INT2 Group Assessment

Group 16: Pratham Bhat, John Cherry, Benji Garment, David Kelly, Lewis McShane, Jiacheng Wu

***Abstract*—This report discusses how group 16 were tasked with implementing a neural network to analyse 3-layer (RGB) images and return a categorisation for each given image. Group 16 achieved this by creating a model based on the VGG-16 architecture, and experimenting with different hyper-parameters to tune the model. Through the creation of this model and subsequent tuning, the group's final classification error on the CIFAR-10 dataset was 7.8%.**

I. Introduction

This project looked to classify a large dataset of images known as the CIFAR-10 dataset by creating a Convolutional Neural Network (CNN) in Python. The dataset is made up of 60,000 32 pixels by 32 pixels three colour (Red, Green, Blue) images which are split into ten categories. The dataset was created by Alex Krizhevsky et al. [1] and is a subset of a larger (80 million images) dataset. The dataset was created in 2009 and has had numerous research papers that have analysed and categorised the data to varying levels of accuracy. Examples of other classification projects include [2], a paper by the dataset creator, which has an accuracy of 78.9% and more recently in 2017 [3] by DeVries and Taylor which had an accuracy of 95.5%. The paper used a CNN and discussed the problem of overfitting individual filters to specific images. The team reduced the error rate caused by overfitting by blocking out sections of the training images, which significantly improved the accuracy of their model. This shows their model should focus on the test data images as formatting them in a certain way may affect the accuracy of the final model.

This project is a valid machine learning problem as it has real-world applications for use. Image recognition is used in a range of situations such as Google's reCAPTCHA project [4] which uses image classification to verify if a website visitor is a human.

We tackled this problem of image classification by creating a CNN using PyTorch. The decision to use PyTorch came from the group's familiarity with the Python programming language as Pytorch was specifically designed to be used in Python unlike its alternative Tensorflow, which was designed to be used with multiple languages. A lot of research was done to ensure that our network was able to accurately classify images. This involved looking at many pre-existing model architectures as inspiration and then exploring these designs. In the end after looking over our research we decided that a VGG-16 CNN was the best-suited model to this task and created our own through PyTorch.

II. Method

The architecture our model is based on is the VGG-16 Convolutional Network [5]. The VGG-16 can be split into different blocks consisting of 13 convolutional layers and 3 fully connected layers, with each block being connected by a max pooling layer and a ReLU activation function. The structure of these layers can be seen in Fig 1, where the first input layer is at the top, and the final classification is done at the bottom with the softmax layer.

The reason for choosing VGG over a technique such as ResNET was due to the fact that VGG has a very modular architecture. It uses different blocks of similar layers to build up a deep neural network.

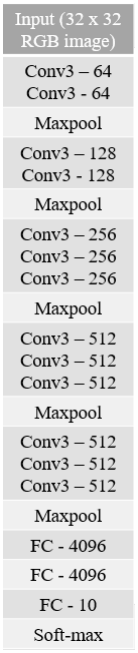

Fig. 1. An Overview of the final model

The input size of the first convolutional layer is an RGB image of size 32x32 (So a total dimension of 32x32x3). Once in the network, the image is passed through 13 convolutional layers, each using a filter of size 3x3 with a stride of 1 and a padding of 1. After a block of convolutional layers comes a MaxPooling layer, which has a 2x2 filter (kernel size) and a stride of 2. After all the pooling and convolutional layers, there are 3 full connected layers. From the starting dimensions and due to the pooling, the first two fully connected layers have 4096 channels each. The final fully connected layer has 10 channels, as it needs to be able to classify 10 types of objects. Finally, the output of the final fully connected layer is put through the softmax function in order to provide a better classification probability. The activation function used across the hidden layers was the same and was the ReLU function, and each hidden layer also made use of Batch Normalisation.

The Cross Entropy Loss function was chosen due to its good performance in both speed and accuracy performance metrics, which has the following formula:

$$L(\hat{y}, y) = -\sum_{k}^{K} y^{(k)} \log \hat{y}^{(k)}$$

Fig. 2. Categorical Cross-Entropy Loss Formula

For optimisation algorithms, we had the choice of using the Stochastic Gradient Descent Algorithm (SGD) and the ADAM algorithm. Although ADAM was the faster optimisation algorithm during testing, SGD was able to generalise better [6], which was important given the aforementioned issue of overfitting. As the time limit for training a model was 24 hours, the loss in speed was not a concern. With the setup as described, we used the following approach to train and validate the model:

Firstly, the dataset was loaded in a batch size of 32 and split into training and testing data. Then for every batch in the training set, the optimiser and then the loss function were applied to calculate the current loss of the model against the training data. Once the data has been optimised, the accuracy and loss of the current state of the model was validated against the testing data, which was then output to the console. This was repeated for 200 epochs.

