Building a more accurate Image Classification Model using data augmentation, CNN, Python and Keras with a GUI Frontend

Comp 4102 Final Project

Author Name : Oluwademilade Edward Akapo

Student No: 101095403

Abstract

In this project I built an Image classification model using CNN with the Cifar-10 dataset with a GUI that takes input images to be classified, the network was trained with augmented dataset and build with multiple layers to be more accurate against altered images from the specified categories. From my results I got a training accuracy of 83% And a validation accuracy of 77% , with a training loss of < 50% and a validation loss of 65%, evaluating the model with the testing set I got an accuracy of 76.43%. Self-testing my program I found that flipped and rotated images are generally classified very accurately but when an image is both flipped and rotated the classifications are much less accurate, to fix this I would investigate adding more layers in the CNN to better extract the features.

Introduction

In this project I built an Image classification model using CNN with the Cifar-10 dataset with a GUI that takes input images to be classified, the network was trained with augmented dataset and build with multiple layers to be more accurate.

I chose to do this project because I have a high familiarity with building CNN's and using Keras and Tensorflow as I've been using them extensively in my AI course and somewhat in my Computer Visions course. Moreover, I've had lots of previous experience building GUI with tkinter for SQL database software.

I originally attempted to use the Cifar-100 dataset with its "fine" 100 labels for my dataset, but I had issues creating the CNN layers. Using such a large dataset the training time per epoch increased dramatically, and it would have taken many epochs to get a sufficient accuracy for my model. Thus, I settled for using the Cifar-10 dataset.

Background

For this project I had to source some articles to fill gaps within my knowledge, when looking into translation invariance in neural networks I used a medium article[1] that spoke about the topic and how training with augmented data can improve the results of the model. For building the CNN layers using Keras I used another medium article[2] to understand the basic layout of a CNN and how the convolution, max-pooling and flatten layers work to extract the features from the image and classify it. Lastly the final source that I used for my project was a mlm article[3] that discussed the different activation functions and which ones to choose for deep learning, I used this to figure out the activation function for my output layer of my CNN, because I am doing multilabel classification I chose the sigmoid activation function. Alongside all these articles I also read a lot of documentation from tensorflow and keras websites relating to CNN.

Methodology

This section should describe your experimental approach and data sets.

The dataset that I used was the Cifar-10 dataset[4], this consist of 60,000 32x32 colour images organized into 10 classes with 6000 images per class. The data set is split into 50,000 training images and 10,000 testing images. I further split this down into 40,000 training images 10,000 validation images and 10,000 test images. The validation set it used to tune the models hyperparameters and configurations and the test set works best if the model hasn't already seen the images so that is why I split the training into a validation set, to get the optimal hyperparameters from the validation set and classification accuracy reports from the training set.

Firstly, I loaded the datasets from the Keras datasets, then I normalized the values so they will be between only 1 and 0. Like I described above I then split the datasets into training set, testing set and validation sets. Next, I created a function for real time data augmentation using the keras.preprocessing.image package, this function allows the model to train with various augmented data while it is running an epoch. After that I built the CNN model, The model is set to be sequential and every layer after that is inferred to be sequential. The first stack has 2 convolutional layers, a max-pooling layer, and a dropout layer. The first two layers removes the number of parameters used and leaves us with only the most significant ones to prevent overfitting, moreover it also reduces the amount of memory the network needs by reducing the size of the images. Then the dropout layer is then used to further prevent overfitting. The second stack is a repetition of the first, then the final stack with a flatten layer and 2 dense layers flattens the probabilities from the convolution layers into a 1D feature vector then the final dense layer is the same length as the number of classes and gives the probability of each class.

The CNN I used resembles the image seen below,

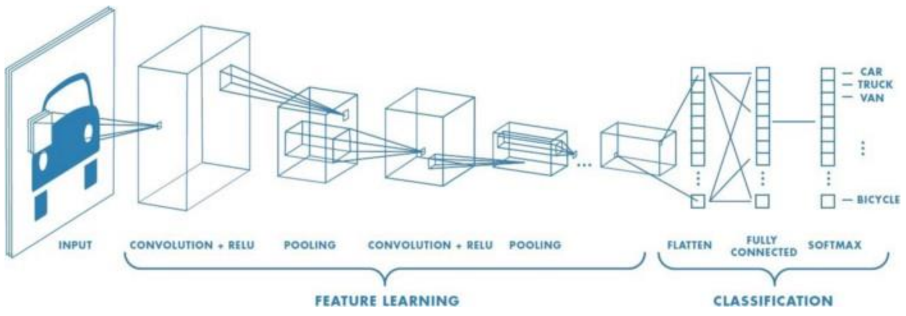


Figure 1: illustration of the CNN process from input to output[2]

Then using the using the “adam” optimizer and “sparce categorical crossentropy loss” I compiled the model and proceeded to fit it 3 times. The first fit was done with the normal train and validation data for 10 epochs, then I ran a second training using the data augmentation generator on the training set using the test set to validate for 5 epochs, lastly, I ran another training with normal data. I did this because training with noisy data(augmented) improves the robustness of the NN and reduces overfitting. After that I evaluate the model with the unseen test set to get the accuracy of the model. Then I plotted the accuracy graphs, saved the model, and created a GUI frontend to input images for classification.

