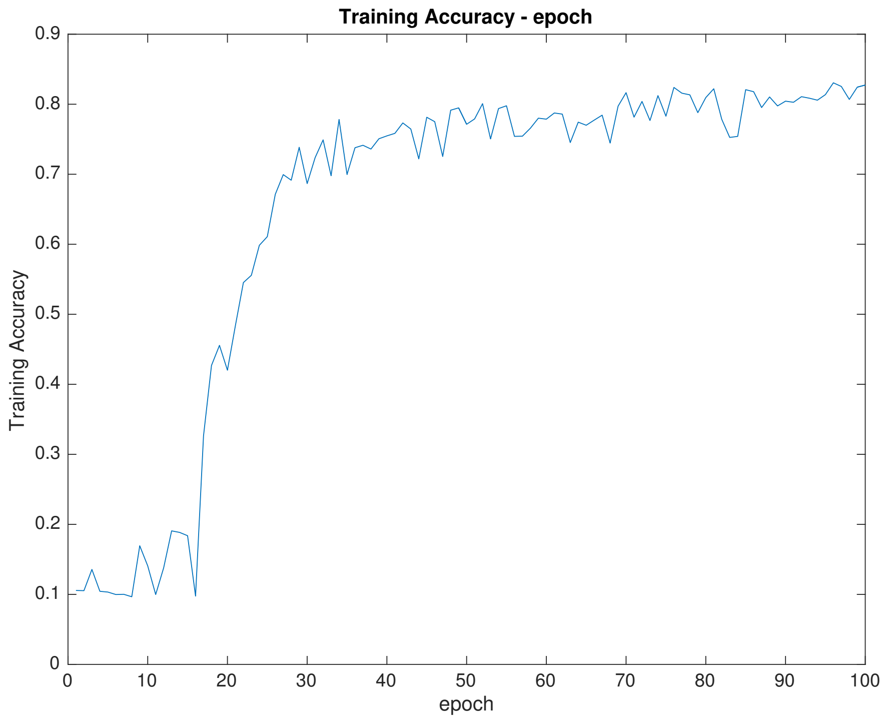

Fig. 3: training accuracy - epoch of stochastic gradient descent method

It could be seen from Fig.3 that in the first several training iterations, the accuracy is roughly 10%, which is reasonable because in the very first stage the network is not fully trained, so the output tend to be random. Since there are totally 10 digits, the random output yields an accuracy of 10%.

For cost function value, it decreases as the training goes on as expected. But its value keeps oscillating. This is because in stochastic gradient descent method, only one sample is used to compute the gradient, which makes the descent method unstable. This problem

