CIFAR-10 Convolutional Neural Network Classifier

Group 14: Natalia Pilarczyk, Harsh Mohan, Joel Beedle, Amy Mason, Harry Moore

***Abstract*—The aim of our project was to train a machine learning model from an image dataset that can classify contents of a given 2D input image and assign it to a set of given categories. We designed the neural network architecture using TensorFlow and trained the model in Python using the CIFAR-10 dataset. Our achieved validation accuracy was $\approx 93.0\%$, meaning our classification error on the CIFAR-10 test set was $\lessapprox 7.0\%$.**

I. Introduction

The task at hand can be justified as image classification - whereby the machine is given a set of visual inputs, in the form of 2D images, and then labels the inputs appropriately. The objective: to train a machine to be able to correctly assign a predefined category to a given 2D input. Image classification is not only constrained to our dataset as it can be performed on multiple different sources and their relevant instances. For example, data is acquired from a range of sensors and then classified it using supervising techniques or categorising acquired data using pixel basis and assigning each one with their relative information remotely.[3]

Initially, when image classification was being developed, the methods of training the machine were slightly different to what we use nowadays. Pioneering work on this topic was looked into and developed by Yann Le Cun in 1989, who created convolutional neural network diagrams. He laid out how the images can be divided into sections and as the input is being diagnosed the machine focuses on more prominent features as it proceeds and, as it learns to recognise the details, uses back-propagation to assign classes to the samples. Between then and now, the networks and machine learning approaches have changed and adapted by introducing new layouts, making the networks deeper, and introducing inception, where the layers are connected to allow multi-feature execution.[2]

We can see image classification as an important machine learning problem which has been researched and advanced since the late 1980s. One of the most significant networks that has contributed to the research and architecture of image classification has been, and still is, Convolutional Neural Networks (CNN). Not only is it still used by computer scientists but has also inspired many new variations of itself and, thus, we are going to be using CNN in TensorFlow.

II. Method

The **CIFAR-10** dataset had 6000 images, that we split into 5000 for training and 1000 for validation, each with the size (32, 32, 3) - dimensions of 32x32 pixels, each pixel has 3 channels of RGB ranging from 0 to 255. The model keeps the 3 channels to extract more data, resulting in us being able to get more accuracy and better overall result. As for the labels, which assign class to an image from 0 to 9, we have done one-hot encoding representing ten different classes. Each one-hot vector can be thought as a probability distribution, meaning as the model learns to predict the values it will also output a probability that the example given belongs to any of the classes.

Our CNN consists of following layers: the input, Conv2D filters, Activation(Relu and Softmax), BatchNormalization, MaxPool2D, Dropout, Flatten filter, and Dense. The Conv2D filter, when applied to our image, attempts to extract some similarities and produces a tensor output, followed by Relu Activation layer which speeds up time by preventing exponential growth in the computation required to operate the CNN. Then BatchNormalization applies a transformation that maintains the mean output close to 0 and its standard deviation close to 1, afterwards MaxPool2D layer down-samples the input along its height and width by taking the maximum value over an input window for each of the all channels, the window is shifted by 2 along each dimension. We apply a combination of Conv2D, Relu Activation, BatchNormalization, and MaxPool2D to go deeper into the image and extract the patterns within it. Once the image becomes 8x8 pixels we extract the pattern using a combination of Conv2D, Relu Activation, and BatchNormalization, however this time they are followed by Conv2D and Dropout instead of MaxPool2D. So, instead of going deeper into the image, we keep and then drop some extracted features to avoid overfitting our model. Using those features we kept, we apply Flatten, BatchNormalization, and Dropout to make a list of all features and again create random noises. Finally, we apply bottleneck and gradually decrease our features from 24576 to a list of 1024 features using Dense, followed by one more Dense to make our feature list 512 and get closer to our targeted 10 features. Lastly we apply another set of BatchNormalization, Dropout to further prevent overfitting and then decrease to a list of 10 features, equivalent to the one-hot label vector.

The nature of our problem is multi-class classification, meaning there are more than two exclusive targets. Due to that, we have used categorical cross-entropy loss function on our training images and one-hot encoded labels to calculate the loss of an example, as it is a good measure of how much two discrete distributions, i.e our output prediction and our target, are distinguishable from each other. The function is given by:

$$Loss = -\sum_{i=1}^{output\ size} y_i \cdot log\ \hat{y}_i$$

where $\hat{y_i}$ is the i-th scalar value in the model output, y_i is the corresponding target value, and output size is the number of scalar values in the model output. The minus sign ensures that the loss gets smaller when the distributions get closer to each other.

