TITLE: MULTICLASS CLASSIFICATION OF AMERICAN SIGN LANGUAGE(ASL)

(Abhishek Maganalli, Aditya Patel, Gurukiran Reddy Kasireddy, Shweta Lal)

Keywords: Exploratory Data Analysis, Image Classification, Multi-Layer Perceptron, CNN

Abstract: American Sign Language (ASL) is a complex and expressive visual-gestural language used primarily by the Deaf and hard-of-hearing communities in the United States of America. To successfully convey information and communicate, ASL uses a wide range of hand forms, face expressions, and body movements. However, for those not proficient in ASL, understanding and interpreting these signs can be challenging. This is where machine learning (ML) and computer vision techniques come into play. The multiclass classification of American Sign Language using machine learning is an exciting and practical application of artificial intelligence that aims to bridge the communication gap between the Deaf community and the hearing world. This project involves training a machine learning model to recognize and interpret the various signs and gestures used in ASL. The primary objective is to enable automatic translation of sign language into text or spoken language, making it more accessible and inclusive for individuals who do not know ASL.

Dataset: https://www.kaggle.com/datasets/lexset/synthetic-asl-alphabet

Links: **Code**: Link_To_Code
Milestone1: Synthetic ASL Alphabets
Milestone3: Literature Review

1.INTRODUCTION AND PROJECT STRUCTURE

We divided the tasks for this project into the following parts. This allowed us to tackle tasks in an efficient way and excel at each part individually.
I. Exploratory Data Analysis: Tried to understand the image dataset and how to proceed with that in order to use it efficiently. This step was particularly important because the chosen dataset is an image dataset and was too large in size.
II. "Simple" Classifiers + Performance Assessment: With the information found in the EDA, we chose to apply the CNN model on image dataset.
III. Live Testing + Performance Assessment: We pass the processed images into our model to find the probabilities.
IV. Performance Assessment: Based on the accuracy we were getting, we tried to change the number of epochs and some other modifications, which led us to get better accuracy of our model.
V. Conclusion: We summarized the findings made in our project along with the overall results. Furthermore, we went over limitations and discussed possible future improvements.

2. RELATED WORKS AND REFINEMENT

Before starting our project, we went through previous studies performed by other people. All of them had a similar structure: started with some brief data analysis and then built a multi-class classifier by removing the <blank> class. But most of the users in kaggle were using transfer learning algorithms like ResNet, MobileNet with excellent accuracies (96% - 98%), but we found that most of the notebooks didn't test it with real world data, in some notebooks they have done some tests but ended up with very bad accuracy. As our dataset was synthetic, we really wanted to check how it would predict actual data. This is the reason we decided to work with GTTS and Audio Stream to go an extra mile ahead.

In Milestone 3, we particularly focused on the literature review[2.1]

3. EXPLORATORY DATA ANALYSIS

We looked into all the images in the dataset:

3.1 Data Exploration

The original dataset comprises 27,000 RGB images from the American sign language each of dimension 512 x 512. There are 27 classes: 26 for alphabet and one for blank. The dataset is already subdivided into train and test folders where in the train set there are 900 images for each class and 100 images in the test set. Each image size is 250 kb. But due to limited resources and machine incapability we reduced the classes to 17 alphabets/classes only i.e. ['A' ,'B' ,'C' ,'D' ,'E' ,'G' ,'I' ,'L' ,'N' ,'R' ,'T' ,'U' ,'V' ,'W' ,'X' ,'Y' ,'Z'].

