SMART TECH CA1 – IMAGE CLASSIFICATION

Michael Flynn & Kyle McQuillan

For this project we were tasked with creating an image classification model based on two datasets Cifar10 and Cifar100. The purpose of this model was to predict different hazards one might come across on the road such as vehicles, pedestrians, animals, and other hazards. The goal was to train the model based off certain classes from Cifar10 & Cifar100 to create an accurate enough model that could when tested with an image to be able to accurately identify what the image was.

**Image loading and extraction**

The first stage of this project would prove to be the most difficult and time consuming as it required us to take the data from both Cifar datasets and combine, then once combined we had to filter certain classes out of each dataset to train our model. The reason why we found this time consuming was we could not agree on the best approach in download the data and spent a lot of our time going back and forth from unpickling the data or importing the data sets. In the end we chose the later option as it meant we did not have to spend time worrying about the labels and matching them with the data.

The first function shown below was used to filter the datasets based on selected classes and done so by returning the filtered images and labels. Once we had our filtered dataset the next was to combine the two datasets together to begin our training, this was done using the built in concatenation function and merged the training labels, training images, test images and test labels of the datasets.

A screen shot of a computer program

Description automatically generated

Once the data had been combined, we then wanted to make sure that the classes we selected were working. To do this we ran a for loop that selected five random images from the list of unique classes.

A group of images of people

Description automatically generated

**Data Processing**

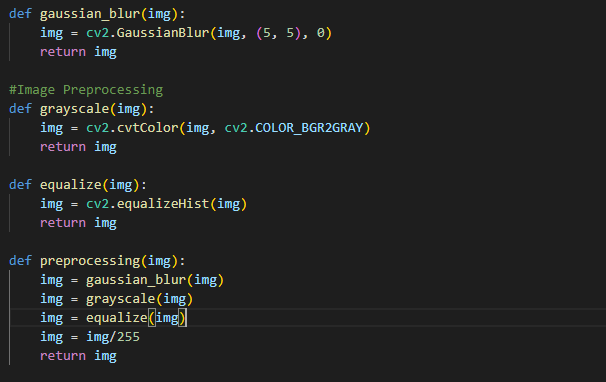
Once we had our classes ready to go before we tested the accuracy of the model we did some preprocessing of the data, in order to check for inconsistencies, noise, etc… once of the first things we did was resize the image using this function.