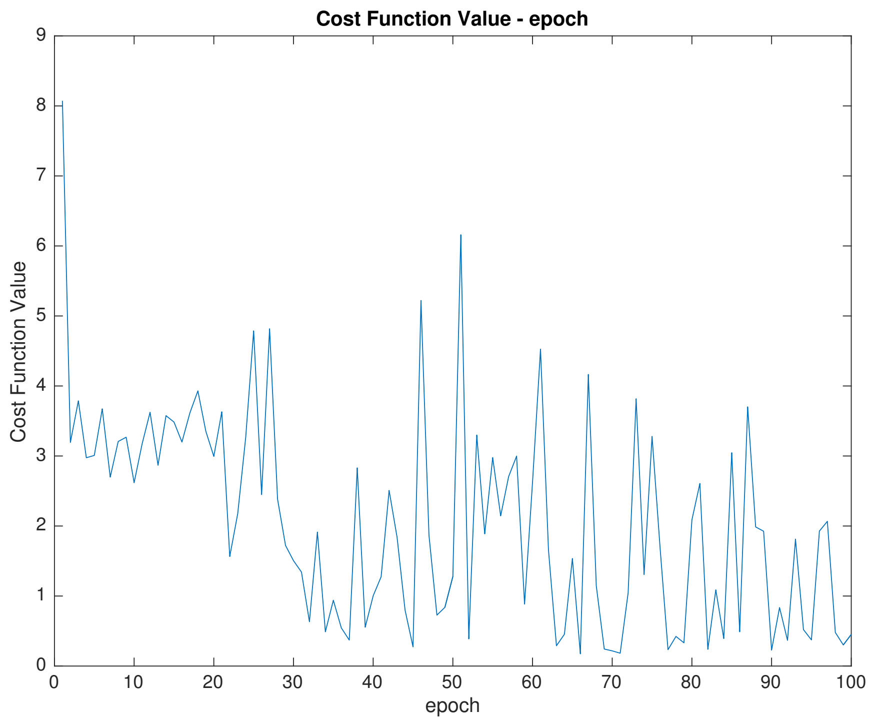

Fig. 4: cost function value - epoch of stochastic gradient descent method

could be handled via batch gradient descent. Fig.5 shows the result for batch size = 1, 5 and 10. It could be seen from the plot that batch gradient descent method not only decreases the oscillation, but also speeds up the descent procedure.

Finally, the validation accuracy is 83.72%

C. Test Results

With the same CNN structure, training is conducted on a training dataset of 60000 images with batch size = 10, and epoch = 60000. Then testing is done on a testing dataset of 10000 images. The test result is 94.11%, and the training accuracy is shown

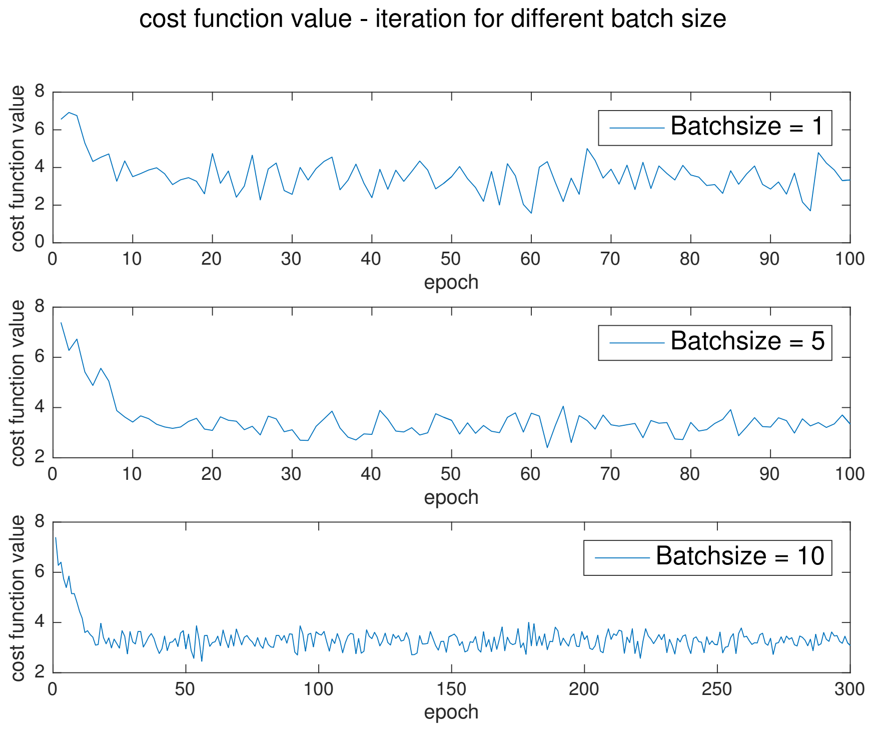

Fig. 5: cost function value - epoch for different batch size

in Fig. 6. This result show that the proposed CNN network structure and corresponding algorithm could be used as handwritten digit classification.

III. Experiment Results - ImageNet object classification database

With the proposed CNN model successfully implemented to handwritten database, further experiment is conducted on the ImageNet object classification database.