Results

After I finished training the model, I plotted the accuracy and loss metrics seen below

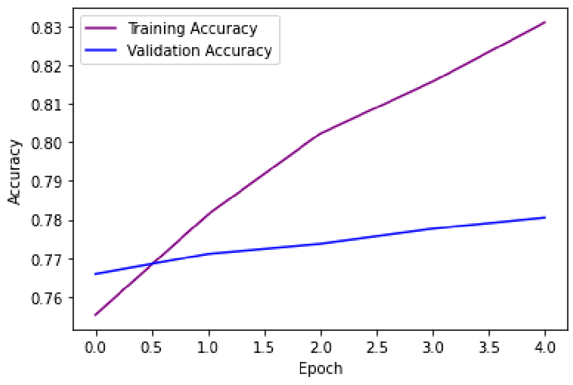


Figure 2 : plots showing training accuracy and validation accuracy

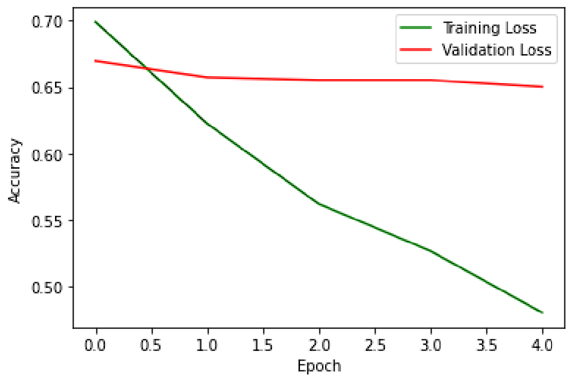
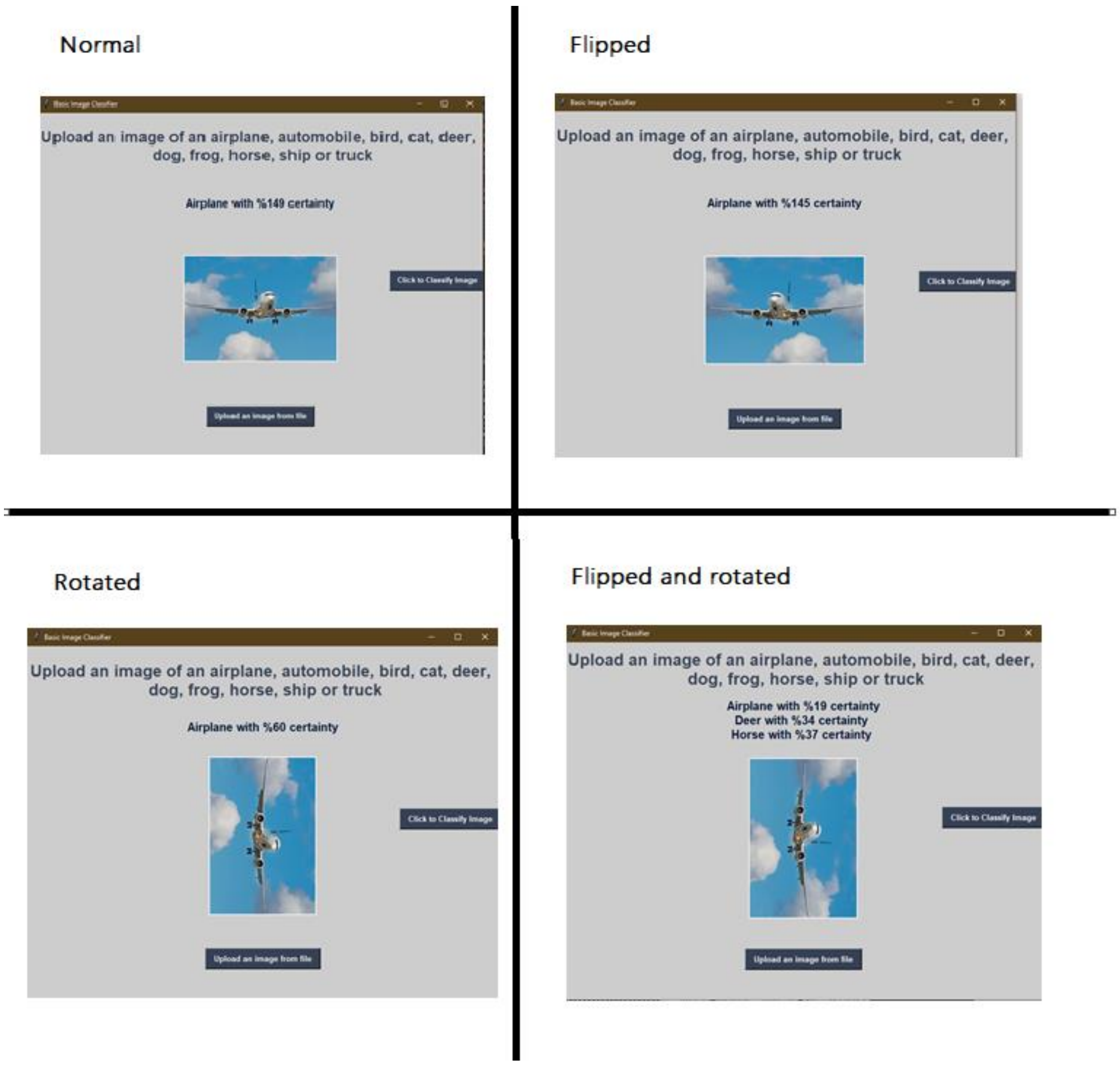


