Sign Language detection using deep learning

Capstone Project

Marion Bonnard

D21126437

09/01/2022

—

CPD Certificate in Foundations of AI

—

TU Dublin & Krisolis

Table of Contents

Project Initiation Document

1. Project Overview

Theme: Sign language recognition (object detection)

This AI project uses deep learning to detect and classify sign language. The idea is to train a model to recognise the letter of the alphabet in sign language using deep learning methods such as the convolutional neural network (CNN). The latter is a multi-layer neural network that can train on its own to improve accuracy. Each layer has a particular function that multiplies the weights with the input data to generate an outcome processed in the next layer as input. CNNs need to have annotated data to work.

1.1 Project Objective

This project aims to build a model that recognises the letters of the sign language alphabet, equivalent to hand gestures. The goal is to help solve the real-world problem of identifying sign language and improve the communication between deaf and hearing people thanks to AI.

1.1.1 Success Criteria

The success criteria are measured by the accuracy, precision, recall and f1-score. Other criteria are if the model managed to recognise the signs correctly. A validation set is used to determine the optimal number of epochs the training data has to go through. And a confusion matrix helps us determine the accuracy (Dias, 2019). Additionally, the success can be measured when the model is used in a real-time application using the camera of a device (mobile phone, laptop) to detect and label the sign. Ultimately, the goal is to build an application that translates sign gestures (including movement and facial expression) with subtitles. Thus, hearing people can understand and communicate with hard of hearing people in real-time.

1.2 Current Business Process

Some work has already been done using a convolutional neural network (CNN) to recognise sign language. In Pigou, et al., 2014, the authors have used CNN to automatically extract features and an artificial neural network (ANN) to classify each gesture or sign.

Many apps currently teach sign language via photos, videos, games, quizzes and help memorise the phrases and idioms with dialogue clips (Wilson & Melegrito, 2021). However, our project wants to design a model to detect and recognise sign language. Application such as the one created by the start-up SLAIT (Coldewey, 2021) is a solution that translates sign language into text in real-time. SLAIT's founders are using MediaPipe and neural networks to build their model. The latter can track features such as hands, arms, and facial expressions. It has been trained to recognise 200 American Sign Language (ASL) signs and simple sentences.

1.3 Project Assumptions

- How can we make computers recognise features associated with each sign?
- How can we use a convolutional neural network to detect the correct sign?

1.4 Risks

The risks are mainly linked with the dataset used. When using datasets from various open sources, we must use them personally or for practice. If a product has to be created, we must ask permission to use the dataset on a business level.

2. Solution Design

2.1 Proposed Analytics Solution

The proposed solution is a device or camera that can recognise the letter of the ASL alphabet. This solution helps break the communication barriers between the deaf and the hearing communities.

2.1.1 Solution Uses

A convolutional neural network (CNN) is trained to recognise the gesture/sign from the ASL dataset. The tools used are Python's programming language, with the TensorFlow and Keras library, some pre-trained model weights, and a large dataset containing thousands of images.

2.2 Data Requirements

There is a training dataset and a test dataset. The American Sign Language (ASL) dataset contains 26 folders with the letters of the alphabet (from A to Z) and three folders that include pictures of "nothing", "space", "delete". In the training dataset, there are 3000 images per folder, therefore 87,000 images in the format of 200x200 pixels. The test dataset contains 29 images of each category.

2.2.1 Data Source Description

The dataset ASL has been retrieved from the Kaggle website, a significant source of datasets and codes used to solve problems or realise a project in data analytics or AI.

3. Plan of Action

	Action	*Timeline*
1.	Business knowledge and continuous technical learning	16th November – 30th December
2.	Collected alphabet signs from the American Sign Language [1] (ASL) containing 29 folders.	29th November – 3rd December
3.	Labelling images for object detection	6th December – 19th December
4.	Converting images into computable format (using pixels)	6th December – 19th December
5.	Feature extraction	6th December – 19th December
6.	Classification	6th December – 19th December
7.	Model evaluation	27th December – 9th January
8.	Sign language detection	27th December – 9th January

[1] *https://www.kaggle.com/grassknoted/asl-alphabet?select=asl_alphabet_train*

Model Development Document

4. Project Overview

4.1 Project Objective

In a society that needs to be more inclusive, we want to ensure that everyone is equal no matter the gender, social group, community, race, or handicap. Inclusion means that everyone should have the chance to be culturally and socially accepted and welcomed (Anon., n.d.). To facilitate the inclusion of deaf people, we should find a way to narrow the communication gap between hearing and hard of hearing people. Not all hearing people understand sign language, and not all hard of hearing people can read on the lips. Additionally, following the pandemic and the obligation to wear masks everywhere, it has become more difficult for those who can read on the lips to understand conversions and see facial emotions. Thus, the objective of this project is to use Artificial Intelligence (AI) to ultimately create a sign language recognition device that translates sign language into sentences in real-time.

4.2 Outline of Solution

Using the American Sign Language (ASL) dataset, the solution is a trained model that detects and classifies the sign to the corresponding letter or symbol. It can be done using a dataset with an extensive collection of images and a deep learning technique called Convolutional Neural Network (CNN). Unlike machine learning, the neural network chooses the features in deep learning, meaning that the algorithm is constantly learning by itself.

5. Solutions Development

The approach taken for this project was to select the most appropriate dataset containing enough images to train the model and obtain the best accuracy.

5.1 Data Sample

5.1.1 Data Sample Cohort Definition

The data sample used to develop the solution is a publicly available Kaggle dataset[2]. The training dataset contains 87,000 images of alphabetic signs split into 29 folders. These folders contain images of the sign of the alphabet, two other signs such as "delete" and "space", and the last folder called "nothing" showing images with a neutral background and no signs or hands. Each folder has 3000 images for each sign. The test set contains 28 images similar to the above training dataset, except there is no "delete" test image.

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

5.1.2 Data Quality and Data Exclusions

In this project, we have only used the 26 signs and ignored the 3 additional signs such as delete, space and nothing. We aimed to only train the model with actual letters without adding more signs that might confuse the model. Additionally, we haven't used the original test dataset as it contained too few images. Instead, we used the training dataset with the 78,000 images and split it into training and test dataset.

5.1.3 Definition of Target Feature (for prediction)

Since we used a supervised deep learning technique, the data had to be annotated. Therefore, the target features of the model were the labels and expected output (A, B, C, D, etc.)

5.1.4 Definition of Features

In deep learning, the neural network chooses the features. The model was fed with a training dataset of alphabetic signs, and the pixels were feeding the neurons as the first layer or input layer. Weights were applied to the pixel values to extract the features from the images; this is called the convolutional step (West, 2019). Following the convolutional filter step, feature maps, which are computed versions of the image, were then generated. If the output layer did not match the desired output, the neural network went back to the hidden layers to adjust the weights, intending to reduce the error margin; this step is called back-propagation.

5.1.5 Training, Validation and Test Datasets

A few prediction models have been run with some changes in parameters. One of the parameters was the number of images selected. Depending on the number of images taken per label, the training sets contained either 21K, 31K or 62K of images; the sample size for the training dataset was determined by the images selected per label. On the final model, we have used the training dataset with 3000 images per label (26 x 3000), which means a total of 78,000 images; and assigned 20% to the test dataset (15,600) and 80% (62,400) to the training set. This step has been done with the below code in figure 1:

