Classification of the CIFAR-10 Dataset

Group 27: Finley Brown, Alec Coates, Samuel Haley, Danny Hendron, Benjamin Jenner, Luke Roberts

***Abstract*—We were tasked with training a deep neural network on the 50,000 CIFAR-10 training images, in order to then classify the 10,000 CIFAR-10 testing images into 10 different classes. We tackled this by creating a convolutional neural network using PyTorch. The neural network we constructed consists of 10 convolutional layers, 3 max-pooling layers, and 3 fully-connected layers. Our network achieved a classification accuracy of 90.6%, making an error rate of 9.4%.**

I. Introduction

IMAGE classification refers to the task of extracting and analysing information from an image to predict it as belonging to a certain class. We are using the CIFAR-10 dataset to construct and train a neural network to accurately identify and classify a given image into defined categories. Image classification has been around since at least 1963 [1] and the problem has been tackled in multiple ways; for example, David Marr developed an early version of image classification which would detect simpler shapes and then use these to build up from a 2D image to a 3D one [2]. Nowadays, image classification often makes use of convolutional neural networks (CNNs). An early form of this was the Neocognitron by Kunihiko Fukushima [3], a self organizing artificial network of simple and complex cells which was unaffected by position shifts and included several convolutional layers.

This field has widespread use in medicine, security, intelligent transportation systems, and various other areas [4]. For example, deep neural networks are used in medical applications to find tumours and determine whether they are malignant or not. Of course, achieving the highest accuracy possible in these classifications is critical.

CNNs are now the standard method in image classification problems, as they consistently perform better and achieve more accurate results than other types of networks or other approaches to the same problem. This is why we chose to implement a CNN to solve the given classification task.

II. Method

The network we have chosen to create has the following architecture, as shown in figure 1. Our network is a 13 layer model, starting with 10 convolutional (conv) layers, followed by 3 fully-connected (FC) layers. The convolutional layers consist of 4 sections, each separated by a 2D max-pooling layer. Each max-pooling layer has a size of 2, a stride of 2 and no padding.

A convolution is where a filter is applied to parts of an image sequentially to detect features in the image; because convolutional filters maintain a relationship between dimensions, they are much better suited to this task than FC layers. Often used in conjunction with conv layers, max-pooling layers are responsible for reducing the spatial size of images by picking the most active neuron from a pool of neurons and discarding the rest; this has the effect of keeping the most important features of an image. By reducing the layer size, pooling layers also reduce the computing power needed to process a layer, and so we thought they would be a useful addition to our network.

The first layer takes in the input 32x32 image with 3 channels, for the RGB channels in an image, and outputs 64 channels, which the next layer takes as an input. The second layer has 64 output channels. The image then goes through a max-pooling layer, reducing the image to 16x16. The third layer has 64 input channels and outputs 128 channels. The next two layers both keep the output at 128 channels. There is then another max-pooling layer, making the image 8x8. The sixth layer then takes in the 128 channels and outputs 256 channels, and the next two layers also output 256 channels. We then have the final max-pooling layer, making the image 4x4. The ninth layer takes 256 input channels and outputs 512 channels, which the next layer takes in and then also outputs 512 channels.

All of our conv layers are 2D batch-normalised, activated with Leaky ReLU, and have a 0.2 dropout applied to them before the output is passed on to the next layer.

Leaky ReLU is a linear activation function based off the popular ReLU function. ReLU is popular because it prevents the function's gradient from becoming too low as it would when reaching an extreme of the sigmoid function. It is also very computationally efficient to calculate. The Leaky variant of ReLU adds a small gradient for inputs below zero, which could otherwise also cause some regular ReLU neurons to 'die' by zeroing the gradient.

The last 3 layers of our NN are fully-connected. We used FC layers as they are good at matching up features from the conv layers to an output class.

Our first FC layer takes in the flattened output of the last conv layer as 8192 input nodes and 512 output nodes. The second FC layer takes in the 512 nodes and also has 512 output nodes. The final FC layer takes in those 512 nodes and has 10 output nodes, one for each classification label.

The first two fully-connected layers are activated using the Leaky ReLU function before the output is passed on to the next layer. The output of the last layer is activated with a 10-way Log Softmax (see equation 2), which is used to calculate which of the 10 output labels, the starting image has the highest probability of belonging to.

We trained the network using the Cross-Entropy Loss function (see equation 1), as it's well suited to multi-class classification problems because it gives improvements to the worst-performing classes, over the best-performing classes, encouraging well-rounded models. We also used the AdamW optimizer algorithm with learning rate decay. We experimented

