MLP Coursework 2: Investigating the Implementation and Optimization of Convolutional Networks

s1655829

Abstract

In this work, we investigate the performance of convolutional neural networks for the classification of color images on CIFAR 100 dataset. We will firstly introduce some background knowledge relates to the investigation topics. Then, we will concisely explain the idea of convolutional layers and how it can be implemented, in term of its computation efficiency and storage demands. Then, we will show a broken convolutional network caused by gradient vanishing problem. Two solutions called "Batch Normalisation" and "Residual Network" will be provided to overcome the problem. The principles of the solutions will be explained. The solutions will be compared and contrasted. At the end, we will combined two solutions as one and used it as our baseline model. Then, three regularisation methods will be used and discussed for improving the generalisation.

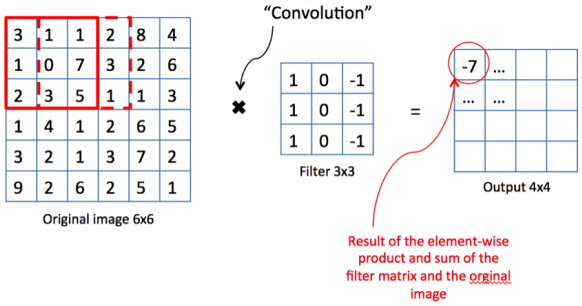
1. Introduction

Recently, deep learning has become so popular because of its supremacy in terms of accuracy when trained with a huge amount of data. Deep learning is simply learning from examples using a neural network (1). Neural networks consist of a set of algorithms that mimics the working of the human brain for recognizing the patterns in a dataset(2). A neuron (or a node) is where computation has taken place and an activation function bypasses the result from one node to another(3). A neural network contains multiple layers and each layer contains multiple neurons(4). The first and last layers are called the input layer and output layer respectively. All the intermediate layers are called hidden layers. The layers that connect every input from one layer to another is called fully connected layers (5), and the network consists only the fully connected layers are fully connected neural network. It offers features learned from all combinations of the features of the previous layers.

Deep learning has many usages in real life such as computer vision and pattern recognition, computing games, robots...(6). A fully connected neural network is competent for tasks such as recognizing images of handwritten digits (5)(7). However, for a fully connected network in high dimension (for recognizing higher pixel images), the learning process will be significantly slowed down due to too many parameters. For example, for an input size of 200 x 200, a network with one hidden layer consists of 1000 hidden units (neurons) would have 40 million parameters. Also, the fully connected network ignores the spatial structure of the input image that is important for classification. (7)(8). A solution to it is adding convolutional layers after the input. The network consisting of convolutional layers are called convolutional neural networks. The convolutional neural network can significantly reduce the number of parameters in the network so fasten the learning. Moreover, it keeps the spatial structure of the images and it is invariant to the translation of images. In this paper, we will concisely explain how convolutional neural networks work in terms of the convolutional layers. Also, we will show a problematic convolutional neural network and discuss the causes to it. Two solutions will be given and discussed to solve the problem. We will also investigate how to improve the generalization performance of convolutional networks by using some regularisation methods.

2. Implementing convolutional networks

The convolutional layer has a smaller matrix of dimension smaller than the input matrix with the dimension (H_{input}, W_{input}). It performs a convolutional operation between the matrix and a smaller part of the input matrix, which is the sum of the products of the corresponding elements. One example is given below:



The matrix in the middle of the above example is called kernel, and the dimension of the kernel is (H_k, W_k). The smaller part of the input matrix, squared by the red solid line in the example above, is called the local receptive field. The size (dimension) of the field depends on (same as) the size of the kernel, which is 3x3 in the case. The kernel is also called shared weights because the weights are shared across different local receptive fields during the convolution operation. (7)(5)(8). The layer also has shared bias, which is added to the outputs of every convolution operation. The smaller matrix squared by the red dotted line indicates the

next receptive fields that will perform the convolution with the kernel. The amount of shifting from the previous field is indicated by a term called stride, which is 1 in our case. The result of adding 4x4 output above with bias for each element in the output is called the feature map. The use of local receptive field and kernel reduce the dimension of the input, which significantly decreases the number of parameters as feature maps now represent features of inputs in smaller dimensions.