III. Results & Evaluation

TABLE I
STATISTICS FOR THE FOUR MODELS

CNN	Conv. Layers	Error	Epoch	Training Time
Model 1	2	34%	200	2h 10m
Model 2	8	13.6%	200	2h 30m
VGG	13	8.2%	200	3h 15m
VGG + decay	13	7.8%	200	3h 14m

Throughout the assessment, we had three major models with which we experimented with.

Our first model was based on the PyTorch Convolutional Network tutorial [7], and consisted of two convolutional layers and three fully connected layers. For each epoch, we ran both the training and validation and printed the accuracy and loss values of the validation dataset.

With Model 1, the initial accuracy started around 33% and reached a maximum of 66% accuracy at around 150 epochs, after which it did not improve substantially. We allowed the model to continue training until 200 epochs to ensure the accuracy had converged. This first model showed that with any model there was an epoch at which no matter the time invested into the training, there would be no improvement. We also noticed that the training started with jumps in accuracy of 10-20%, but quickly slowed down to improvements as little as 0.1% between epochs.

Using Model 1 as a base point, Model 2 had eight convolutional layers, three max pool layers and two fully connected layers. With this model it seemed to have the same performance, however, due to being a deeper network, it plateaued at a higher accuracy of 86.4%.

VGG had the most modifications made to it due to its great potential. This ended up being the model being used as our final version. Initially, this model was based on the VGG-16 architecture with changes made to make it work with CIFAR-10. The first run of this model resulted in a high of 89% accuracy, which was a minor improvement over the previous model. The improvement in accuracy from Model 2 was lesser than the difference in accuracy between Model 1 and 2, which indicated that from this point, adding more layers would add little to the accuracy of the final model.

As a result of this, the changes made to get the final accuracy of 92.2% were to the training process. To improve the accuracy there were experiments with extreme batch sizes, both large and small, as well as introducing learning rate decay. These two changes resulted in an accuracy of 91.1%. During training, the training accuracy would quickly reach values of 98-99% which indicated that the model was overfitting. Because of this, weight decay was introduced, which is what allowed the model to reach an overall accuracy of 92.2%.

The method of training and validating the model is the same as that described in the Method section, which was used to train and validate the final model, the results of which can be seen in Table 1.

IV. Conclusion & Further work

The main observation made while researching different Convolutional Networks is that increasing the number of convolutional layers has a general improvement to accuracy. However, in the case of CIFAR-10, which is a relatively small dataset, there is the issue of overfitting.

The final model which used the VGG-16 architecture was able to achieve an accuracy of 92.2%. Given the time scale of this project, this is a high accuracy to achieve and is approaching the upper limits of what can be achieved with the VGG-16 architecture [8]. This architecture was a good choice as it was able to achieve this accuracy of 92.2% in less than 4 hours of training and within 200 epochs.

One of the limitations of the VGG-16 network was overfitting. When there were not enough transforms or the correct hyperparameters set, the network would reach a high training accuracy even before the validation accuracy was at an acceptable level, which indicates overfitting. We were able to overcome some of this by making use of more randomising transforms and weight decay, however, the issue was still present to some extent. Given more time to explore this network, the next iteration of the model would look to get better values for the weight decay as well as adding transforms that crop or remove sections of training images.

References

[1] Alex Krizhevsky. Learning multiple layers of features from tiny images. *University of Toronto*, 05 2012.
[2] Alex Krizhevsky. Convolutional deep belief networks on cifar-10, 2010.
[3] Terrance Devries and Graham W. Taylor. Improved regularization of convolutional neural networks with cutout. *CoRR*, abs/1708.04552, 2017.
[4] "about recaptcha" [online] accessed 3/5/2021. https://www.google.com/recaptcha/about/.
[5] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*, 2015.
[6] Nitish Shirish Keskar and Richard Socher. Improving generalization performance by switching from adam to SGD. *CoRR*, abs/1712.07628, 2017.
[7] "training a dataset" [online] accessed 3/5/2021. https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html.
[8] Felipe O. Giuste and Juan C. Vizcarra. CIFAR-10 image classification using feature ensembles. *CoRR*, abs/2002.03846, 2020.