Using Neural Networks to recognize sign language

Introduction & Background

Sign language is a form of communication used primarily by individuals who are unable to produce audible speech for communication. It is also used regularly by people who have to interact with hearing or speech impaired people on a regular basis.

In situations where options for writing are either not available or are impractical, efforts to communicate by people who use sign language to people who do not understand sign language can become extremely emotionally distressing.

Currently, the majority of people in developed countries have mobile phones which are equipped with a camera and are powerful enough to house applications which run machine learning models. This presents opportunities to develop a mobile application that could translate sign language into text or speech.

Having an application that could translate sign language into either text or speech would be akin to how Google translate is used to help bridge language challenges.

The first step towards this however is to develop a model that is able to translate simple static signs, such as those used for spelling out the alphabet.

In this study, we will explore the use of CNNs and DNNs for the development of a model for categorising hand signs used for spelling letters in the alphabet.

Method Description

For this study, we will be testing two different types of neural networks and the impact of different hyperparameters on model performance. The neural networks we will test are a Dense Neural Network (DNN) and a Convolutional Neural Network (CNN) to classify letters shown in sign language.

The hyperparameters we will focus on in this study are hidden sizes, learning rate, activation functions and optimizers. The dataset comes from the Sign Language MNIST dataset.

Dataset

The training and test sets consist of 27455 and 7172 images respectively, each representing a letter in the alphabet (A to Z). Each image is in grayscale with a dimension of 28x28 pixels. For training, validation and testing, we will use the full training set for training and split the test set equally to create a test and validation set.

CNN Model

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 28, 28, 16)        160

 max_pooling2d (MaxPooling2D  (None, 14, 14, 16)       0
 )

 conv2d_1 (Conv2D)           (None, 14, 14, 32)        4640

 max_pooling2d_1 (MaxPooling  (None, 7, 7, 32)         0
 2D)

 conv2d_2 (Conv2D)           (None, 7, 7, 16)          4624

 flatten (Flatten)           (None, 784)               0

 dense (Dense)               (None, 25)                19625

=================================================================
Total params: 29,049
Trainable params: 29,049
Non-trainable params: 0
```

Dense Model

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten_1 (Flatten)         (None, 784)               0

 dense_1 (Dense)             (None, 16)                12560

 dense_2 (Dense)             (None, 32)                544

 dense_3 (Dense)             (None, 16)                528

 dense_4 (Dense)             (None, 25)                425

=================================================================
Total params: 14,057
Trainable params: 14,057
Non-trainable params: 0
```

The DNN model consists of 1 flatten layer and 4 dense layers.

Layer Description

Below is a summary describing the purpose of each layer :

- **Conv2D** : This layer is used to enhance certain features of an image by applying a convolutional filter which will either increase or decrease the value of individual pixels.
- **MaxPool2D** : Max pooling is used to down-sample the input image to reduce its pixel dimensions which helps reduce the number of parameters in the model.
- **Flatten** : A Flatten layer converts the 2D image to a 1D vector.
- **Dense** : Dense layers are the standard deeply connected neural network layer

Hyperparameter Description

Hyperparameters are parameters that can be configured to guide the training process. How hyperparameters are setup will influence the values of the model parameters, which in turn determines the models performance.

Experiment

In total, we will perform 8 experiments after which we will select the best performing hyperparameters and model to train an optimised model and assess its performance on the test set.

The experiments will be broken down as follows :

- Experiment 1-1 : CNN Model - hiddensizes
- Experiment 1-2 : CNN Model - learning rate
- Experiment 1-3 : CNN Model - activation function
- Experiment 1-4 : CNN Model - optimizer
- Experiment 2-1 : DNN Model - hiddensizes
- Experiment 2-2 : DNN Model - learning rate
- Experiment 2-3 : DNN Model - activation function
- Experiment 2-4 : DNN Model - optimizer

Approach

In this study, we test the performance of two types of neural networks, DNN and CNN, for the classification of hand signs to letters of the alphabet. For each model, we will attempt to identify the best performing hyperparameters. The four hyperparameters we will test in this experiment are hidden sizes, learning rate, activation functions and optimizers.

Method Implementation (Code)

Conclusions and Recommendation

The current model has about 60 percent accuracy when applied to the Test data, which is consistent with the results of the validation data.

The difference in performance between the training and test data may suggest that the model is overfitting to the training data, however other issues that could be contributing to the poor accuracy of the model may include :

- Significant differences in the images used for training and testing leading to poor representation of test images in training data set
- Insufficient variations of hand signs in training data leading to over learning particular hand attributes for specific letters and weakened ability to generalise
- Poor model definition - the model may have been too simple
- Poor model optimisation - selected hyperparameters may not have been the optimal choices

For future experiments, we may want to add regularisation to help the model generalise more to the test data if the problem is mainly due to overfitting.

With further model development and optimisation, I believe the model could be improved to a higher level of accuracy.