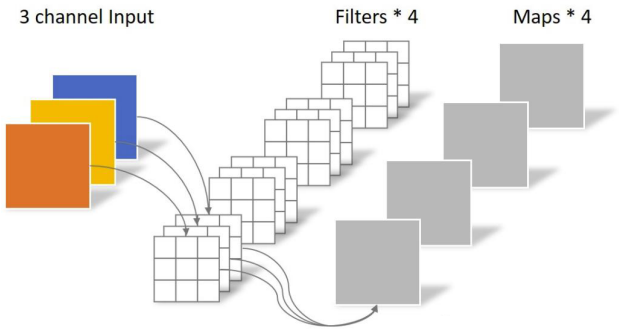
Increasing the kernel size can reduce the input dimension further. However, increasing the kernel size can result in more parameters which are undesired. A method called dilation achieves the effect of increasing the kernel size without increasing the parameters in the network. It enlarges the kernel size by inserting spaces between the kernel elements.

Moreover, a method called polling, that simplify the information in the output from the convolutional layer, is used to reduce the size of the feature map, so it helps to reduce the number of parameters required in later layers.

In our case, the feature map is defined by a set of 3x3 kernel, and a single shared bias (if the bias is used). The example above shows a convolution operation between 2D inputs and 2D kernels. This is not really the case for the convolutional layer in real use, which requires to multiplying 4D tensors (4D inputs and 4D kernels).

It is possible to have multiple feature maps for a single input using a different set of kernels. For example, three feature maps where each feature map is defined by a set of kernels and a single bias. This adds an additional dimension of the kernels for a convolution layer, F_{out}, indicating the number of feature maps.

Moreover, we could have multiple inputs that contribute to each feature map (sum of the outputs):



This adds the fourth dimension to the kernels, F_{in}. Therefore, a convolution layer consists a kernel that has dimension of (F_{out}, F_{in}, H_k, W_k). Similarly, images can come in batches (multiple inputs at a time) which adds the fourth dimension of input, indicates the size of batch. Hence, input has dimension of (BatchSize,F_{in},H_{input},W_{input}). It is nice to spot that the one single feature map has size of ($H_{input} - H_k + 1, W_{input} - W_k + 1$) when stride = 1.

Overall, a convolutional layer in real use has 4D inputs and 4D kernels. In the neural networks, forward propagation maps the output of the neural network from the input by using convolution operation between the input and the kernel and backward propagation helps to optimise the neural network (5)(7)(9). Padding is a method to enlarge the matrix by filling zero terms around the matrix. It is used in the backward propagation.

Both propagations require to multiply the 4D tensors. There are few approaches for implementing the (10) propagation in the term of multiplying tensors:

- Explicitly loop over the dimensions: simple to code, but inefficient in both time efficiency and storage space.
- Serialisation: by replicating the input and kernel matrices, it is possible to convert the 4D multiplication to the large dot product.
- Convolutions: using the explicit convolutional function.

The computation efficient for serialization is remarkably efficient when matrices are very large. Depending on the serialization algorithm, the computation efficient could vary slightly. It has $O(n^{2.3728639})$ time efficiency as published by Le Gall in 2014 (11). However, it is slower than my implementation when the matrices are small, because the reshaping operations would be time-consuming. The storage demands for using serialisation would be large because of the resultant matrix has repeated elements. The computation depends directly on the convolutional function used. Some functions are much efficient than others. For example, PyTorch has a convolutional function that can directly apply to 4D tensors, which is much more computation efficient than the ones cannot.