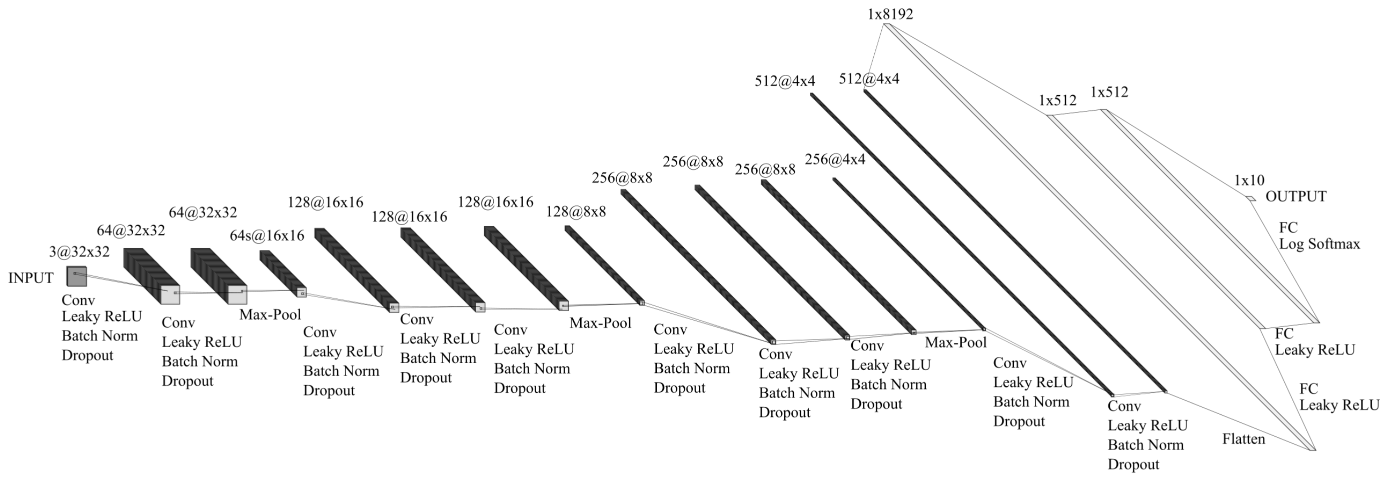
INPUT
3@32x32
64@32x32
64@32x32
64s@16x16
128@16x16
128@16x16
128@16x16
128@8x8
256@8x8
256@8x8
256@8x8
256@4x4
512@4x4
512@4x4
1x8192
1x512
1x512
1x10
OUTPUT
Conv Leaky ReLU Batch Norm Dropout
Conv Leaky ReLU Batch Norm Dropout
Max-Pool
Conv Leaky ReLU Batch Norm Dropout
Conv Leaky ReLU Batch Norm Dropout
Conv Leaky ReLU Batch Norm Dropout
Max-Pool
Conv Leaky ReLU Batch Norm Dropout
Conv Leaky ReLU Batch Norm Dropout
Conv Leaky ReLU Batch Norm Dropout
Max-Pool
Conv Leaky ReLU Batch Norm Dropout
Conv Leaky ReLU Batch Norm Dropout
Flatten
FC Leaky ReLU
FC Leaky ReLU
FC Log Softmax

Fig. 1. Network architecture

with several optimizer algorithms and found AdamW to be one of the fastest converging optimiser. A comparison between AdamW and SGD (with momentum) is described in section III and table I.

The formula for Cross-Entropy Loss:

$$L(y_i, \hat{y_i}) = -\frac{1}{M}\sum_{i=1}^{M} y_i \cdot log(\hat{y_i}) \tag{1}$$

The formula for Log Softmax:

$$LogSoftmax(x_i) = x_i - log(\sum_{j=1}^{K} e^{x_j}) \tag{2}$$

III. Results And Evaluation

TABLE I
Optimizer functions

	SGD (%)	ADAMW (%)	Change (%)
Plane	87.7	90.4	+2.7
Car	94.6	96.9	+2.3
Bird	79.6	87.1	+7.5
Cat	59.8	75.7	+15.9
Deer	83.3	89.7	+6.4
Dog	83.1	90.1	+7.0
Frog	93.9	94.8	+0.9
Horse	90.3	94.0	+3.7
Ship	93.8	95.1	+1.3
Truck	90.7	92.2	+1.5
Total	**85.7**	**90.6**	**+4.9**

Initially, we trained our model using Stochastic Gradient Descent over the course of 100 epochs. Our starting learning rate was set at 0.0003 and decayed by 90% every 40 epochs, while our weight decay was set at 0.0015. After researching other optimisation algorithms, we switched to the AdamW algorithm, and gave it the same learning rate and weight decay as before. When experimenting with a lower number of epochs, AdamW performed better than any other algorithm, and so we chose that as our final optimisation algorithm. It is still possible that a different algorithm might have performed better given a longer training time, but we lacked the computational capacity to fully test that within the time limit.

The network was trained using CUDA on a Nvidia 1660TI, taking an average of 100 seconds per epoch; just over 8 hours for 300 epochs of training.

We decided to evaluate the model using the Rank-1 accuracy metric, as described in the project brief. That is, the accuracy score in the table is just “What percentage of images are classified into the correct class?” The CIFAR-10 training set was exclusively used to train the neural network, with the test error being computed purely from the test data set. When retesting our model on the training data it had an accuracy of 98.1%, showing that there has been some over-fitting, as this is significantly higher than the accuracy on the testing data.

Using the AdamW optimiser algorithm, we achieved an overall accuracy of 90.6%, which is an increase of 4.9% (see table I) compared to SGD.

IV. Conclusion And Further Work

In conclusion, the overall performance of our classifier was quite good with an average accuracy of 90.6% using the AdamW optimizer. Our architecture is a good choice, considering convolutional neural networks are now the go-to method for classifying images. We are proud of our neural network’s ability to classify most classes. For example, it is particularly accurate at classifying cars, with a 97% accuracy.

One area for further work would be improving the network’s accuracy when it comes to classifying cats, which currently only has an accuracy of 76%. To do this we might consider tweaking the overall architecture of the network; for instance, switching to a hierarchical classification model could produce better results by lowering the loss in cases where the model misclassifies images as a similar class e.g. cats and dogs.

References

[1] L. G. Roberts, ”Machine Perception of Three-Dimensional Solids”, Ph.D dissertation, Dept. Elect. Eng., MIT, 1963.

[2] D. Marr, *Vision: A computational investigation into the human representation and processing of visual information*, San Francisco: W. H. Freeman, 1982.

[3] K. Fukushima, ”Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position”, *Biological Cybernetics*, vol. 36, no. 4, pp. 193-202, 1980.

[4] B. Javidi, *Image Recognition and Classification: Algorithms, Systems, and Applications*, New York: Marcel Dekker, 2002.