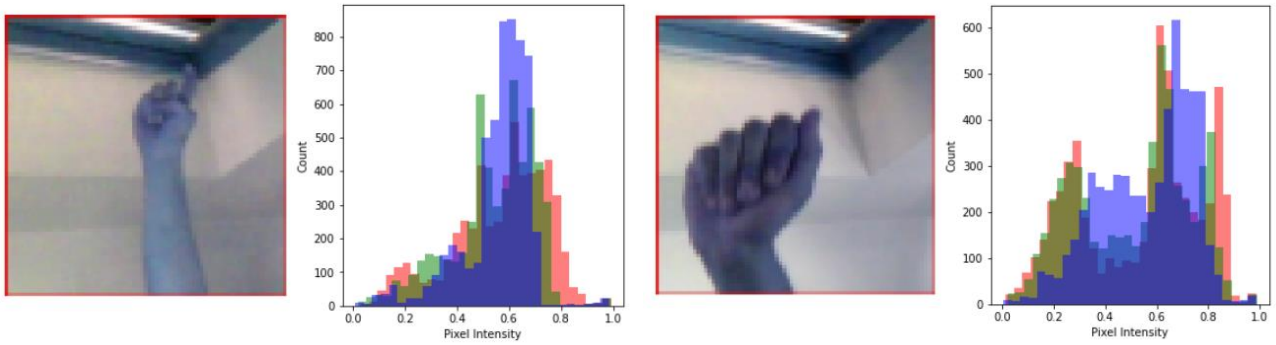
FIGURE 1

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X_train, y_train, test_size=0.2)
```

5.2 Exploratory Analysis

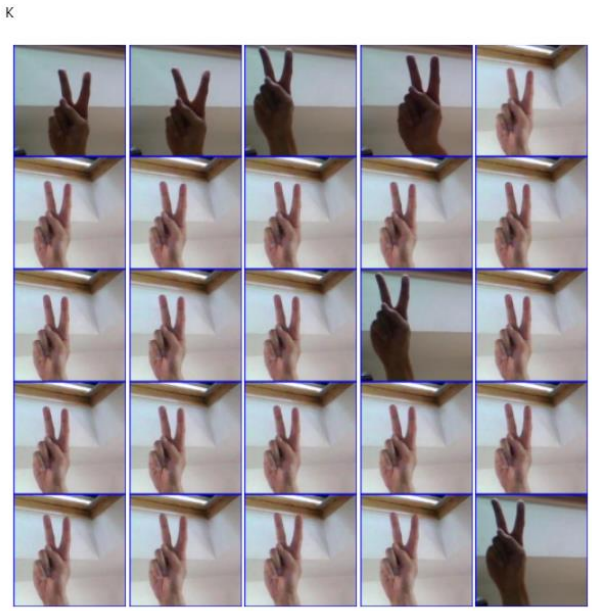
Some exploratory analyses have been completed to understand what is in the dataset. Some histograms have been plotted to showcase the primary colour's intensity in two images (figure 2). Since machines cannot process images the same as the human eye can, thus we have turned images input into numeric values. The neural network could perceive images with Red, Green and Blue values for each pixel.

FIGURE 2: HISTOGRAMS OF LETTERS X AND A



We have also displayed some images from the training set corresponding to a specific label in figure 3.

FIGURE 3: DISPLAYING LETTER K



Finally, a bar chart showing the frequency of each letter in the training dataset has been displayed in figure 4. This chart has been processed after the data shuffling step described in section 5.3.1. A training set of 62K images contained around 2,400 images per label, verifying a balanced dataset.

Another way of presenting the frequency of both training and test dataset is shown in figure 5.

FIGURE 4

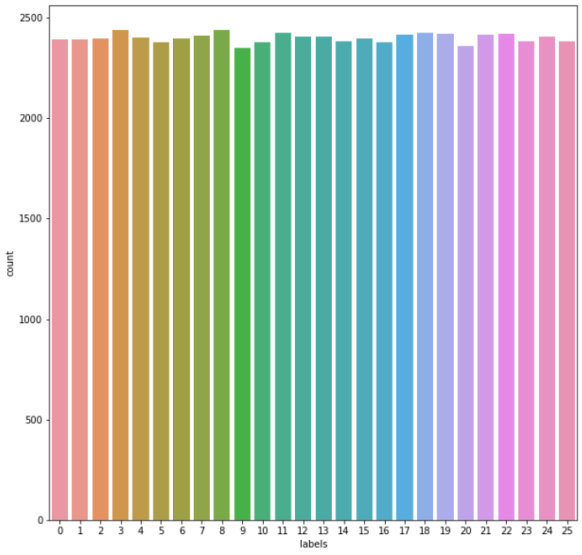


FIGURE 5: FREQUENCY OF TRAINING AND TEST IMAGE SETS