Under the framework our implementation is made, I used three for-loops and a convolutional (cross-correlation indeed, which is flipped convolution)function to implement both propagations. The computational efficiency is much better than the one explicitly loop over the dimensions but it is less efficient than the serialisation when the dataset is large. My implementation (using loops and convolutional function) has storage demand less than Serialization. For the forward and backward propagation, the computation efficiencies are:

$$T = \text{BatchSize x } F_{out} \text{ x } F_{in} \text{ x } T_{function} + \Theta(1)$$

The storage demands for forward propagation is:

$$S_{forward} = \text{BatchSize x } F_{out} \text{ x } H_{out} \text{ x } W_{out} + \Theta(1)$$

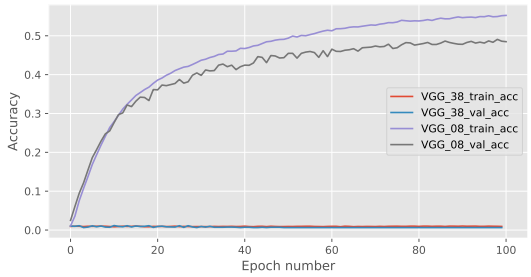
The storage demands for backward propagation is:

$$S_{backward} = \text{BatchSize x } F_{in} \text{ x } H_{in} \text{ x } W_{in} + \Theta(1)$$

3. Optimization problems in convolutional networks

VGG neural networks are the deep convolutional network for object recognition developed and trained by Oxford's renowned Visual Geometry Group (12). We will show a

health VGG convolutional neural network (VGG-08) with 7 convolutional layers plus one fully connected layer, and a problematic VGG network (VGG-38) with 37 convolutional layers plus one fully connected layer. The accuracy on training and validation set (the dataset will be explicitly described later) for both models are shown below:



By intuitions, We would expect the deeper model to be doing better at learning than the shallow ones. However, this is not the case. The accuracy of VGG-38 converges around 0 on both the training and validation set. This is because of the gradient vanishing problem (5)(7)(13).

Gradient vanishing problem arises when the architecture becomes deeper, the derivative terms of the function that are less than 1 or larger than 1 will be multiplied each other many times so that the values will become either very small towards zero hence vanishing or the value tends to infinite large and explode the gradient. This can cause network stopping or be to slow to learn further. The multiplication of derivative terms of the function too many times is one cause to gradient vanishing problem. This can be addressed by ReLU functions(14). However, if the input gets too large or too small, it can also cause gradient vanishing problem since gradient respect to parameters depends on both the derivative term and the input.

3.1. Solution 1: Batch normalisation

One solution to the problem is using batch-normalisation before all activation functions(15).

Batch-normalisation is initially proposed to solve a problem called Internal Co-variate Shift. The shift refers to the change in the distribution of the input values to the intermediate layers, forcing each layer to continuously adapting to its changing inputs. The basic idea behind the normalisation is to normalise the inputs by subtracting the batch mean and dividing by the batch standard deviation, as such that the distribution of input remains fixed over time. This helps to solve the gradient vanishing problems because the distribution of non-linearity input after using batch normalisation remains more stable so the gradient is less likely to vanish as it depends also on the input.

However, after the normalization by some randomly initialized parameters, the weights in the layer are no longer optimal. SGD (Stochastic gradient descent) undoes the normalization if it can minimise the loss function. Therefore, batch normalization adds two trainable parameters to each layer, and the normalised input is multiplied the "standard deviation" (gamma) and adds a "mean" (beta), so the batch normalization allows SGD to perform the de-normalisation by changing only these two parameters for each activation, instead of changing all the weights and hence lose the stability of the network (13).

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

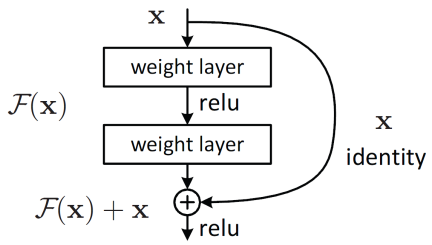
$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.

3.2. Residual Network

An alternative solution to the problem is using residual networks (16).

As we have mentioned above, the deeper neural network should be better at learning than the shallow ones. However, some experiments show that as the network depth increases, accuracy gets saturated and then degrades rapidly (16). Residual networks are initially proposed to solve the problem of degradation. Degradation is not caused by overfitting, but the depth of the deep neural network. A solution to building up a deep neural network without degradation problem is having all layers copied from the shallow network and adding many more identity mapping ($\mathcal{H}(x) = x$) on it. Therefore, it will have accuracy no less than the shallow version. However, many experiments show that it is not possible to find such solution that is comparatively better than the constructed solution in a feasible time, meaning it is very hard to optimize the identity mapping. An alternative solution to it is the residual network (res net). The key idea of the res net is having shortcuts in the network, as the graph is shown below:



The term $\mathcal{F}(x)$ is called residual in res net, such that $\mathcal{F}(x) = \mathcal{H}(x) - x$. The shortcut carries the input x from the layer L^l to the layer L^{l+2} (not necessary L^{l+2}, but L^{l+2} is used for explanation) after the next layer and adds to the output $\mathcal{F}(x)$ from the layer L^{l+1}. It is very difficult to optimize $\mathcal{H}(x)$ but it is much easier to optimise $\mathcal{F}(x)$. Also, if the

identity mapping were optimal, it would be easier to push the residual to zero to fit an identity mapping.

The reason why the res net helps to solve the gradient vanishing problem because of the x term added to the output. During the backward propagation, it can propagate back to a node through shortcuts so the gradient vanishing problem due to chain multiplication could be avoided.

3.3. Compare and contrast of the solutions

Two solutions can both be used to solve the gradient vanishing problem. The key difference between the solutions is that batch normalisation solves the problem by normalising the distribution of the input whereas the res net solves the problem by adding shortcuts between the layers. Both methods are not primarily designed to solve the gradient vanishing problem, where the former is designed to solve the internal co-variant shifting and the latter is designed to solve the problem of degradation. In this work, we will first find out how each solution performs after it solves the problem, in terms of the accuracy on the validation set. We are interested in finding out which solution has better performance. Then, we will combine both solutions to see how it performs, as the solutions do not conflict and can be applied to the network simultaneously. The combined solutions will be used as our baseline model for further improvements since it intuitively should have better performance than a single solution. If it is not the case, the solution with the best accuracy on the validation set will be used as the baseline model.

4. Improving the generalization performance of convolutional networks

Generalization is a term used to describe a model's performance on new data. The model is trained using the training dataset, and validation set is an "unseen" dataset for the model, that is used to test the model's performance after the training and further improve the model. The test set is not used for improving the model because the model should not be improved based on the test set, in order to have the generalization performance on truly unseen data. If the model is able to perform accurate prediction on the training set but cannot predict accurately for new data, then we call the model is overfitting. (5) (7) (17) Overfitting occurs in a model when the accuracy on the training continuously to increase while the accuracy on the validation stops growing or starting to decrease.

Overfitting is undesired because we do not want the model to learn too much about the data itself but recognizing the patterns in the dataset. Therefore, it is important to avoid overfitting to improve generalization performance. There are many regularisation methods can be used to alleviate the overfitting in the models (so improve the generalization performance). Regularisation meaning modifying the model slightly such that the model generalizes better. In this paper, I will introduce three regularisation methods we will use for improving the generalization.

4.1. Data augmentation – Random Erasing

Random erasing is a type of data augmentation, which is a data pre-processing method. It randomly selects a rectangle region in an image and erases its pixel with random values. (18). It is demonstrated in the paper (18) that it actually improves the generalization and alleviates the overfitting problem. In another word, the method works because the model learns dataset with fewer features so it should be able to recognize the data better on validation set when it has more input features, and it reliefs the overfitting problem because it learns less about the data itself, and enforcing the model to learn the pattern of the dataset.

4.2. L2 Regularisation

Regularisation is a way to avoid overfitting by penalizing the large weights parameters. It avoids some parameters get too large so becomes dominant during the learning process. L2 penalty adds a penalty term which is a square of the magnitude of weight coefficients. The values of the weight matrices decrease due to the penalty term as larger weights will increase the cost of the model. A neural network with smaller weights leads to simpler models (5) (19), hence, it will reduce overfitting effect.

4.3. Dropout

Dropout is one method to address the overfitting problem by ignoring randomly chosen neurons during the training phase of a certain set of neurons (20). The units are not considered during a particular forward or backward process. Then the units are recovered and a new set of randomly chosen neurons will be ignored for the next process. It helps to reduce the interdependent learning amongst the neurons (21), and it can be used to alleviate the overfitting problems shown in some experiments (20).

