CIFAR-10 Image Classification using PyTorch

Group 8: Mircea Zisu, Matt Cooper, Rebecca Wardle, William Griffiths, Zijun Zou, Wing Ka Lin

***Abstract*—Our task is to create an artificial deep neural network for classifying images in the CIFAR-10 dataset. We used the PyTorch machine learning framework to create a convolutional neural network. Starting with a basic convolutional neural network, we modified and improved it using error analysis until we reached our final network. We successfully trained our network to classify images from the CIFAR-10 dataset with an error rate of 14.35%.**

I. Introduction

IMAGE classification is the problem of assigning a single class of a fixed set of classes to a given image. Contextually, the image classification problem is core to many areas of artificial intelligence integration and the key to computer vision. Examples include medical imagery, object recognition, and robotics. Hence, successful classifiers have many practical uses and it is necessary for them to have a high accuracy.

The classical approach to this problem is through linear regression or classification based neural networks (NN), with networks such as the Multi-layer Perceptron. However, modern techniques have largely shifted towards convolutional neural networks (cNN), as they are consistently more accurate and more proficient than other techniques.

For our group project, we have been tasked to create a neural network classifier to solve the problem of image classification to a sufficient degree of accuracy. Per the assessment specification, we will be using the PyTorch framework and exclusively using the CIFAR-10 dataset.

II. Methodology

We have approached the problem by creating a convolutional neural network using PyTorch. By taking inspiration from different existing networks, our team has created a network with 1) 7 2D convolution layers, 2) 3 fully-connected layers, 3) 2 Dropout 2D layers, 4) 2 Dropout 1D layers, 5) 2 Max Pooling layers, 6) 6 Batch Normalisation layers.

The network is trained through an iterative process. The dataset is split into batches of 64. In the training loop, the network first iterates through the batch, and classify the images. The prediction is then passed into the loss function, along with the ground truth, to calculate the loss. The optimiser has its gradients set to 0, and the error is back-propagated by calling $\text{loss}.\text{backward}()$. $\text{Optimiser}.\text{step}()$ is then called to update all the tensors (parameters).

Convolution layers extract low to high level features of the original image by convolving a specified filter across the multi-channelled image. This is done through the convolution of the filter and all spatial locations of the image, resulting in a feature map. As the input image is composed of multiple channels, the filter's depth must directly reflect the number of channels. Also, we have chosen to stack convolution layers together as they enable the network to select more complex features of the image progressively. Overall, convolutional layers significantly reduce the complexity of the network through the feature mapping. However, they can become resource-heavy and so it is important to choose effective but efficient parameters. [1]

LeakyReLU activation function is used for the convolution and fully-connected layers. The rationale behind this instead of the standard rectified linear unit (ReLU) function is to mitigate the dying neuron problem. LeakyReLU slightly increases the gradient of the horizontal line when $x < 0$. The default gradient in PyTorch is $y = 0.01x$. The positive gradient allows neurons stuck in the negative stage to recover, and not remain stuck outputting 0. [2]

Regularisation is needed to prevent the model from overfitting. Dropout and Dropout2d layers are used as the regularisation technique. Dropout stochastically zeroes out some neurons, helping prevent overfitting by reducing the total weights, minimising the effect on the loss function. [3] Dropout2d stochastically zeroes out entire filters. Dropout2d layer is placed after the 4th and the final convolution block, with $p = 0.48$, while Dropout is placed after the 3rd convolution layer ($p = 0.35$) and the 1st fully-connected layer ($p = 0.4$). The efficacy is supported widely in the machine learning community. [4] [5]

Batch normalisation is similar to how data is preprocessed and normalised before training – normalise the mean and variance of the output of a convolution layer. The rationale behind batch normalisation is to counteract with the large variation (the distribution) in the output value of the convolution layer – "internal co-variate shift". [6] Batch normalisation is applied after every convolution & activation block.

The data is downsampled using the max pooling layer. Max pooling was chosen over average pooling as max pooling downsamples and still retains important information, such as edges and textures. Different kernel size is chosen based on the position of the pooling layer in the pipeline. Our network uses a (2×2) kernel size after the 4th convolution layer, and a (3×2) kernel size at the final convolution layer. The intuition behind the placement is to reduce the number of parameters while retaining as much information for the network as possible.