```
Training set:
        A: 2390, B: 2390, C: 2395, D: 2438, E: 2404, F: 2377, G: 2398, H: 2413, I: 2439, J: 2349, K: 2379, L: 2427, M: 2408, N:
2407, O: 2383, P: 2395, Q: 2377, R: 2415, S: 2427, T: 2419, U: 2361, V: 2417, W: 2421, X: 2382, Y: 2408, Z: 2381
Test set:
        A: 610, B: 610, C: 605, D: 562, E: 596, F: 623, G: 602, H: 587, I: 561, J: 651, K: 621, L: 573, M: 592, N: 593, O: 617,
P: 605, Q: 623, R: 585, S: 573, T: 581, U: 639, V: 583, W: 579, X: 618, Y: 592, Z: 619
```

5.3 Methodology

To build and run the below model, we have used the Python programming language. The code (in appendix 1) used to develop this model comes from the Kaggle website[3] and was published by Paul Mooney.

5.3.1 Pre-processing steps

The first step was to download the ASL dataset from Kaggle. Then, all the necessary packages have been imported, such as Keras, a neural network library built-in Python. It is a straightforward and user-friendly interface used in neural networks and is widely used in CNN. It uses TensorFlow in the backend. Tensorflow is an open-source library from Google that "excels at numerical computing", it is widely used in voice and image recognition and translation applications (Chandra, 2019). After importing the packages, the training and test datasets have been loaded. We then labelled the training dataset folders with numerics, for example, assigning 0 to letter A, 1 to letter B, etc. The next step was to split the training and test dataset as described in part 2.1.5, "Training, validation and test datasets". The labels were then encoded into One-Hot encoding vectors; this was necessary to run the model in Keras. Then, we shuffled the data to obtain the test and training set used in the modelling part.

5.3.2 Modelling

The modelling involved creating the functions that saved the metrics after each Epoch, creating the confusion matrix and printing the accuracy and loss curves.

We have run the functions using the "pre-trained models"[4]. The latter used in this project contained some weights that pre-defined patterns; at the last layer, the model used random weights to recognise unknown patterns and compared them to the correct answer. After comparing the answer, the model went back to the layers to change the weights until the result matched the proper label. The pre-trained model chosen for this work was InceptionV3 and VGG16. InceptionV3 is a pre-trained image recognition model (Gupta, 2020). Its main functions were to extract features and classify images; this model had 48 layers. This model was a transfer learning using images from the ImageNet database. VGG16 is a pre-trained model with pre-defined weights that define the patterns and keep self-learning to determine the useful patterns. It was considered an excellent "vision model" and has been somehow outperformed by Inception models (Chollet, 2016). The difference between these two models was that InceptionV3 had fewer parameters than VGG16; on the other hand, VGG16 was a simple model with 3X3 convolutional layers and had 16 sets of weights; however, the training phase was time-consuming.

Additionally, there were two hyper-parameters to consider in the code: the batch size and the number of epochs. The batch size corresponds to the number of samples passing through the neural network at one time. "An epoch […] is the number of passes that the machine learning algorithm has made over the entire dataset" (Hossain, 2021)

Finally, a model evaluation code has been programmed within a function using a test dataset to evaluate the accuracy, unveiling the confusion matrix and evaluation curves.

6. Results and Conclusions

After running a few models, we obtained the following results by changing a few hyperparameters.

6.1 Results and Evaluation

The first parameter to use was the pre-trained model, either VGG16 or InceptionV3. The other hyperparameter to consider was the number of images used in the training dataset, Epoch and batch. The number of batches remained

[3] *https://www.kaggle.com/paultimothymooney/interpret-sign-language-with-deep-learning/notebook*

[4] *https://www.kaggle.com/gaborfodor/keras-pretrained-models?select=vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5*

at 26 since it was the number of folders in our dataset; this way, 26 samples of the training set passed through the neural network at a time.

We have run six models (appendix 2) with various parameters and adjusted the number of Epoch and images selected. We have obtained the accuracies as a result shown in table 1.

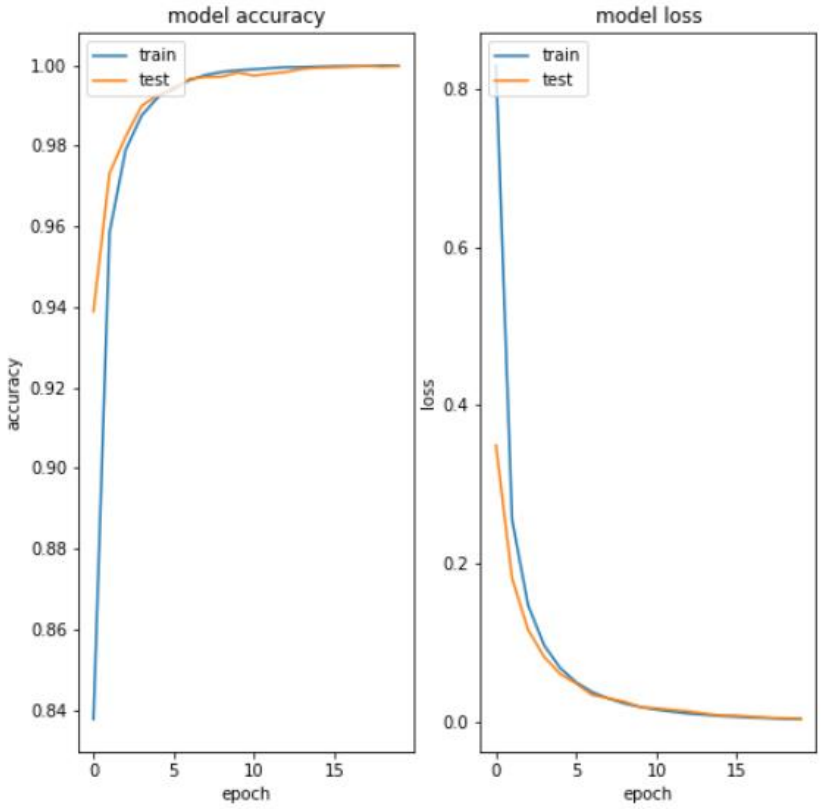
TABLE 1

Model number	Pre-trained model used	Nb of images selected	Training data (X_train & y_train)	Test data (X_test & y_test)	Nb of Batch	Nb of Epoch	Accuracy
1	VGG16	1000	20820	5206	26	20	0.9982
2	InceptionV3	1000	20820	5206	26	20	0.9907
3	VGG16	1500	31220	7806	26	5	0.9888
4	InceptionV3	1500	31220	7806	26	5	0.9711
5	VGG16	3000	62400	15600	26	20	**0.9996**
6	InceptionV3	3000	62400	15600	26	20	0.9805

To validate the best model, we have first compared which model brought the best accuracy; it seemed that the more images used in the labelled phase, the higher was the accuracy. Also, with 15+ Epoch, the models tended to get better accuracy.

The model with the best performance with 99.96% accuracy was model number five, a VGG16 pre-trained model with 3000 images selected, thus with a training dataset of 62K images and 20 epochs. The accuracy and loss curves displayed in figure 6 proved that over 15 epochs, we have obtained an accuracy close to 100%. In addition, the model loss had to be close to zero to reduce the errors.

FIGURE 6: MODEL ACCURACY AND MODEL LOSS CURVES



The subsequent performance measures were the precision, recall and F1-score table shown in table 2. In this table, we observed a rate of 99% for the precision for label 22, which corresponds to the letter 'W'. The precision rate indicates the model's reliability when classifying samples as positive (Korstanje, 2021). In this case, the precision rate is 99% accurate when predicting the letter W. Therefore, 1% of the test samples (579) were incorrectly predicted as 'W', where it should have been another sign. The previous result is evenly reflected in the recall. With a recall score of 99% for the letter 'S' (corresponding to label 18), the model has wrongly predicted 1% of the 573 samples as another letter while it should have been 'S'. These two results are connected and observed in the confusion matrix in figure 8. Finally, the F1-score combined precision and recall as the mean of these measures. As both precision and recall were high, the F1-score obtained a 100% score for all the labels.

TABLE 2: PRECISION, RECALL, F1-SCORE TABLE

	precision	recall	f1-score	support
0	1.00	1.00	1.00	610
1	1.00	1.00	1.00	610
2	1.00	1.00	1.00	605
3	1.00	1.00	1.00	562
4	1.00	1.00	1.00	596
5	1.00	1.00	1.00	623
6	1.00	1.00	1.00	602
7	1.00	1.00	1.00	587
8	1.00	1.00	1.00	561
9	1.00	1.00	1.00	651
10	1.00	1.00	1.00	621
11	1.00	1.00	1.00	573
12	1.00	1.00	1.00	592
13	1.00	1.00	1.00	593
14	1.00	1.00	1.00	617
15	1.00	1.00	1.00	605
16	1.00	1.00	1.00	623
17	1.00	1.00	1.00	585
18	1.00	0.99	1.00	573
19	1.00	1.00	1.00	581
20	1.00	1.00	1.00	639
21	1.00	1.00	1.00	583
22	0.99	1.00	1.00	579
23	1.00	1.00	1.00	618
24	1.00	1.00	1.00	592
25	1.00	1.00	1.00	619
accuracy			1.00	15600
macro avg	1.00	1.00	1.00	15600
weighted avg	1.00	1.00	1.00	15600

In the confusion matrix, figure 8, we can see that model five has wrongly predicted five images. First, it was for label 14, corresponding to the letter 'O', but it was actually 'D'. Three misclassifications were done with the letter 'W' (label 22), while it should have been the letter 'S' (label 18). Finally, we have observed that the model has wrongly classified one image 'V' as an 'R'. These misclassified images have been displayed in figure 9. It was understandable why the model has misclassified letter 'O' with 'D' and letter 'R' with 'V'; however, we could easily distinguish 'W' and 'S' where 'S' clearly showed a closed fist, while 'W' formed a W with three fingers.

FIGURE 7: CONFUSION MATRIX

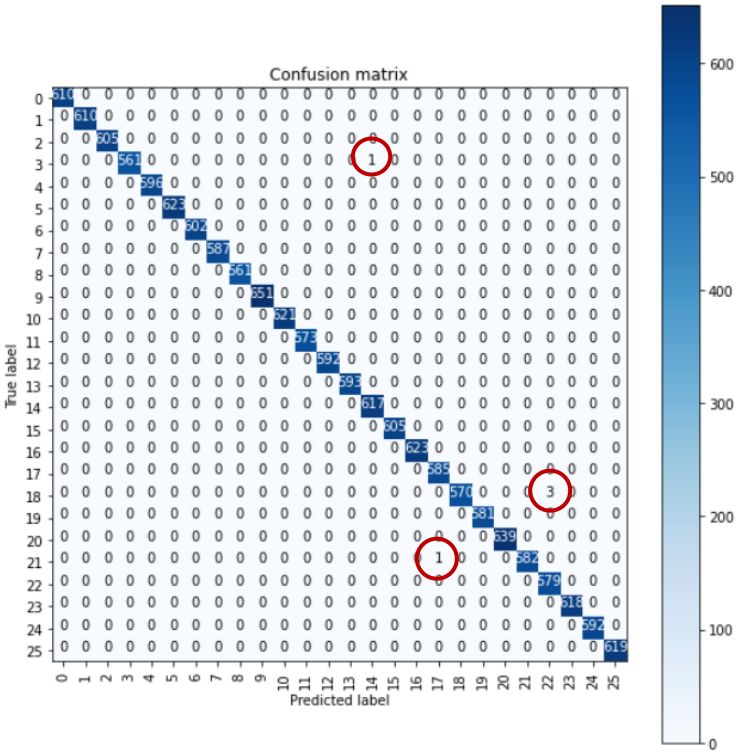


FIGURE 8: MISCLASSIFIED LETTERS

'O'

'D'

'W'

'S'

'R' 'V'


6.2 Conclusions and Limitations

To conclude, we have managed to reach the main objective of building a model that has detected and classified the sign language letters and obtained a high accuracy of 99.96%. The precision, recall and confusion matrix have displayed the classes where the model did not fully identify the correct letter. We could see that some signs were very much alike, such as 'R' and 'V' or 'O' and 'D' and therefore could easily be misclassified.

By obtaining a score of 99%, we can ask ourselves if the model is too good to obtain high accuracy and if it is working with a new test dataset. To test the model further and materialise it, we could develop an application connecting a camera and integrate the model. We could show some signs in this real-time application and see if the model correctly labels the sign. Alternatively, there are applications such as Gradio[5] , a python library that allows us to create a user-friendly interface for the model we have built.

In this project, we have faced a main limitation: the difficulty of processing a large amount of data on the Jupyter notebook. The labelling phase was sometimes not processed due to memory size issues. Also, for big data, it would preferably be to use GPU as when the model was running, it could take up to 14 hours.

[5] *https://gradio.app/working_with_ml/*

References

Akash, 2018. *ASL Alphabet.* [Online]
Available at: https://www.kaggle.com/grassknoted/asl-alphabet

Anon., n.d. *What is Diversity & Inclusion?.* [Online]
Available at: https://globaldiversitypractice.com/what-is-diversity-inclusion/
[Accessed 12 2021].

Chandra, R., 2019. *The What's What of Keras and TensorFlow.* [Online]
Available at: https://www.upgrad.com/blog/the-whats-what-of-keras-and-tensorflow/

Chollet, F., 2016. *How convolutional neural networks see the world.* [Online]
Available at: https://blog.keras.io/how-convolutional-neural-networks-see-the-world.html

Coldewey, D., 2021. *SLAIT's real-time sign language translation promises more accessible online communication.* [Online]
Available at: https://techcrunch.com/2021/04/26/slaits-real-time-sign-language-translation-promises-more-accessible-online-communication/
[Accessed 14 11 2021].

Dias, R., 2019. *American Sign Language Hand Gesture Recognition.* [Online]
Available at: https://towardsdatascience.com/american-sign-language-hand-gesture-recognition-f1c4468fb177
[Accessed 15 11 2021].

FOX 4 now, 2020. *Communication problems for the deaf community during the Coronavirus Pandemic.* [Online]
Available at: https://www.youtube.com/watch?v=uQmy7NYW2MA
[Accessed 27 12 2021].

Gupta, S., 2020. *Classify any Object using pre-trained CNN Model.* [Online]
Available at: https://towardsdatascience.com/classify-any-object-using-pre-trained-cnn-model-77437d61e05f

Hossain, E., 2021. *Difference Between the Batch size and Epoch in Neural Network.* [Online]
Available at: https://medium.com/mlearning-ai/difference-between-the-batch-size-and-epoch-in-neural-network-2d2cb2a16734

Korstanje, J., 2021. *The F1 score.* [Online]
Available at: https://towardsdatascience.com/the-f1-score-bec2bbc38aa6#:~:text=The%20F1%20score%3A%20combining%20Precision,work%20well%20on%20imbalanced%20data.

Mooney, P., 2018. *Interpret Sign Language with Deep Learning.* [Online]
Available at: https://www.kaggle.com/paultimothymooney/interpret-sign-language-with-deep-learning/notebook

Pigou, L., Dieleman, S., Kindermans, P.-J. & Schrauwen, B., 2014. Sign Language Recognition Using Convolutional Neural Networks. *Computer Vision ECCV 2014 Workshops,* Volume Part 1, pp. 572-578.

West, M., 2019. *Convolutional Neural Networks : The Theory.* [Online]
Available at: https://www.bouvet.no/bouvet-deler/understanding-convolutional-neural-networks-part-1

Wilson, D. & Melegrito, R., 2021. *Sign language apps for ASL: What to know and best options.* [Online]
Available at: https://www.medicalnewstoday.com/articles/sign-language-apps#_noHeaderPrefixedContent
[Accessed 14 11 2021].

Appendices

Appendix 1: Python code

Data source: https://www.kaggle.com/paultimothymooney/interpret-sign-language-with-deep-learning/data