4.4. Compare and contrast between the methods

Three methods are work to help to avoid overfitting in a different part of the neural network. The key differences between the methods are that random erasing works on the dataset of the neural network, the regularisation works on the weight coefficients, and the dropout works on the neurons in the network. The methods intrinsically differ from each other. Therefore, the methods could all be applied to the network to avoid overfitting. We will explore to find out the effects of each regularisation methods.

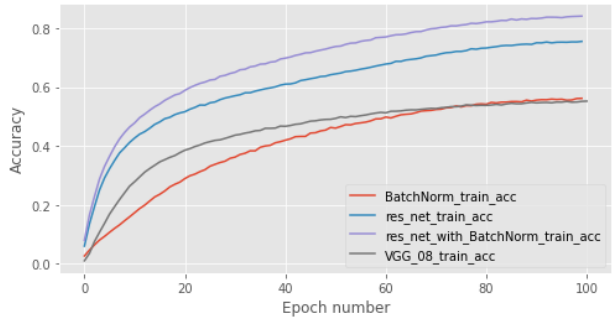
5. Experiments

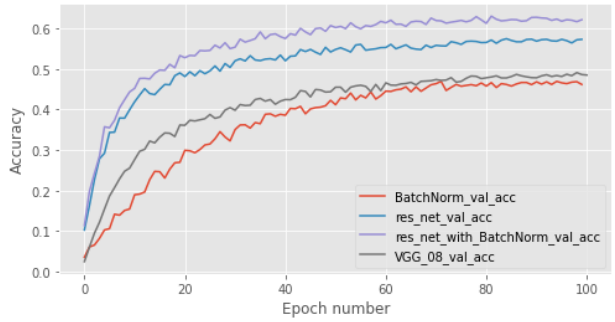
We will carry out our experiments using convolutional neural networks to recognize the colour images on CIFAR100 dataset. CIFAR100 consists of 60000 colour images that each image has a dimension of 32 x 32 x 3. The pixel of the image is 32 x 32 and the colour is defined by RGB (so 3) colours. The images have 100 classes, with 600 images per class. The entire dataset is split into 47500 training, 2500 validation and 10000 testing sets.

The broken VGG-38 network has one entry block that consists a convolutional layer, a batch-normalisation layer (which will be explained later) and a Leaky RelU activation function (22), followed by 3 stages where each stage has 5 convolution blocks which contain 2 convolutional layers and 2 Leaky Relu functions per block. At the end of each stage, it contains a dimension reduction block that has 2 convolutional layers, 2 Leaky ReLU functions, and one average pooling layer (with reducing factor = 2) between the two layers. Each convolutional layer has 32 output channels (F_{out}) and the kernel size of 3 x 3. No dilation is used for the kernel and both strides and padding equal to 1. Our exploration is based on the broken VGG-38 network, which has batch size = 100 and seed = 0, epoch number = 100, Adam learning rule with learning rate 0.001, cosine annealing learning schedule with the minimum learning rate of 0.00002. The learning rule and the learning schedule are state-of-art optimisers for the neural network that enables the network computes individual adaptive learning rate, that helps to fasten the learning (23)

5.1. Solutions to the broken network and the baseline

We carried out the experiments for the batch normalisation, residual network, and the combined one. We are interested in finding out which is the better solution to the broken network. We are also interested to find out a baseline model for our further explorations for generalisation. The results are shown below:





As all the curves converge as epoch number increases, we use the accuracy of epoch 100 as the measurement of the accuracy for the model for convenient, and all accuracy is approximate to 4 decimal place. The result shows that the combined solution has the highest accuracy on both training and validation set (≈0.8410, 0.6212 respectively), as we expected. Interestingly, the single use of batch normalisation (≈ 0.4616)shows that it has slightly lower accuracy than the shallow network (≈ 0.4848). One possible cause to it could be the deep layers network is more complicated than the shallow one, so the model slightly overfits, as indicated in the slow increase of the training accuracy while the validation accuracy has converged. The residual network has much higher accuracy (≈ 0.5728) than the batch normalised one. I think this is expected since the res net is designed and optimised to have accuracy no less than the shallow ones (16). Also, res net, in theory, is more computational efficient and less storage demand than the batch normalisation (BN) since it only adds a short cut which only sums up the input and the residual, where is simpler than the normalisation operation. Apart from the computation efficiency, res net has less storage demand than the BN since res net adds no parameters to the network whereas the BN requires two learnable parameters for each activation layers (15).