A. Data Description

Four classes of dogs images are chosen as datasets, each class contains roughly 160 images, respectively. The dog images is cropped to $256\ pixels * 256\ pixels$ before feeding

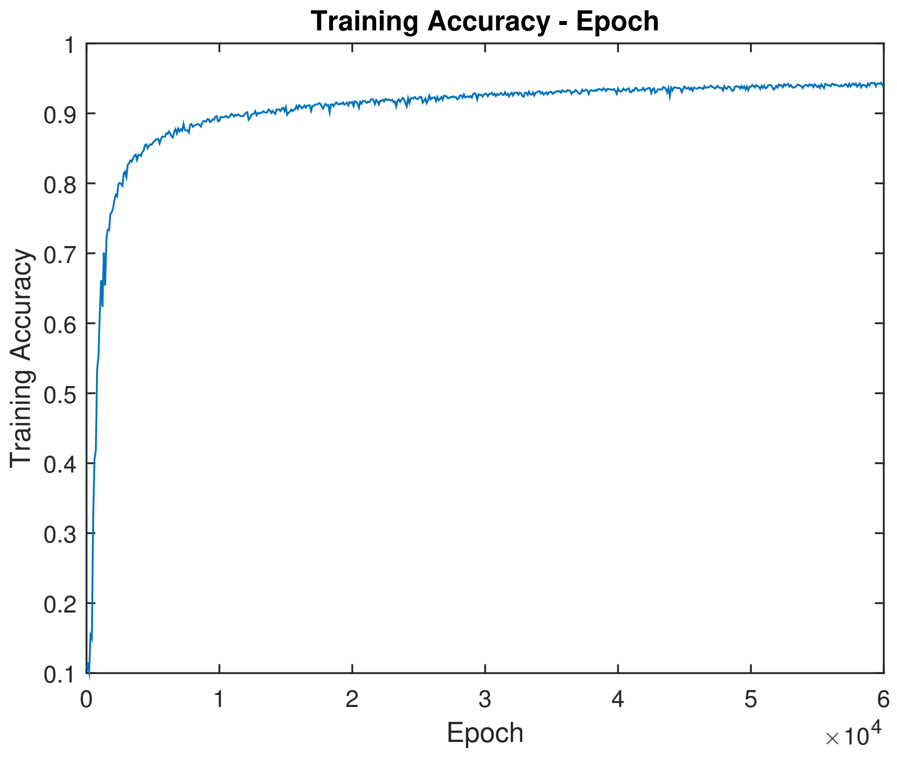

Fig. 6: cost function value - epoch for different batch size

into the network. 80% of data is chosen from each class randomly as training dataset, and the rest is test dataset. The network structure is the same as previous experiment.

B. Test Results

Stochastic gradient descent method is used in this experiment. The test accuracy is 45.24%. It could be seen, both from Fig.8 and the test accuracy that the proposed method also works for real object image classification. This is because if the network is not properly trained, the output would be random and the accuracy would be close to $\frac{1}{4} = 25\%$, while in the result the accuracy is around 45%, which proves that the network is being trained.

Fig. 7: example of dog classification images

However, the accuracy is not as good as MNIST handwritten digit results. The reasons are two-fold. First of all, dog images have larger pixel amounts, and the inputs have three depths (RGB), which makes it harder to train. In the meantime, the size of training dataset is much less than the handwritten dataset, which also prevents a better result.

For future research, a larger dataset and longer training time would make the results better.

IV. Some comments and typos in Section 27.25.6 of the notes

- in page 2545, equation (47.1223a), the size should be $S_c D_{c-1} * 1$.
- In Section 27.25.6, the symbol S_c is used to denote the size of the partitions presented to layer $c+1$ by layer c (Check Eq. 47.1257 and Eq. 47.1276). However, in earlier parts of the notes, particularly when explaining the feed forward algorithm the symbol used is S_{c+1} instead.
- In Algorithm (47.1276), there is a duplicate μ in the update of $W_{\ell,n}$.
- In Algorithm (47.1276), there is a typo in the update for the bias in the full neural