Figure 3 : plots showing training loss and validation loss

From figure 2 and 3 below we can see that our model is performing gradient descent well and gives very good performing, we can see the increasing accuracy and decreasing loss on the graphs.

Evaluating the model with the testing set we get a testing accuracy of 76.43%

In figure 4 we can see the results from testing the model with the GUI on a sample image of a plane gotten from the internet in different orientations, normal, flipped, rotated, flipped plus rotated:



Normal:

Upload an image of an airplane, automobile, bird, cat, deer, dog, frog, horse, ship or truck

```
Airplane with %149 certainty
```

Flipped:

Upload an image of an airplane, automobile, bird, cat, deer, dog, frog, horse, ship or truck

```
Airplane with %145 certainty
```

Rotated:

Upload an image of an airplane, automobile, bird, cat, deer, dog, frog, horse, ship or truck

```
Airplane with %60 certainty
```

Flipped and rotated:

Upload an image of an airplane, automobile, bird, cat, deer, dog, frog, horse, ship or truck

```
Airplane with %19 certainty
Deer with %34 certainty
Horse with %37 certainty
```

Figure 4 : results from the GUI

From the figure we can see that the program can take inputs of flipped and rotated images and they are generally classified very accurately but when an image is flipped and rotated the classifications is much less accurate.

Discussion

To improve the accuracy of the model in general and more so for flipped plus rotated images I could investigate adding multiple layers as this will help the model deal with multiple augmentations at the same time.

Other ways to improve the performance of the CNN would have been:

- To better tune our parameters like epoch and learning rate. If we could run for more epochs the model will have tuned its hyperparameters better.
- Using data augmentation to increase our training set size, more data would increase the accuracy of the system.
- Adding more layers to the CNN and increasing the size of the kernels, although this would make the network take a longer time to train, if I had the time, I could have adjusted these components so that it would have produced better results

For further work I could either investigate using a different dataset with more classifications on my model and testing it or I could investigate adding any of the accuracy improvements I mentioned above.

Architecture

The architecture of my CNN model

- Conv2D layer – we will add 2 convolutional layers of 32 filters, size of 3*3, and activation as relu
- Max Pooling – MaxPool2D with 2*2 layers
- Dropout with a rate of 0.1
- 2 Convolutional layers of 64 filters and size of 3*3
- Max Pooling – MaxPool2D with 2*2 layers
- Dropout with a rate of 0.25
- Flatten layer to squeeze the layers into 1 dimension
- Dense, feed-forward neural network(128 nodes, activation=”relu”)
- Dense layer(nodes=10, activation=”softmax”)
- MaxPool2D – Maximum pooling layer is used to reduce the size of images
- Dropout – Dropout is a regularization technique to reduce overfitting
- Flatten – to convert the parrel layers to squeeze the layers
- Dense – for feed-forward neural network
- the last layer will have an activation function as softmax for multi-class classification.

Bibliography

[1] Divyanshu .M, 2020, “Translational Invariance Vs Translational Equivariance” https://towardsdatascience.com/translational-invariance-vs-translational-equivariance-f9fbc8fca63a

[2] Angel .D , 2020, “Convolution Neural Network for Image Processing — Using Keras” https://towardsdatascience.com/convolution-neural-network-for-image-processing-using-keras-dc3429056306

[3] Jason .B, 2021 , “How to Choose an Activation Function for Deep Learning” https://machinelearningmastery.com/choose-an-activation-function-for-deep-learning/

[4] Alex .K, 2009, https://www.cs.toronto.edu/~kriz/cifar.html