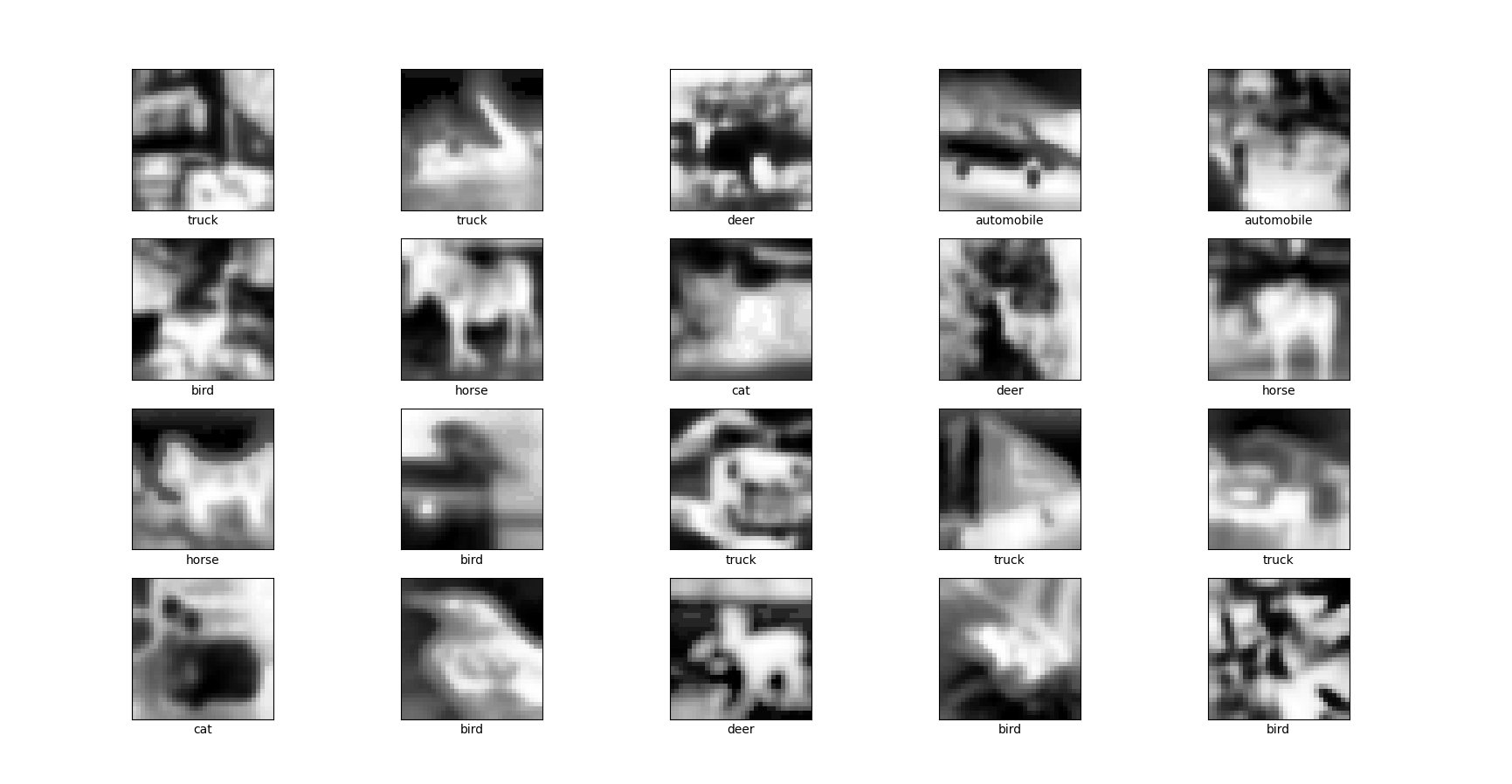
A screen shot of a computer code

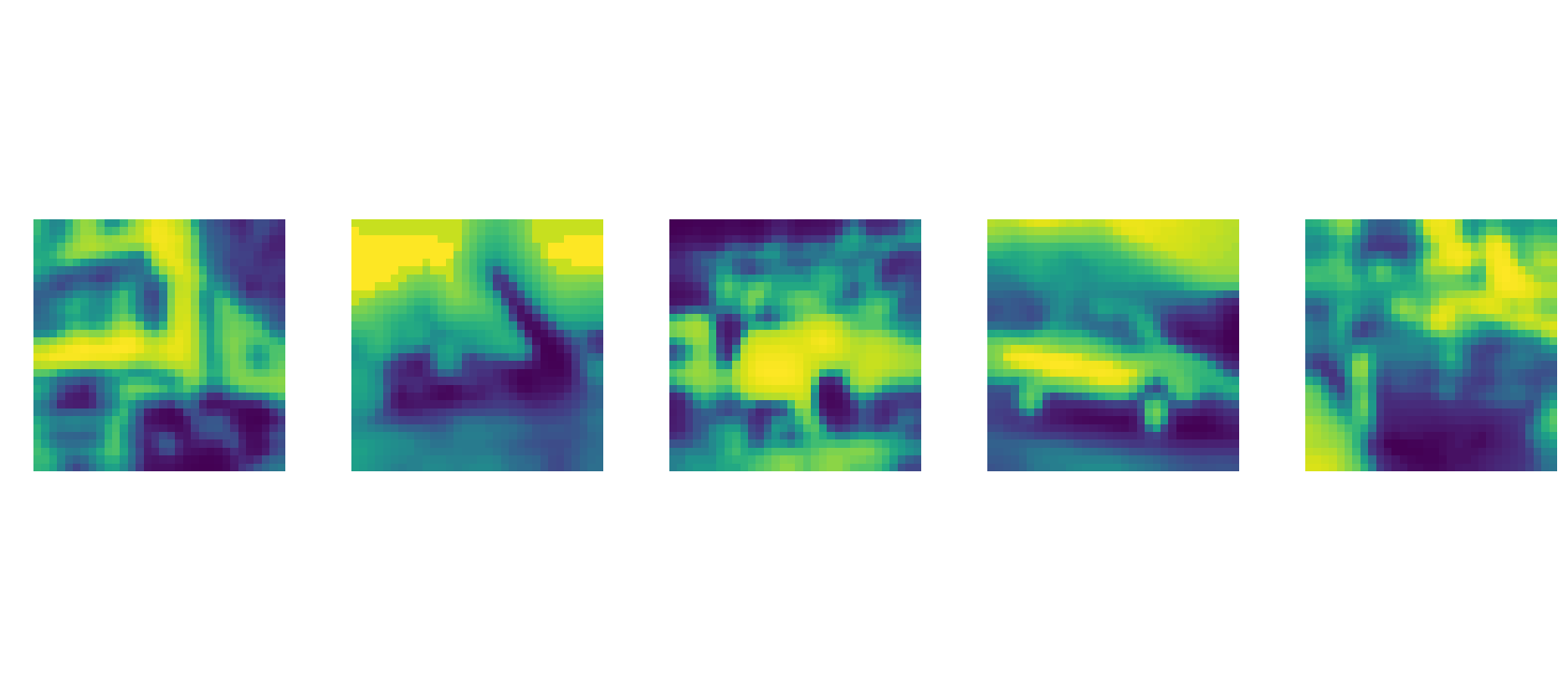
Description automatically generated

We did this to make sure all the images we are using will be 32 x32.

The next step for processing the image was changing the images look by applying grayscale to the image as it makes it easier for the computer to process the image, we also equalised the image, this can only be applied to an image after the grayscale and is used to standardise the lighting of a grayscale image, finally we normalised the image by dividing by 255. We also added a gaussian blur function as well



Some examples of the pre-processed data



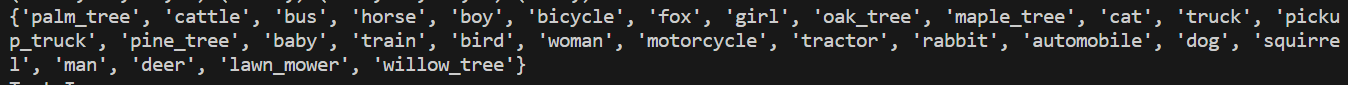
**Data Exploration**

The total number of images for the training and test sets after the combination was 50000 this was for both x and y

A black background with white text

Description automatically generated

There was 27 unique classes (this is including the classes with the tree superclass



The shape of the x\_train, y\_train, x\_test and y\_test

A black background with white numbers

Description automatically generated

**Building and Testing the Model**

Model One – Alpha Model

Once the two datasets had been combined into one, we created our first model which we simply called alpha model, this model was based off of one we had worked on in class as we felt like this was a good point to see what our accuracy would be

A screen shot of a computer program

Description automatically generated

As this is the first model we wanted to get a general idea of what the data would look like without any augmentation. We used preprocessing to grayscale the image. The accuracy came out around 55%. We ran this model for 10 epochs and looking at the screenshots below the data is underfitted

A graph of a training loss

Description automatically generatedA graph with a line and a red line

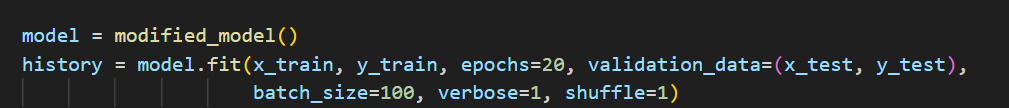
Description automatically generated

Before we started to work with augmenting the data to try and improve the accuracy, we made slight modifications to the alpha model function which we titled modified model. The main difference between these two models is that we changed the number of filters in the conv2d layer, we removed one of the dropout layers and finally we decreased the learning rate from 0.001 to 0.0001 in the hopes of better improving the training and validation accuracy of the model.