```
# In[ ]:

from tensorflow.keras.optimizers import SGD, RMSprop, Adam, Adagrad, Adadelta, RMSprop

import keras

from keras.layers import Dense, Dropout, Activation, Flatten, Conv2D, MaxPooling2D, Lambda, MaxPool2D,
BatchNormalization

from keras.utils import np_utils

from keras.utils.np_utils import to_categorical

from keras.preprocessing.image import ImageDataGenerator

from keras import models, layers, optimizers

from sklearn.model_selection import train_test_split

from sklearn.metrics import confusion_matrix, accuracy_score

from sklearn.utils import class_weight

from keras.models import Sequential, model_from_json

from keras.layers import Activation,Dense, Dropout, Flatten, Conv2D, MaxPool2D,MaxPooling2D,AveragePooling2D,
BatchNormalization

from keras.preprocessing.image import ImageDataGenerator

from keras.callbacks import ReduceLROnPlateau, ModelCheckpoint

from keras import backend as K

from keras.applications.vgg16 import VGG16

from keras.models import Model

from keras.applications.inception_v3 import InceptionV3

import os
```

```
from glob import glob

import matplotlib.pyplot as plt

import random

import cv2

import pandas as pd

import numpy as np

import matplotlib.gridspec as gridspec

import seaborn as sns

import zlib

import itertools

import sklearn

import itertools

import scipy

import skimage

from skimage.transform import resize

import csv

from tqdm import tqdm

from sklearn import model_selection

from sklearn.model_selection import train_test_split, learning_curve,KFold,cross_val_score,StratifiedKFold

from sklearn.utils import class_weight

from sklearn.metrics import confusion_matrix

from imblearn.over_sampling import RandomOverSampler

from imblearn.under_sampling import RandomUnderSampler

get_ipython().run_line_magic('matplotlib', 'inline')
```

```
# In[ ]:

train_dir = "C:/Users/magno/Desktop/Fondations in AI/Python/Project/asl_alphabet_train/asl_alphabet_train"

test_dir = "C:/Users/magno/Desktop/Fondations in AI/Python/Project/asl_alphabet_test/asl_alphabet_test"

# In[ ]:

train_dir = "C:/Users/magno/Desktop/Fondations in AI/Python/Project/asl_alphabet_train/asl_alphabet_train/"



# In[ ]:

imageSize=75

from tqdm import tqdm

def get_data(folder):

    """

    Load the data and labels from the given folder.

    """

    X = []

    y = []

    for folderName in os.listdir(folder):

        if not folderName.startswith('.'):

            if folderName in ['A']:

                label = 0

            elif folderName in ['B']:

                label = 1

            elif folderName in ['C']:

                label = 2

            elif folderName in ['D']:
```

```
        label = 3
    elif folderName in ['E']:
        label = 4
    elif folderName in ['F']:
        label = 5
    elif folderName in ['G']:
        label = 6
    elif folderName in ['H']:
        label = 7
    elif folderName in ['I']:
        label = 8
    elif folderName in ['J']:
        label = 9
    elif folderName in ['K']:
        label = 10
    elif folderName in ['L']:
        label = 11
    elif folderName in ['M']:
        label = 12
    elif folderName in ['N']:
        label = 13
    elif folderName in ['O']:
        label = 14
    elif folderName in ['P']:
```

```
    label = 15
elif folderName in ['Q']:
    label = 16
elif folderName in ['R']:
    label = 17
elif folderName in ['S']:
    label = 18
elif folderName in ['T']:
    label = 19
elif folderName in ['U']:
    label = 20
elif folderName in ['V']:
    label = 21
elif folderName in ['W']:
    label = 22
elif folderName in ['X']:
    label = 23
elif folderName in ['Y']:
    label = 24
elif folderName in ['Z']:
    label = 25


num_images=0
for image_filename in tqdm(os.listdir(folder + folderName)):
```

```
            img_file = cv2.imread(folder + folderName + '/' + image_filename)
            if img_file is not None:
                img_file = skimage.transform.resize(img_file, (imageSize, imageSize, 3))
                img_arr = np.asarray(img_file)
                X.append(img_arr)
                y.append(label)
            num_images= num_images + 1
            if num_images > 3000:    # parameter that can be changed for less images in the dataset
                break
    X = np.asarray(X)
    y = np.asarray(y)
    return X,y
X_train, y_train = get_data(train_dir)


from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X_train, y_train, test_size=0.2)


# Encode labels to hot vectors (ex : 2 -> [0,0,1,0,0,0,0,0,0,0]) One hot encoded vector
from keras.utils.np_utils import to_categorical
y_trainHot = to_categorical(y_train, num_classes = 26)
y_testHot = to_categorical(y_test, num_classes = 26)


# Number of A's in the training dataset
num_A_train = sum(y_train==0)
```

```
# Number of B's in the training dataset

num_B_train = sum(y_train==1)

# Number of C's in the training dataset

num_C_train = sum(y_train==2)

num_D_train = sum(y_train==3)

num_E_train = sum(y_train==4)

num_F_train = sum(y_train==5)

num_G_train = sum(y_train==6)

num_H_train = sum(y_train==7)

num_I_train = sum(y_train==8)

num_J_train = sum(y_train==9)

num_K_train = sum(y_train==10)

num_L_train = sum(y_train==11)

num_M_train = sum(y_train==12)

num_N_train = sum(y_train==13)

num_O_train = sum(y_train==14)

num_P_train = sum(y_train==15)

num_Q_train = sum(y_train==16)

num_R_train = sum(y_train==17)

num_S_train = sum(y_train==18)

num_T_train = sum(y_train==19)

num_U_train = sum(y_train==20)

num_V_train = sum(y_train==21)

num_W_train = sum(y_train==22)
```

```
num_X_train = sum(y_train==23)

num_Y_train = sum(y_train==24)

num_Z_train = sum(y_train==25)


# Number of A's in the test dataset

num_A_test = sum(y_test==0)

# Number of B's in the test dataset

num_B_test = sum(y_test==1)