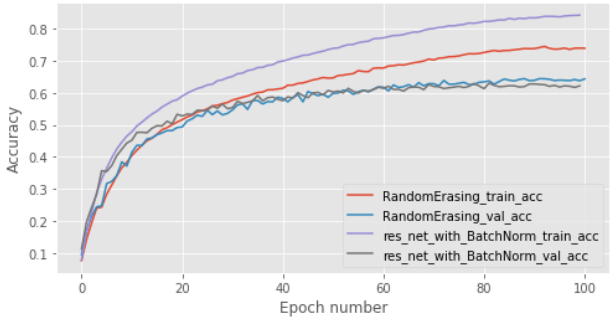
As we have expected, the combined solutions have the best accuracy on the validation set. Therefore we have the res net combined with BN for our baseline model. The test accuracy of the baseline 0.6112. I think this is a good baseline model since the best model of the year 2016 when res net is published has the accuracy of 0.8262, which is a dense network (24).

5.2. Improving the generalisation

From the graph above, there is clear overfitting for our baseline model, since the training accuracy is very high and it is still growing while the validation accuracy has converged. Therefore, we could apply the regularisation to it.

5.2.1. Random Erasing

As we have mentioned above, we want to find out how random erasing can help to avoid the overfitting and improves the generalisation. The random erasing method is implemented with probability p = 0.5 to erase a region. The range of proportion of erased area against the input image is (0.02,0.33) and the range of aspect ratio of the erased area is (0.3,3.3) (25). The result is shown below:

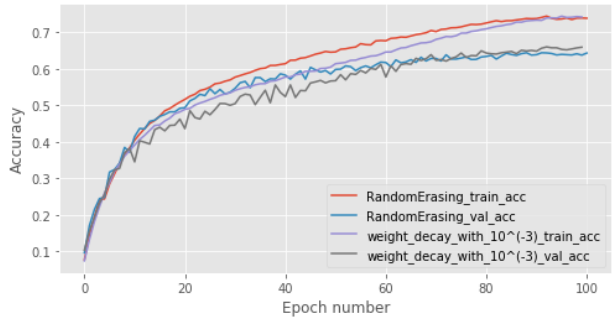


Clearly, the performance of the network on the validation set has improved ≈ 0.6431 compare to ≈ 0.6212. Also, the overfitting has been improved, indicated by the decreasing of the distance between the validation accuracy and training accuracy at epoch 100. The method only adds a small amount of time to pre-process the dataset compare to the baseline. However, the model seems to be possible to make further regularisation. Therefore, we apply another

regularisation method

5.3. L2 regularisation

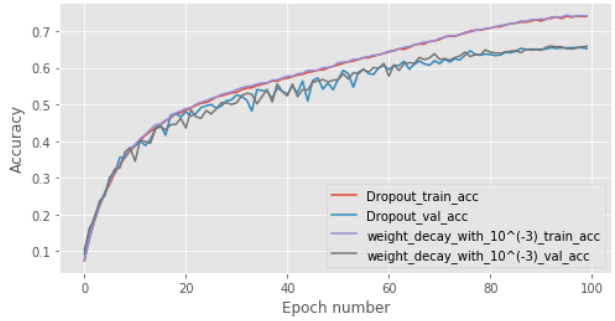
We will find out where the method improves the generalisation on the model with data augmentation. The penalty coefficient we will use is p = 0.001. The value is chosen by my instincts after performing regularisation on other tasks before. The result is shown below:



The graph shows that the validation accuracy again improved slightly ≈ 0.6589. The L2 penalty is added by adding an augment to the optimiser in the network. It should only add small computation time to the network since it penalizes the weight by adding a small term to lost. This computation time is very small compared to the other operations. We now see if the dropout would help to improve the generalization further.