Cross-entropy loss function – a combination of the negative log likelihood (NLL) loss function and the LogSoftmax activation function – was used. As the accuracy of the network's guesses increase, the value output by the loss function decreases, tending to 0.

$$\text{loss}(x, \textit{label}) = -\log\left(\frac{e^{x[\textit{label}]}}{\Sigma_j \exp(x[j])}\right) \tag{1}$$

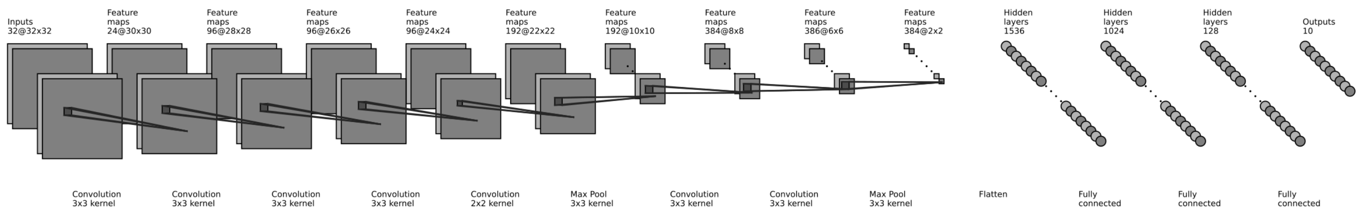

Figure 1. A diagram of the network

III. Results & Evaluation

Our team evaluates our model based on: 1) Time taken per iteration of training, 2) Time taken for model's accuracy to stabilise after n iterations, 3) Error rate of the network, and 4) Number of parameters in the network.

To evaluate our model, the model is set into evaluation mode, where batch normalisation and dropout layers are turned off (as these layers interfere with the evaluation result). The CIFAR-10 testing data is passed into the model, and the model outputs a prediction, which is passed into the loss function – similar to how the training loop works. There is no backpropagation which means that the test data is not used for training the model at all, and is only used for evaluation of the network. –The correct rate and the loss rate are accumulated and calculated. These metrics are used to evaluate the model's performance. With our model, we have achieved 85.65% accuracy rate (14.35% error rate) in 2023 seconds.

The network was trained for less than 1 hour on a system with NVidia GeForce GTX 1070 with 8GB of memory, AMD Ryzen 5 3600, and 16 GB DDR4 RAM.

Stochastic gradient descent (*SGD*) is used as the model optimiser. Adaptive momentum (Adam) algorithm was trialled as an alternative initially. When implemented, the error rate would fluctuate aggressively and the accuracy would struggle to exceed 80%. Adam, despite having a faster convergence, fell short of the generalisation of SGD, and therefore SGD was chosen. This result is further supported by other researches [7], where Adam generalises worse than SGD.

A learning rate (LR) of 0.01 was chosen initially, as this provided rapid optimisation, achieving an error rate of around 23% within 30 iterations of training. Running the programme further however showed little to no improvement. In an attempt to improve accuracy, we reloaded previously trained networks with LR of 0.001 or less and found that they learnt well up to an error rate of 20%. With this knowledge in hand, we looked into variable LR algorithms supported by PyTorch.

Reduced LR on plateau algorithm (*ReduceLROnPlateau*) is provided by PyTorch, and is used to schedule the LR. The algorithm requires some measurement value to be passed into as an indicator. The algorithm reduces the LR by a factor of 10 after a specified number of iterations over which the model has not improved (a "patience"), we found through testing that a "patience" value of 10 provided the best overall learning rate for our networks.

Initially, our network did not have any regularisation. The accuracy was consistently under 70%. Introducing dropout and dropout2d improved our network performance to above 70%. Introducing 1D dropout in between convolution layers further increased the performance to 80% and above.

Over the course of running at least 70 different variations of our network, with different parameters such as dropout probability, number of channels, pooling kernel size, and the addition and removal of layers, we saw gradual but definite improvements in our network's accuracy rate. The result of those trials are that most of them have at least 80% accuracy rate.

IV. Conclusion & Further work

We have implemented a model that achieves above 85%, well within our expectations. However, our network is far from state of the art, compared to high-performing well-designed networks such as ResNet, VGG16, DLA, etc.

Overall, the architecture is a good choice. We did not brute-force the problem by adding more and more layers and increasing the filter count, nor did we specialise the network too much based on the evaluation result. Similarly, we also did not adapt other network types such as recurrent NN or support vector machines.

Setting up the environment and producing tangible results were achieved within a couple of days, which allowed our team to quickly start experimenting and evolving our network structure. We failed to lay starting frameworks which would allow us to properly organise everyone's experiment and contributions to the code base. Research around visualisation and tooling lagged behind experimentation so we could not take full advantage of the tools available to us.

To improve our network, we could introduce more 1D dropout layers in between the convolution layers, and see if the regularisation could improve the performance further.

References

[1] K. O'Shea and R. Nash, "An introduction to convolutional neural networks," 2015.

[2] B. Xu, N. Wang, T. Chen, and M. Li, "Empirical evaluation of rectified activations in convolutional network," 2015.

[3] baeldung. How relu and dropout layers work in cnns. [Online]. Available: https://www.baeldung.com/cs/ml-relu-dropout-layers

[4] J. Tompson, R. Goroshin, A. Jain, Y. LeCun, and C. Bregler, "Efficient object localization using convolutional networks," 2015.

[5] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," 2012.

[6] S. Santurkar, D. Tsipras, A. Ilyas, and A. Madry, "How does batch normalization help optimization?" 2019.

[7] P. Zhou, J. Feng, C. Ma, C. Xiong, S. HOI, and W. E, "Towards theoretically understanding why sgd generalizes better than adam in deep learning," 2020.