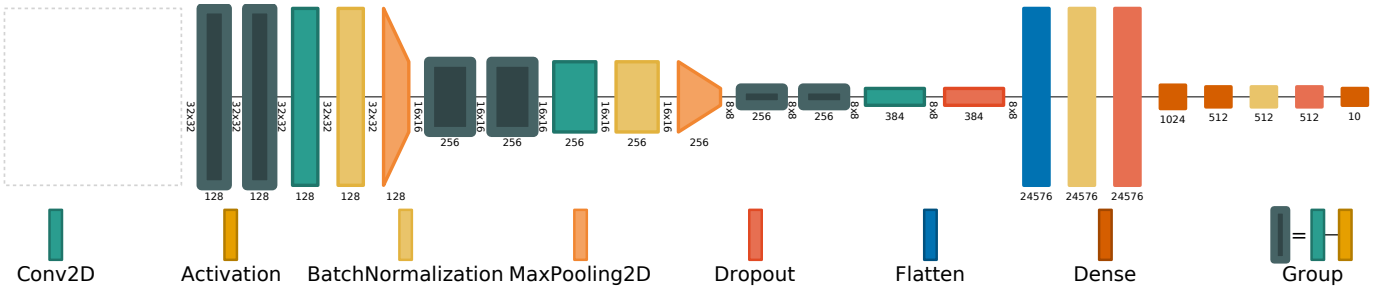
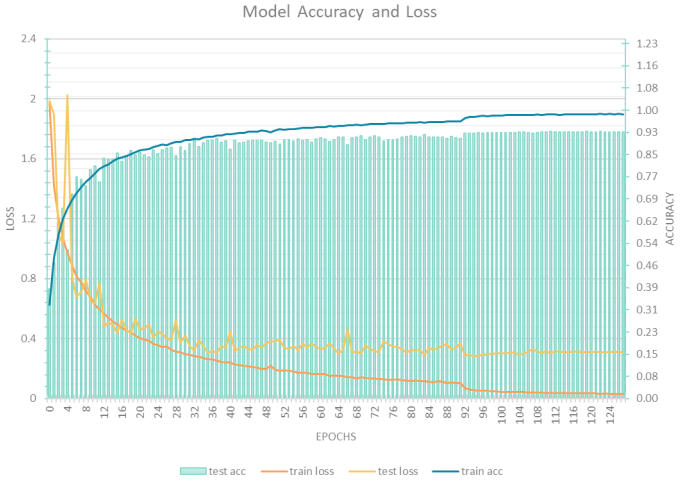


Fig. 1. We produced a model of the network architecture using Net2Vis [1], detailing each step of the neural network and the operations done in order to classify the data. In the Model, dark grey blocks are group of Conv2D, Relu and BatchNormalisation and the final layer is followed by Softmax.

III. Diagram of Network Architecture

Fig. 1. Visualizes our CNN Architecture in detail.

IV. Results & Evaluation



We approached the task by making one solid convolutional neural network model, and then derived numerous models with different combination of layers, numbers of filters, and neurons. We trained all these models for low epochs,compared patterns, and then picked the most promising model.

We got 89.5% accuracy when we ran the selected model on 100 epoch, which revealed a promising result as we've had a lot of things to try on the model to boost it's accuracy. We started with light data augmentation that included - $\pm$ 20 degrees rotation, 10% width and height shift range (3.2 pixels in our case), 10% zoom range, and a horizontal flip, using a random combination of all. This gave us a dramatic boost in accuracy, but we did not want to run our model for arbitrary epochs, thus we set up early stopping with dynamic hyperparameters tuning, based on reduce learning rate when a loss has stopped improving. The combination of both and Adam optimizer is very powerful, since we can put the model on train with very high learning rate and then the machine will automatically change the learning rate when it is nearer to the minimum whilst, also, automatically stop the training once it sees no improvement.

The evaluation matrices we used was categorical accuracy which matches often predictions match one-hot labels. The graph above shows both accuracy and loss of model training and validation performed on only encoded **CIFAR-10** dataset. It is quite clear that we have reached a plateau at $\sim 93.0\%$ and our model stopped training due to no improvement in loss after just ~ 15 *minutes*.

We trained our network with **Tensorflow** 2.4.0 on a *NVIDIA GTX 3080 Ti* graphics card using Nvidia's **CUDA API** (*version 11.2*), as well as **cuDNN** *version 8.1*. We facilitated testing by using an Anaconda Navigator environment, with all of the relevant Python modules installed as well as the *version 3.8* of Python specific to the **CUDA API**.

V. Conclusion & Further Work

The images of the dataset was quite small to being with, which would have made it very hard to extract useful information if the architecture is too deep. Due to this we had to keep the convolutional neural network short whilst, also, extract maximum features, while keeping the size of neutrons in mind, as with increased number of neurons the model can just easily learn whole training set, resulting poorly on unseen data. We managed to tackle it with data augmentation and constraint model size, however, if given more time and resources we could have made improvements regarding this. Overall, the project was a great initiative to tackle the exclusive multi-classification problem, and it is only few percentiles away from high-end neural networks. Things mentioned above can be dealt with the use of aggressive data augmentation and pruning in further work.

Due to this very good performance, it can be concluded that the architecture that we used was an effective and appropriate one for it's use.

References

[1] T. Ropinski A. Bäuerle C. van Onzenoodt. *Net2Vis – A Visual Grammar for Automatically Generating Publication-Tailored CNN Architecture Visualizations*. 2021.

[2] Afshine Amidi and Shervine Amidi. *The evolution of image classification explained*. URL: https://stanford.edu/~shervine/blog/evolution-image-classification-explained.

[3] Siddhartha Sankar Nath et al. "A survey of image classification methods and techniques". In: *2014 International Conference on Control, Instrumentation, Communication and Computational Technologies (ICCICCT)*. 2014, pp. 554–557. DOI: 10.1109/ICCICCT.2014.6993023.