# Number of C's in the test dataset

num_C_test = sum(y_test==2)

num_D_test = sum(y_test==3)

num_E_test = sum(y_test==4)

num_F_test = sum(y_test==5)

num_G_test = sum(y_test==6)

num_H_test = sum(y_test==7)

num_I_test = sum(y_test==8)

num_J_test = sum(y_test==9)

num_K_test = sum(y_test==10)

num_L_test = sum(y_test==11)

num_M_test = sum(y_test==12)

num_N_test = sum(y_test==13)

num_O_test = sum(y_test==14)

num_P_test = sum(y_test==15)

num_Q_test = sum(y_test==16)
```

```
num_R_test = sum(y_test==17)

num_S_test = sum(y_test==18)

num_T_test = sum(y_test==19)

num_U_test = sum(y_test==20)

num_V_test = sum(y_test==21)

num_W_test = sum(y_test==22)

num_X_test = sum(y_test==23)

num_Y_test = sum(y_test==24)

num_Z_test = sum(y_test==25)



# Print statistics about the dataset

print("Training set:")

print("\tA: {}, B: {}, C: {}, D: {}, E: {}, F: {}, G: {}, H: {}, I: {}, J: {}, K: {}, L: {}, M: {}, N: {}, O: {}, P: {}, Q: {}, R: {}, S: {}, T: {}, U: {}, V: {}, W: {}, X: {}, Y: {}, Z: {}".format(num_A_train,

num_B_train, num_C_train, num_D_train, num_E_train, num_F_train, num_G_train, num_H_train, num_I_train, num_J_train,

num_K_train, num_L_train, num_M_train, num_N_train, num_O_train, num_P_train, num_Q_train, num_R_train,
num_S_train,

num_T_train, num_U_train, num_V_train, num_W_train, num_X_train, num_Y_train, num_Z_train))

print("Test set:")

print("\tA: {}, B: {}, C: {}, D: {}, E: {}, F: {}, G: {}, H: {}, I: {}, J: {}, K: {}, L: {}, M: {}, N: {}, O: {}, P: {}, Q: {}, R: {}, S: {}, T: {}, U: {}, V: {}, W: {}, X: {}, Y: {}, Z: {}".format(num_A_test,

num_B_test, num_C_test, num_D_test, num_E_test, num_F_test, num_G_test, num_H_test, num_I_test, num_J_test,

num_K_test, num_L_test, num_M_test, num_N_test, num_O_test, num_P_test, num_Q_test, num_R_test, num_S_test,

num_T_test, num_U_test, num_V_test, num_W_test, num_X_test, num_Y_test, num_Z_test))



# In[ ]:
```

```
# Shuffle data to permit further subsampling

from sklearn.utils import shuffle

X_train, y_trainHot = shuffle(X_train, y_trainHot, random_state=13)

X_test, y_testHot = shuffle(X_test, y_testHot, random_state=13)

X_train = X_train[:90000]

X_test = X_test[:30000]

y_trainHot = y_trainHot[:90000]

y_testHot = y_testHot[:30000]



# In[ ]:

len(X_test)

len(y_testHot)

len(y_train)

len(X_train)

len(y_trainHot)

print(X_train)

print(y_train)

print(y_trainHot)

len(y_trainHot)

len(y_testHot)

# In[ ]:

def plotHistogram(a):

    """

    Plot histogram of RGB Pixel Intensities
```

```
    """
    plt.figure(figsize=(10,5))
    plt.subplot(1,2,1)
    plt.imshow(a)
    plt.axis('off')
    histo = plt.subplot(1,2,2)
    histo.set_ylabel('Count')
    histo.set_xlabel('Pixel Intensity')
    n_bins = 30
    plt.hist(a[:,:,0].flatten(), bins= n_bins, lw = 0, color='r', alpha=0.5);
    plt.hist(a[:,:,1].flatten(), bins= n_bins, lw = 0, color='g', alpha=0.5);
    plt.hist(a[:,:,2].flatten(), bins= n_bins, lw = 0, color='b', alpha=0.5);
plotHistogram(X_train[2])
# In[ ]:
multipleImages = glob('C:/Users/magno/Desktop/Fondations in AI/Python/Project/asl_alphabet_train/asl_alphabet_train/A/**')
def plotThreeImages(images):
    r = random.sample(images, 3)
    plt.figure(figsize=(16,16))
    plt.subplot(131)
    plt.imshow(cv2.imread(r[0]))
    plt.subplot(132)
    plt.imshow(cv2.imread(r[1]))
    plt.subplot(133)
    plt.imshow(cv2.imread(r[2]))
```

```
    #;
plotThreeImages(multipleImages)
# In[ ]:
multipleImages = glob('C:/Users/magno/Desktop/Fondations in AI/Python/Project/asl_alphabet_train/asl_alphabet_train/B/**')
def plotThreeImages(images):
    r = random.sample(images, 3)
    plt.figure(figsize=(16,16))
    plt.subplot(131)
    plt.imshow(cv2.imread(r[0]))
    plt.subplot(132)
    plt.imshow(cv2.imread(r[1]))
    plt.subplot(133)
    plt.imshow(cv2.imread(r[2]))
plotThreeImages(multipleImages)
# In[ ]:
print("A")
multipleImages = glob('C:/Users/magno/Desktop/Fondations in AI/Python/Project/asl_alphabet_train/asl_alphabet_train/A/**')
i_ = 0
plt.rcParams['figure.figsize'] = (9.0, 9.0)
plt.subplots_adjust(wspace=0, hspace=0)
for l in multipleImages[:25]:
    im = cv2.imread(l)
    im = cv2.resize(im, (128, 128))
    plt.subplot(5, 5, i_+1) #.set_title(l)
```

```
    plt.imshow(cv2.cvtColor(im, cv2.COLOR_BGR2RGB)); plt.axis('off')
    i_ += 1
# In[ ]:
print("B")
multipleImages = glob('C:/Users/magno/Desktop/Fondations in AI/Python/Project/asl_alphabet_train/asl_alphabet_train/B/**')
i_ = 0
plt.rcParams['figure.figsize'] = (10.0, 10.0)
plt.subplots_adjust(wspace=0, hspace=0)
for l in multipleImages[:25]:
    im = cv2.imread(l)
    im = cv2.resize(im, (128, 128))
    plt.subplot(5, 5, i_+1) #.set_title(l)
    plt.imshow(cv2.cvtColor(im, cv2.COLOR_BGR2RGB)); plt.axis('off')
    i_ += 1
# In[ ]:
map_characters = {0: 'A', 1: 'B', 2: 'C', 3: 'D', 4: 'E', 5: 'F', 6: 'G', 7: 'H', 8: 'I', 9: 'J', 10: 'K', 11: 'L',
                  12: 'M', 13: 'N', 14: 'O', 15: 'P', 16: 'Q', 17: 'R', 18: 'S', 19: 'T', 20: 'U', 21: 'V', 22: 'W',
                  23: 'X', 24: 'Y', 25: 'Z'}
dict_characters=map_characters
import seaborn as sns
df = pd.DataFrame()
df["labels"]=y_train
lab = df['labels']
dist = lab.value_counts()
```

```
sns.countplot(lab)

print(dict_characters)

# In[ ]:

map_characters = {0: 'A', 1: 'B', 2: 'C', 3: 'D', 4: 'E', 5: 'F', 6: 'G', 7: 'H', 8: 'I', 9: 'J', 10: 'K', 11: 'L',
                  12: 'M', 13: 'N', 14: 'O', 15: 'P', 16: 'Q', 17: 'R', 18: 'S', 19: 'T', 20: 'U', 21: 'V', 22: 'W',
                  23: 'X', 24: 'Y', 25: 'Z'}

dict_characters=map_characters

import seaborn as sns

df = pd.DataFrame()

df["labels"]=y_test

lab = df['labels']

dist = lab.value_counts()

sns.countplot(lab)

print(dict_characters)

# In[ ]:

# Helper Functions  Learning Curves and Confusion Matrix

from keras.callbacks import Callback, EarlyStopping, ReduceLROnPlateau, ModelCheckpoint  #saving checkpoint

class MetricsCheckpoint(Callback):
    """Callback that saves metrics after each epoch"""
    def __init__(self, savepath):
        super(MetricsCheckpoint, self).__init__()
        self.savepath = savepath
        self.history = {}
    def on_epoch_end(self, epoch, logs=None):
```

