Marcus Clarke

13/01/2019

Domain Background

The project is based around Deep Learning of neural networks for image classification. This is an area that has been around for a long time but has seen monumental expansion and development in more recent years. This has been a result of the increase in computer processing power and in particular the introduction of Graphic Processing Unit's (GPU's). Deep learning is of particular interest to me because of its ability to adapt to a wide variety of use cases.

Social media applications have developed a use for deep learning to classify images uploaded to their platforms, for example recognising human faces. Snapchat is an excellent example of how social media applies this deep learning technology. When the application detects a human face, the feature allows the user to alter their face in real-time, for example by applying dog ears, which follow the face when the screen moves. Facebook currently uses this image recognition technology to detect faces, and suggest their names to 'tag.' Apple also use image recognition as a security mechanism to unlock their new phones.

This project will focus on how image classification can be utilized to detect and translate letters of the alphabet from American Sign Language (ASL) into written text. Solving this particular problem would be very beneficial to allowing people who use ASL to communicate with people who do not understand ASL.

A link to the dataset discussed in this proposal is included in the footnoted below. [1]

Problem Statement

The problem to be solved is two fold:

1. Allowing people who communicate using ASL, to communicate more easily with people who cannot interpret sign language. This will operate at a very simple level by taking an image of a letter in the ASL alphabet and translating it into text.

[1] https://www.kaggle.com/grassknoted/asl-alphabet

2. Providing a simple to use learning interface, which can detect and recognise someone signing and provide accurate feedback, thereby allowing people to more easily learn ASL.

This is a common problem in the real world and one that would be very beneficial to solve via classification.

Datasets and Inputs

The dataset for this project was sourced from Kaggle[1]. This dataset contains 3,000 images for each letter of the ASL alphabet, along with the symbols for 'space', 'delete' and 'nothing'. It also contains a testing dataset with 1 image for each of the symbols mentioned above. The images are 200x200 pixels in size.

The training set is larger than will be necessary, however the testing set is not extensive enough. To counteract this I will split the training set into training, validation and testing sets. The training set will be approximately 200 images for each of the 29 different categories (4,988 in total), with the validation and testing sets being approximately 5-10% of the training set to reflect a truer accuracy level.

Solution Statement

The solution to the problem described above will be to train a CNN (Convolutional Neural Network) to correctly classify letters of the ASL alphabet. The problem is quantifiable as it is easy to measure the accuracy of the solution, which will be described in the evaluation metric section below.

If successful this solution could be used to allowing people who don't understand ASL to communicate with people who use ASL to communicate. It also has the potential to act as an interactive learning application for someone wanting to learn ASL.

Benchmark Model

Due to the complexity of comparing results from generated from two different datasets, I will be creating my own benchmark model to compare my final model against. The benchmark model will be a CNN, which I will build from scratch and generate some statistics on how accurate the model is.

This benchmark model can then be used as a reference when I have developed my final model, which will use transfer learning to build more complex and more accurate model.

Evaluation Metrics

The evaluation metrics for the model developed in this project will be based on how many of the testing images are detected and classified correctly. The formula for this is given below:

$$Accuracy = 100 * \frac{No.of\ correctly\ classified\ testing\ images}{Total\ No.of\ testing\ images}$$

This accuracy will be derived from the testing dataset that will be generated out of the raw dataset used for this project. I will also input a defined list of images that spell out a particular sentence and inspect how accurately this can be translated into written text. This will provide additional testing evidence, so anyone reading the project can easily see the output of the model in written text.

Project Design

This project will be split into three main sections as described below.

1. Importing the data and splitting it into training, validation and testing datasets.
    - The first step will be to download the data from Kaggle.
    - Choose an appropriate number of images to train, validate, and test. The validation and testing dataset should be around 5-10% of the training dataset.
    - Resize the images so they are all of equal size for analysing.
    - Convert the images into the correct RGB format. This will be outputted as a 4D tensor.
    - Some of the python libraries which can be used to help with this image processing are:
        - keras.preprocessing.image
        - tqdm.tqdm
        - PIL.ImageFile
        - cv2.imread

2. Generate models to process and analyse the data.
    - I will first develop a CNN model from scratch to display that the process will work by achieving an accuracy of greater than 3.5% (this will prove more accurate than using a random selection with 29 possible outcomes).

- Secondly, I will use transfer learning to make use of predefined CNN model.
- VGG16 is an example of a model that can be used for transfer learning. The last few fully connected layers of the VGG16 model can be removed and bottleneck features can be calculated from this.
    - from keras.applications.vgg16 import VGG16
    - model = VGG16(include_top = False)
- This will allow me to train my model to a high accuracy without needing to use a large number of trainable parameters. For example the VGG16 model uses 138,357,544 trainable parameters, which would take a significant length of time to compute. To display the VGG16 CNN model, the following code can be ran in python:
    - from keras.applications.vgg16 import VGG16
    - model = VGG16()
    - model.summary()

3. Test the accuracy of the models.
    - To measure the accuracy of the solution I will input a number of images to the CNN model. Accuracy will be measured as the percentage of correctly classified images based on the total number of testing images.
    - I will also input the images corresponding to a particular sentence and use my model to translate this into written text that can be analysed for accuracy.