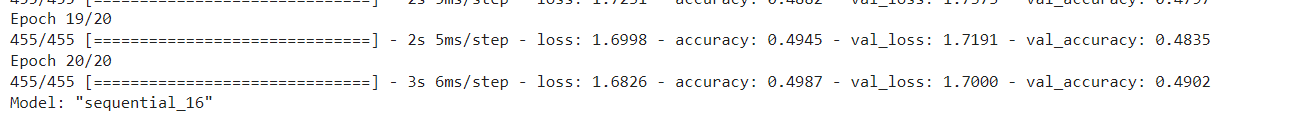
A screen shot of a computer code

Description automatically generated

We ran this model for 20 epochs with a batch size of 100



However when we actually ran this model the results turned out to be worse then the first one by almost 5%



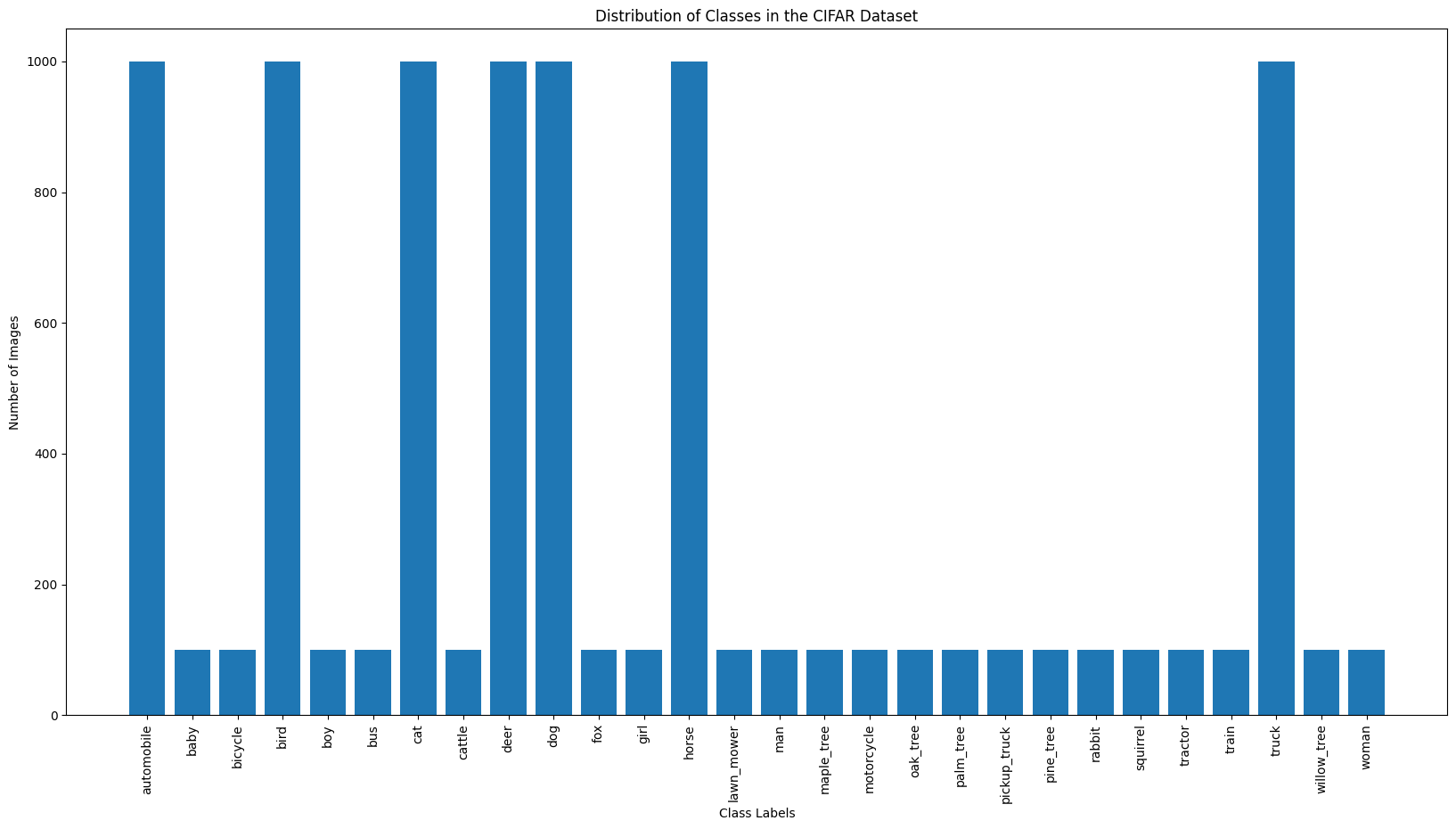
There was one positive of this model as it seem to be better fitted then the previous one. (there is slight overfitting and the loss is high)