```
        for k, v in logs.items():
            self.history.setdefault(k, []).append(v)
        np.save(self.savepath, self.history)


def plotKerasLearningCurve():
    plt.figure(figsize=(10,5))
    metrics = np.load('logs.npy')[()]
    filt = ['acc'] # try to add 'loss' to see the loss learning curve
    for k in filter(lambda x : np.any([kk in x for kk in filt]), metrics.keys()):
        l = np.array(metrics[k])
        plt.plot(l, c= 'r' if 'val' not in k else 'b', label='val' if 'val' in k else 'train')
        x = np.argmin(l) if 'loss' in k else np.argmax(l)
        y = l[x]
        plt.scatter(x,y, lw=0, alpha=0.25, s=100, c='r' if 'val' not in k else 'b')
        plt.text(x, y, '{} = {:.4f}'.format(x,y), size='15', color= 'r' if 'val' not in k else 'b')
    plt.legend(loc=4)
    plt.axis([0, None, None, None]);
    plt.grid()
    plt.xlabel('Number of epochs')
    plt.ylabel('Accuracy')
def plot_confusion_matrix(cm, classes,          #confusion matrix
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
```

```
    """
    This function prints and plots the confusion matrix.
    Normalisation can be applied by setting `normalize=True`.
    """
    plt.figure(figsize = (8,8))
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=90)
    plt.yticks(tick_marks, classes)
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, cm[i, j],
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")
    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')


def plot_learning_curve(history):          #learning curve
    plt.figure(figsize=(8,8))
```

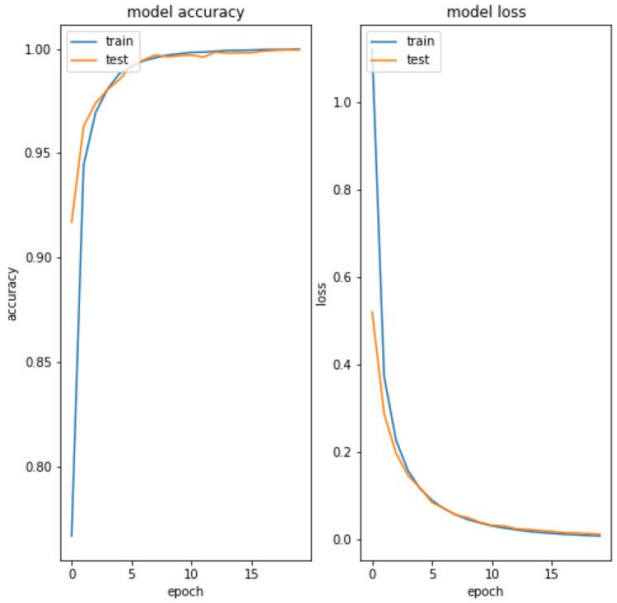
```
    plt.subplot(1,2,1)
    plt.plot(history.history['acc'])
    plt.plot(history.history['val_acc'])
    plt.title('model accuracy')
    plt.ylabel('accuracy')
    plt.xlabel('epoch')
    plt.legend(['train', 'test'], loc='upper left')
    plt.savefig('./accuracy_curve.png')
    plt.subplot(1,2,2)
    plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.title('model loss')
    plt.ylabel('loss')
    plt.xlabel('epoch')
    plt.legend(['train', 'test'], loc='upper left')
    plt.savefig('./loss_curve.png')
# In[ ]:
map_characters1 = map_characters
class_weight1 = class_weight.compute_class_weight('balanced', np.unique(y_train), y_train)
weight_path1 = 'C:/Users/magno/Desktop/Fondations in
AI/Python/Project/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5'
#weight_path2 = 'C:/Users/magno/Desktop/Fondations in
AI/Python/Project/inception_v3_weights_tf_dim_ordering_tf_kernels_notop.h5'
pretrained_model_1 = VGG16(weights = weight_path1, include_top=False, input_shape=(imageSize, imageSize, 3))
#pretrained_model_2 = InceptionV3(weights = weight_path2, include_top=False, input_shape=(imageSize, imageSize, 3))
```

```
optimizer1 = Adam()

optimizer2 = RMSprop(lr=0.0001)

def pretrainedNetwork(xtrain,ytrain,xtest,ytest,pretrainedmodel,pretrainedweights,classweight,numclasses,numepochs,optimizer,labels):

    base_model = pretrainedmodel # Topless

    # Add top layer

    x = base_model.output

    x = Flatten()(x)

    predictions = Dense(numclasses, activation='softmax')(x)

    model = Model(inputs=base_model.input, outputs=predictions)

    # Train top layer

    for layer in base_model.layers:

        layer.trainable = False

    model.compile(loss='categorical_crossentropy',

                  optimizer=optimizer,

                  metrics=['acc'])

    callbacks_list = [keras.callbacks.EarlyStopping(monitor='val_acc', patience=3, verbose=1)]

    model.summary() #printing the details



    # Fit model

    history = model.fit(xtrain,ytrain, epochs=numepochs, validation_data=(xtest,ytest), verbose=1,callbacks = [MetricsCheckpoint('logs')])

    plot_learning_curve(history)

    plt.show()

    return model
```

```
# In[ ]:

def evaluate_model (model,xtest,ytest):

  # Evaluate model

  score = model.evaluate(xtest,ytest, verbose=0)

  print('\nKeras CNN - accuracy:', score[1], '\n')

  y_pred = model.predict(xtest)

  print('\n', sklearn.metrics.classification_report(np.where(ytest > 0)[1], np.argmax(y_pred, axis=1)), sep='')

  Y_pred_classes = np.argmax(y_pred,axis = 1)

  Y_true = np.argmax(ytest,axis = 1)

  confusion_mtx = confusion_matrix(Y_true, Y_pred_classes)

  plot_confusion_matrix(confusion_mtx, classes = list(range(0,26)))

  plt.show()

# In[ ]:

trained_model_VGG16= pretrainedNetwork(X_train, y_trainHot, X_test,
y_testHot,pretrained_model_1,weight_path1,class_weight1,

                        26,20,optimizer1,map_characters1)

# In[ ]:

evaluate_model(trained_model_VGG16,X_test, y_testHot)  #testing data

# In[ ]:

trained_model_VGG16.save('asl_saved_model_VGG16_26_3000_20') #save the model with 2000 images per letter and 10
Epoch

# In[ ]:

loaded_model_5= keras.models.load_model('asl_saved_model_VGG16_26_3000_20')