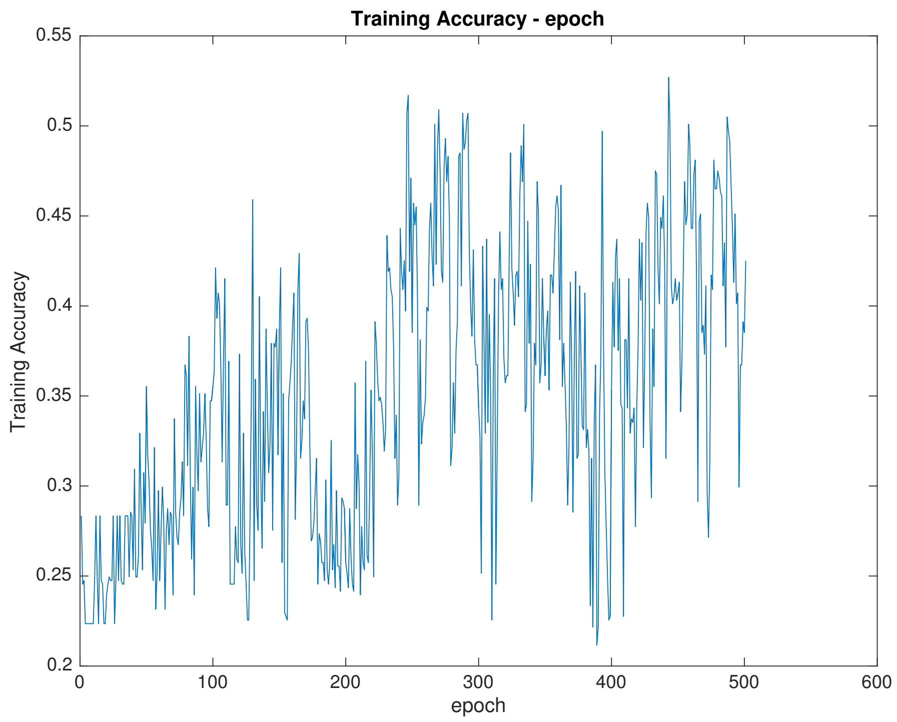

Fig. 8: training accuracy- epoch, dog images classification

network. The correct should be

$$\theta_{\ell,n} = \theta_{\ell,n-1} + \mu\delta_{\ell+1,n},$$

while in the notes, the second term is subtracted instead of the addition.

- We believe the explanation of the upsampling operation can be made for reader-friendly by including what is the vector that we use to upsample. For example, instead of using

$$\mathrm{upv}(x) \implies \mathrm{upv}(x, v),$$

where v is the vector used to upsample x.

- Another suggestion that the author might be interested to follow in the presentation of Convolution neural networks is to include the partitioning as an operation in the new layer. The way the structure is introduced in the paper makes the partitioning a task of the preceding layer, but this can introduce notation confusion when reading through the document. The suggestion therefore is to have the following structure instead.

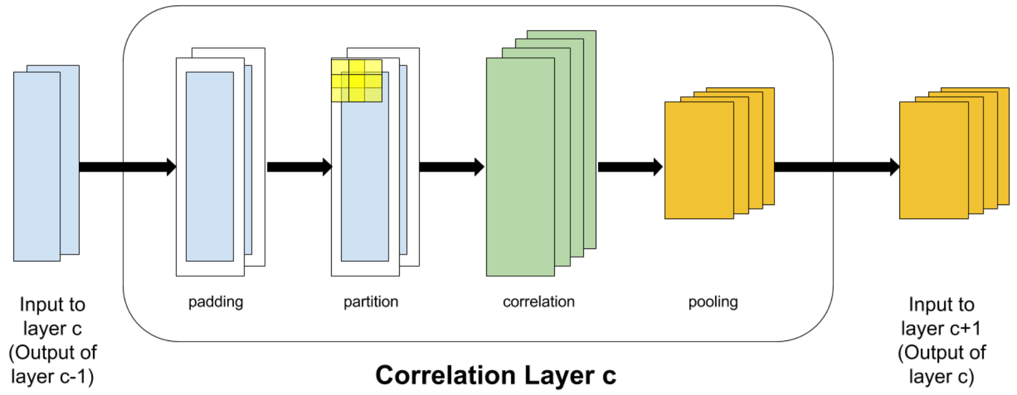
Input to layer c (Output of layer c-1)
padding
partition
correlation
pooling
Input to layer c+1 (Output of layer c)
Correlation Layer c

Fig. 9: New Proposed structure for the Correlation layer.