A graph of a training loss

Description automatically generatedA graph of a graph

Description automatically generated

Looking at the distribution table shown below we can see that the there are certain classes that have very low data when compared when to the other classes (cifar10 has 4500 more then cifar100), so try and improve the accuracy without making changes to the model we then decided to augment some of the data in the hopes of improving accuracy this way.

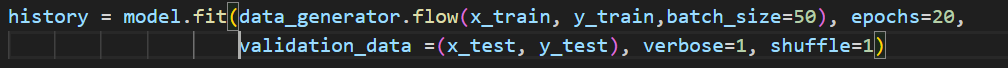


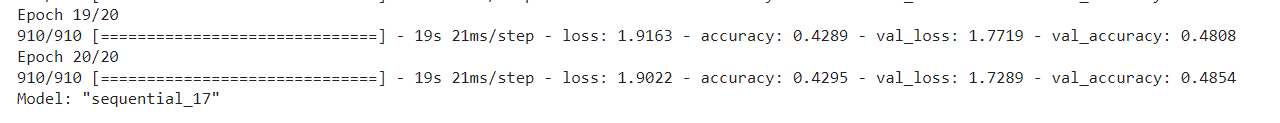
We used ImageDataGenerator to augment data by shifting the images, zooming them in and rotating them

A screen shot of a computer

Description automatically generated

We then refitted the model, to include this data generator to see what the accuracy would be like. However even with the data augmentation our accuracy was still poor





Model 3 – Beta Model

Our third model would prove to be the best model when it comes to accuracy, Again similar to our modification model the first change we did was to change the filters in Conv2d, we also added batch normalisation, an extra dropout layer and we reverted the learning rate back to 0.001

A screen shot of a computer program

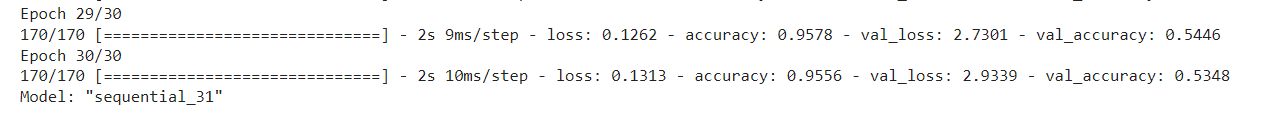
Description automatically generated

We ran this model twice, the first time without any data augmentation for 30 epochs and with a batch size of 200

A screen shot of a computer

Description automatically generated

The results for the accuracy came in at 95% but the validation accuracy was way below that it 50% which has been the case for the previous two models

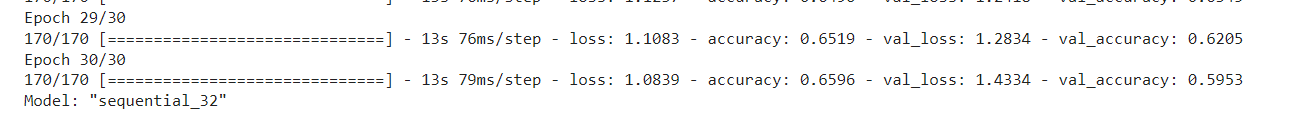


Again we ran this model with the image augmenter used in the modified models, we ran the same batches and epochs to give see how much of an impact the augmenter has on the data

A computer screen shot of a program code

Description automatically generated

With the augmentation there was an improvement in the validation accuracy but there was a large decrease in the training by nearly 30%



A graph of a training loss

Description automatically generatedA graph of a training performance

Description automatically generated