# In[ ]:

evaluate_model(loaded_model_5,X_test, y_testHot)  #testing data
```

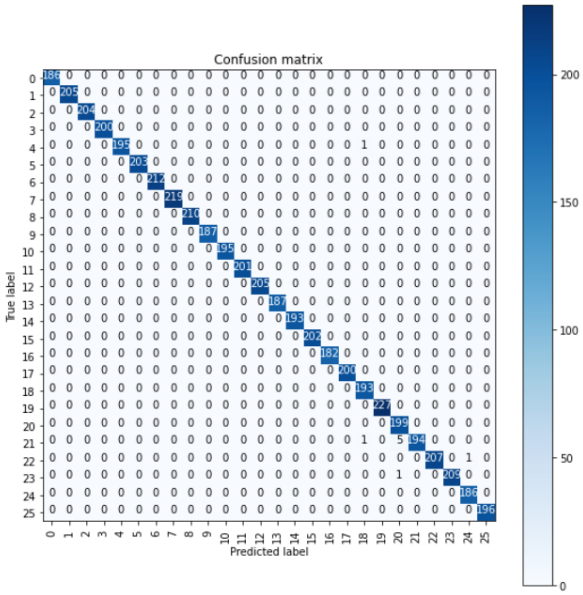
Appendix 2: Models evaluation

Model 1: Using VGG16 with 1000 images per letter and 26 letters and 20 EPOCH

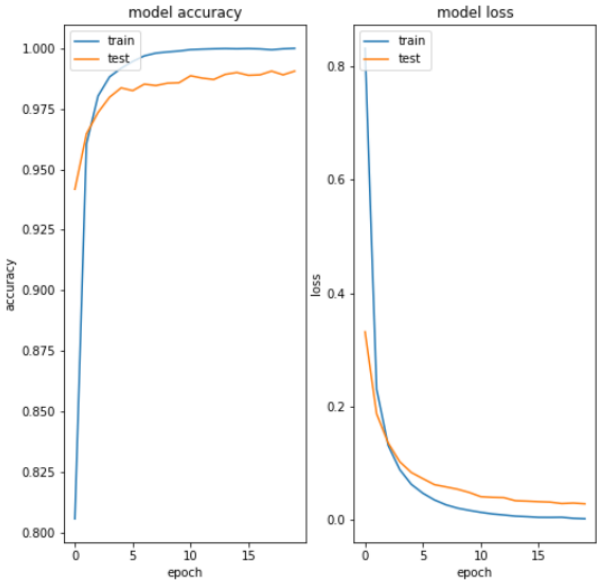


Keras CNN - accuracy: 0.9982712268829346

	precision	recall	f1-score	support
0	1.00	1.00	1.00	186
1	1.00	1.00	1.00	205
2	1.00	1.00	1.00	204
3	1.00	1.00	1.00	200
4	1.00	0.99	1.00	196
5	1.00	1.00	1.00	203
6	1.00	1.00	1.00	212
7	1.00	1.00	1.00	219
8	1.00	1.00	1.00	210
9	1.00	1.00	1.00	187
10	1.00	1.00	1.00	195
11	1.00	1.00	1.00	201
12	1.00	1.00	1.00	205
13	1.00	1.00	1.00	187
14	1.00	1.00	1.00	193
15	1.00	1.00	1.00	202
16	1.00	1.00	1.00	182
17	1.00	1.00	1.00	200
18	0.99	1.00	0.99	193
19	1.00	1.00	1.00	227
20	0.97	1.00	0.99	199
21	1.00	0.97	0.98	200
22	1.00	1.00	1.00	208
23	1.00	1.00	1.00	210
24	0.99	1.00	1.00	186
25	1.00	1.00	1.00	196
accuracy			1.00	5206
macro avg	1.00	1.00	1.00	5206
weighted avg	1.00	1.00	1.00	5206

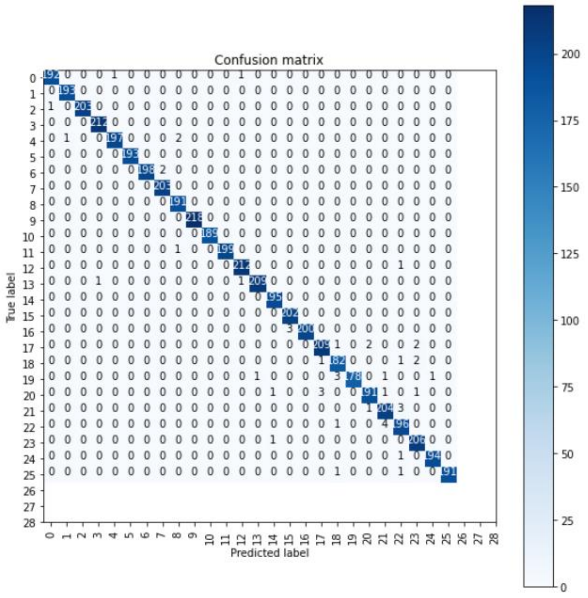


Model 2: Using INCEPTIONV3 with 1000 images per letter and 26 letters and 20 EPOCH
(running for about 1 hour)

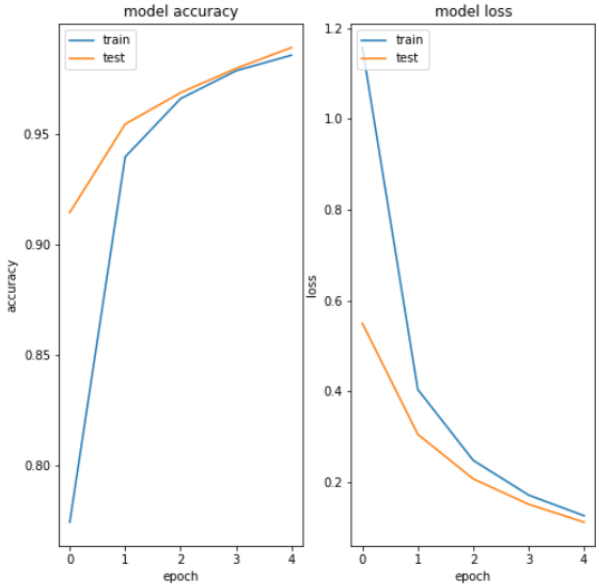


Keras CNN - accuracy: 0.9934690594673157

	precision	recall	f1-score	support
0	0.99	0.99	0.99	194
1	0.99	1.00	1.00	193
2	1.00	1.00	1.00	204
3	1.00	1.00	1.00	212
4	0.99	0.98	0.99	200
5	1.00	1.00	1.00	193
6	1.00	0.99	0.99	200
7	0.99	1.00	1.00	203
8	0.98	1.00	0.99	191
9	1.00	1.00	1.00	218
10	1.00	1.00	1.00	189
11	1.00	0.99	1.00	200
12	0.99	1.00	0.99	213
13	1.00	0.99	0.99	211
14	0.99	1.00	0.99	195
15	0.99	1.00	0.99	202
16	1.00	0.99	0.99	203
17	0.98	0.98	0.98	214
18	0.97	0.98	0.97	186
19	1.00	0.97	0.98	184
20	0.98	0.97	0.98	197
21	0.97	0.98	0.98	208
22	0.97	0.98	0.97	201
23	0.98	1.00	0.99	207
24	0.99	0.99	0.99	195
25	1.00	0.99	0.99	193
accuracy			0.99	5206
macro avg	0.99	0.99	0.99	5206
weighted avg	0.99	0.99	0.99	5206

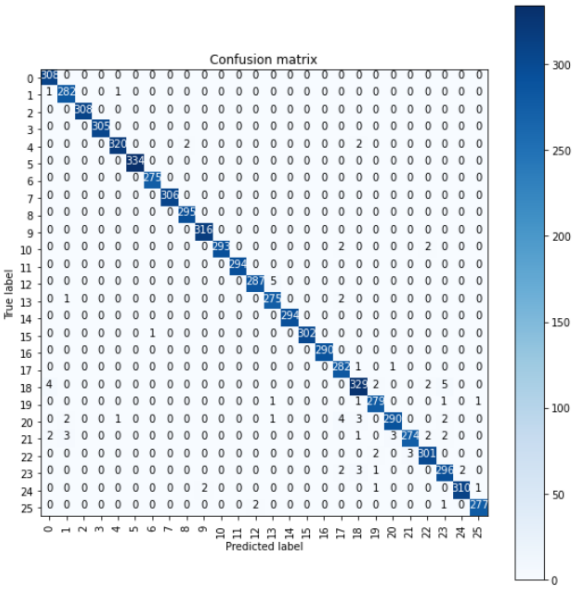


Model 3: Using VGG16 with 1500 images per letter and 26 letters and 5 EPOCH
(running for about 1 hour)

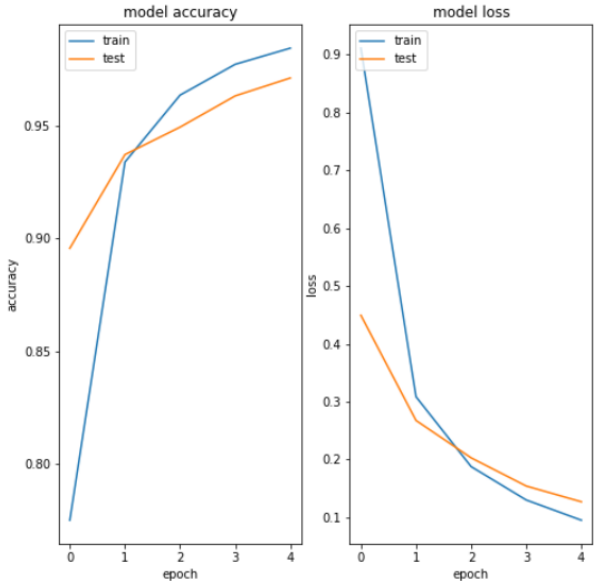


Keras CNN - accuracy: 0.9892390370368958

	precision	recall	f1-score	support
0	0.98	1.00	0.99	308
1	0.98	0.99	0.99	284
2	1.00	1.00	1.00	308
3	1.00	1.00	1.00	305
4	0.99	0.99	0.99	324
5	1.00	1.00	1.00	334
6	1.00	1.00	1.00	275
7	1.00	1.00	1.00	306
8	0.99	1.00	1.00	295
9	0.99	1.00	1.00	316
10	1.00	0.99	0.99	297
11	1.00	1.00	1.00	294
12	0.99	0.98	0.99	292
13	0.98	0.99	0.98	278
14	1.00	1.00	1.00	294
15	1.00	1.00	1.00	303
16	1.00	1.00	1.00	290
17	0.97	0.99	0.98	284
18	0.97	0.96	0.96	342
19	0.98	0.99	0.98	283
20	0.99	0.96	0.97	303
21	0.99	0.95	0.97	287
22	0.98	0.98	0.98	306
23	0.96	0.97	0.97	304
24	0.99	0.99	0.99	314
25	0.99	0.99	0.99	280