5.4. Dropout

The dropout uses the probability p = 0.5 randomly dropping the neurons in the fully connected. We will find out whether it helps for further improvements. The results are shown below:



Clearly, two curves are very close to each other at the end of the training. The accuracy for the network with dropout implementation has ≈ 0.6524, which is slightly lower than the previous one. Therefore, we decide not to explore further. We will not use the drop out since it, in theory, makes the network slower to learn since some of the neurons are drop out and it is not improving the generalisation.

6. Discussion

The outcomes of the experiments are reasonable. We have found out that res net has much better performance on VGG-38 network, compared to the BN. I think the reason for it is that res net is designed to improve the performance of deep neural network whereas the BN is designed to overcome the co-variate shifting problem. (15) (16) Therefore, res net is more adapted to the framework of our experiments. The choice of the baseline model is the same as we have expected. The res net combined with BN is very powerful as shown in (16). The regularisation methods except for the drop out all improved the generalisation. I think the reason why the dropout did not work is that the network after the two regularisation methods is no longer overfits (or very small overfitting). Also, BN reduces the need for the drop out as mentioned in (15). The methods improved the generalisation because they helped to avoid overfitting, which causes low performance on unseen data. Our final choice of the model is the res net with BN, which applied with random erasing and L2 regularisation methods. The final model has a test accuracy of ≈ 0.6541 which is about 4% higher than the baseline model.

7. Conclusions

Overall, we have learned many things from the experiments. Firstly, the way of implementing the convolutional layer, considering the computing coefficient and storage demands, is much clearer now. Secondly, there will be a gradient vanishing problem and degradation problem in a deep neural network. The batch normalisation and residual network can be used to solve the problems. Finally, the residual network combining with batch normalisation shows large improvements in generalisation for deep neural network, and the generalisation is important for improving the generalisation by avoiding overfitting.

There are a few things I want to further explore. I want to explore the implementation of dense network and compare it to the final model we achieved. This is because dense work is an 'updated' version of the residual network. Also, we spot that information is lost when using convolution to reduce the dimension of input for the residual network. I want to find out a better way to have less information lost during the dimension reduction. Additionally, I want to find out and compare the performance of deep convolutional neural networks with fully connected layer on simpler tasks such as digit classification on MNIST dataset to see if deep convolutional neural networks are necessary for such tasks.

References

[1] What is deep learning and how does it work? | content simplicity, 09 2019.

[2] How deep learning can help prevent financial fraud, 04 2019.

[3] A gentle introduction to the rectified linear unit (relu) for deep learning neural networks, 04 2019.

[4] Skymind. A beginner's guide to neural networks and deep learning, 2019.

[5] Michael A Nielsen. Neural networks and deep learning, 2018.

[6] yaron. 30 amazing applications of deep learning, 03 2017.

[7] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. The Mit Press, 2017.

[8] Poulastya Mukherjee. Convolution neural networks vs fully connected neural networks, 01 2019.

[9] Assaad MOAWAD. Neural networks and backpropagation explained in a simple way, 02 2018.

[10] Hakan Bilen. Convolutional networks 2: Training, deep convolutional networks, 2019.

[11] Thomas Sauerwald. Ii. matrix multiplication, 2016.

[12] VGG. Visual geometry group - university of oxford, 2014.

[13] Chi-Feng Wang. The vanishing gradient problem, 01 2019.

[14] Avinash Sharma V. Understanding activation functions in neural networks, 03 2017.

[15] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift, 2015.

[16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2016.

[17] WordPress for Western. Generalization and overfitting, 01 2017.

[18] Zhun Zhong, Liang Zheng, Guoliang Kang, Shaozi Li, and Yi Yang. Random erasing data augmentation, 11 2017.

[19] Analytics Vidhya. An overview of regularization techniques in deep learning (with python code), 04 2018.

[20] Journal of Machine Learning Research. Dropout: A simple way to prevent neural networks fromoverfitting, 06 2014.

[21] Amar Budhiraja. Learning less to learn better—dropout in (deep) machine learning, 12 2016.

[22] ML. Activation functions — ml cheatsheet documentation, 2011.

[23] Vitaly Bushaev. Adam—latest trends in deep learning optimization., 10 2018.

[24] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Weinberger. Densely connected convolutional networks, 2017.

[25] pytorch. Pytorch documentation — pytorch master documentation, 2019.