accuracy			0.99	7806
macro avg	0.99	0.99	0.99	7806
weighted avg	0.99	0.99	0.99	7806

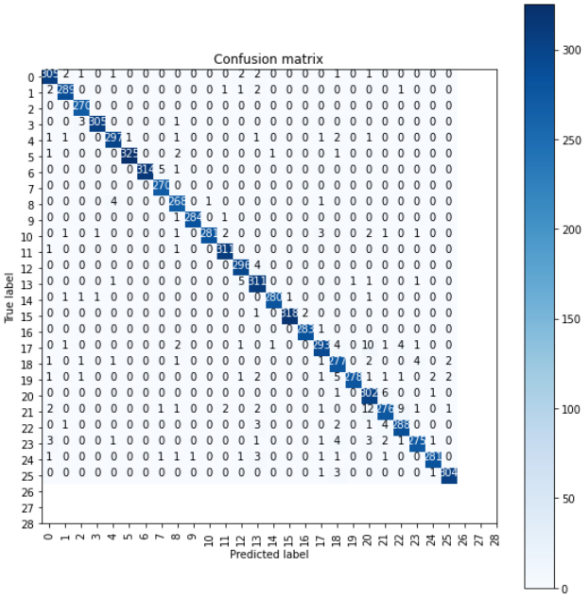


Model 4: Using INCEPTIONV3 with 1500 images per letter and 26 letters and 5 EPOCH
(running for about 1 hour)

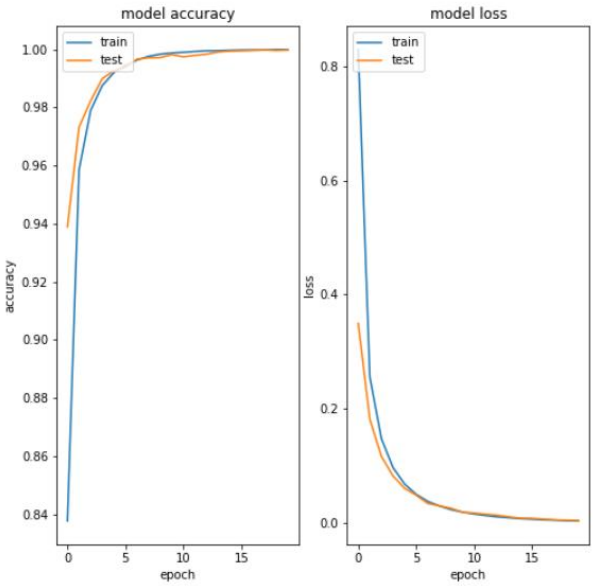


Keras CNN - accuracy: 0.971176028251648

	precision	recall	f1-score	support
0	0.96	0.97	0.96	315
1	0.98	0.98	0.98	296
2	0.97	1.00	0.99	270
3	0.99	0.99	0.99	309
4	0.97	0.97	0.97	306
5	1.00	0.98	0.99	330
6	1.00	0.98	0.99	320
7	0.97	1.00	0.99	270
8	0.95	0.98	0.97	274
9	1.00	0.99	0.99	286
10	1.00	0.96	0.98	293
11	0.98	0.99	0.99	313
12	0.96	0.99	0.98	300
13	0.94	0.97	0.95	320
14	0.99	0.98	0.99	285
15	1.00	0.99	0.99	321
16	0.99	1.00	0.99	284
17	0.96	0.92	0.94	318
18	0.92	0.96	0.94	290
19	1.00	0.94	0.97	296
20	0.90	0.97	0.93	310
21	0.95	0.90	0.92	308
22	0.95	0.96	0.96	299
23	0.97	0.94	0.96	292
24	0.98	0.96	0.97	292
25	0.98	0.98	0.98	309
accuracy			0.97	7806
macro avg	0.97	0.97	0.97	7806
weighted avg	0.97	0.97	0.97	7806

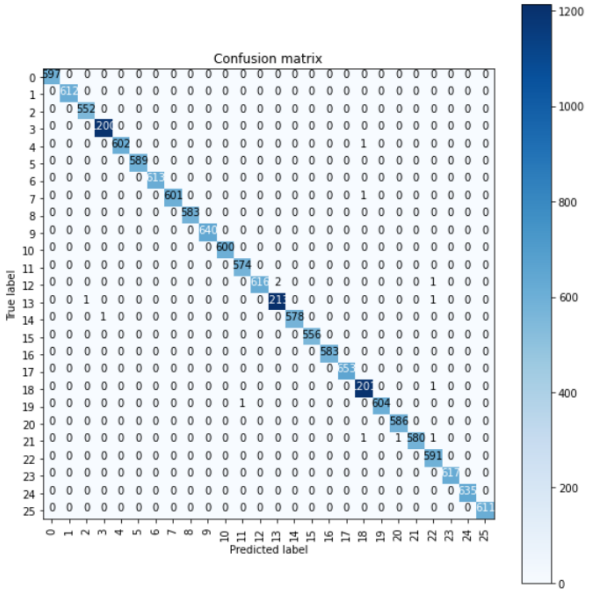


Model 5: Using VGG16 with 3000 images per letter and 26 letters and 20 EPOCH
(running for about 10 hours)

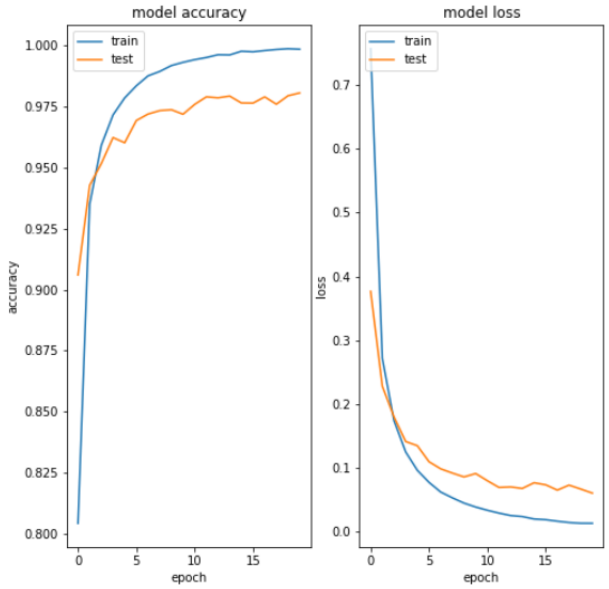


Keras CNN - accuracy: 0.9996795058250427

	precision	recall	f1-score	support
0	1.00	1.00	1.00	610
1	1.00	1.00	1.00	610
2	1.00	1.00	1.00	605
3	1.00	1.00	1.00	562
4	1.00	1.00	1.00	596
5	1.00	1.00	1.00	623
6	1.00	1.00	1.00	602
7	1.00	1.00	1.00	587
8	1.00	1.00	1.00	561
9	1.00	1.00	1.00	651
10	1.00	1.00	1.00	621
11	1.00	1.00	1.00	573
12	1.00	1.00	1.00	592
13	1.00	1.00	1.00	593
14	1.00	1.00	1.00	617
15	1.00	1.00	1.00	605
16	1.00	1.00	1.00	623
17	1.00	1.00	1.00	585
18	1.00	0.99	1.00	573
19	1.00	1.00	1.00	581
20	1.00	1.00	1.00	639
21	1.00	1.00	1.00	583
22	0.99	1.00	1.00	579
23	1.00	1.00	1.00	618
24	1.00	1.00	1.00	592
25	1.00	1.00	1.00	619
accuracy			1.00	15600
macro avg	1.00	1.00	1.00	15600
weighted avg	1.00	1.00	1.00	15600



Model 6: Using INCEPTIONV3 with 3000 images per letter and 26 letters and 20 EPOCH
(running for about 1 hour)



Keras CNN - accuracy: 0.980512797832489

	precision	recall	f1-score	support
0	0.96	0.98	0.97	601
1	0.98	0.98	0.98	592
2	1.00	0.99	1.00	606
3	1.00	0.98	0.99	649
4	0.96	0.97	0.97	594
5	1.00	1.00	1.00	574
6	0.99	0.99	0.99	589
7	1.00	0.99	1.00	613
8	0.98	0.96	0.97	627
9	0.99	0.98	0.99	567
10	0.97	0.99	0.98	613
11	1.00	1.00	1.00	602
12	0.99	0.98	0.98	595
13	0.98	0.99	0.99	612
14	1.00	0.98	0.99	599
15	0.98	0.99	0.99	589
16	0.99	0.99	0.99	623
17	0.95	0.97	0.96	603
18	0.97	0.97	0.97	620
19	0.98	0.98	0.98	594
20	0.96	0.95	0.96	605
21	0.97	0.95	0.96	580
22	0.96	0.97	0.97	594
23	0.95	0.98	0.96	589
24	0.99	0.97	0.98	581
25	0.99	1.00	0.99	589
accuracy			0.98	15600
macro avg	0.98	0.98	0.98	15600
weighted avg	0.98	0.98	0.98	15600