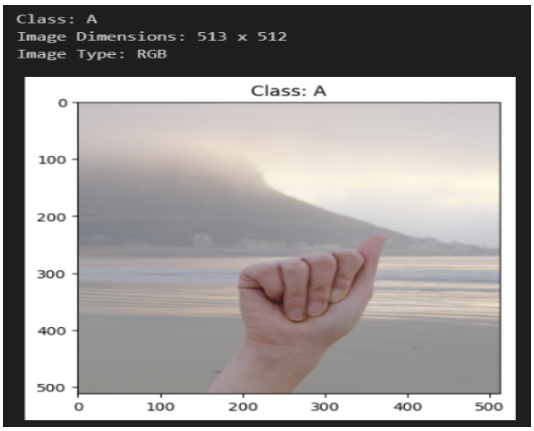

Fig1. Analysis of the Image

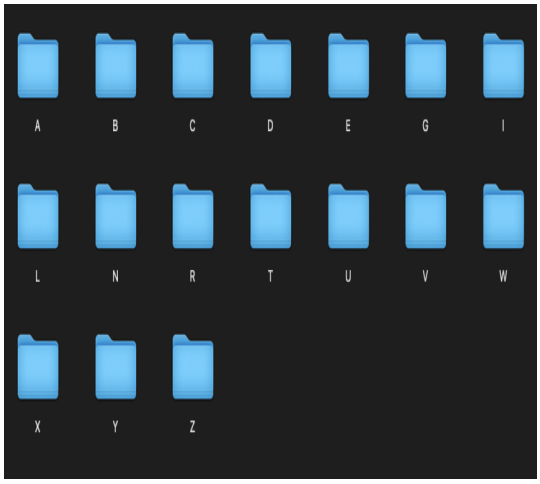

Fig2. Chosen 17 classes

3.2 Data Pre-Processing

Originally the images were RGB and contained a background which created overhead while preprocessing them. We had to scale the image so that we could train the model within our machine capabilities.

The following techniques were used to resize the images

- Re-scale the dimensions of the images from 512 X 512 to 256 X 256
- Normalize the data to change the range of pixel intensity values between 0 and 1
- Converted the images into grayscale
- Removed background from the images using **Rembg library**
- Removed inherent noise from the images (salt and pepper noise)

This preprocessing was vital in our model building process as it reduced the size of the images by 80%.

Fig3. Before Preprocessing the images

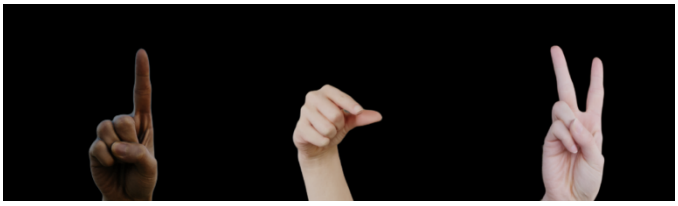

Fig4. After Preprocessing the images

4. CLASSIFICATION MODELS AND ASSESSMENT

The steps we followed on this part are influenced by the discoveries we made during the exploratory data analysis. In the EDA, we drew conclusions about our image dataset and pre-processed it. In this section, we worked on applying a machine learning model to 17 classes and saw the outcomes.

4.1 Model Building

We have built a Convolutional Neural Network (CNN) with three hidden layers. Each hidden layer has ReLU as its activation function to introduce nonlinearity to learn complex patterns from the images. The activation function for the output layer is Softmax which converts the raw output scores from the network into probability distributions over multiple classes. We have introduced MaxPooling2D to reduce the spatial dimensions of the input data for ease of computation. We have also added Dropout Regularization technique to improve the model's generalization performance and prevent overfitting. When you compile a model, you specify various settings that determine how the training process should behave. This includes the choice of optimizer, the loss function, and the metrics used to evaluate the model's performance. In this model we have used Adam as the optimizer, Categorical cross entropy as the loss function and Accuracy as the metric.

```python
# Function to build the CNN model
def build_cnn_model(num_classes):
    model = models.Sequential()
    model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=input_shape))
    model.add(layers.MaxPooling2D((2, 2)))
    model.add(layers.Conv2D(64, (3, 3), activation='relu'))
    model.add(layers.MaxPooling2D((2, 2)))
    model.add(layers.Conv2D(128, (3, 3), activation='relu'))
    model.add(layers.MaxPooling2D((2, 2)))
    model.add(layers.Flatten())
    model.add(layers.Dense(128, activation='relu'))
    model.add(layers.Dropout(0.5))  # Adding dropout for regularization
    model.add(layers.Dense(num_classes, activation='softmax'))

    return model

# Function to compile the model
def compile_model(model):
    model.compile(optimizer='adam',
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])
```

Fig5. Model Architecture

4.2 Hyperparameter Tuning and Evaluation

In Hyperparameter tuning we mainly are focusing on changing the number of epochs for which our model will run. We have used a constant learning rate of 0.001 as we were using ADAM optimizer. Due to heavy data being fitted in our systems, we were

facing issues in making the model run for a greater number of epochs. We ultimately ended up on the number 17 where our model ran successfully without crashing and delivered a desirable result. As our model was crashing numerous times, we couldn't focus on other approaches (as mentioned by professor we could have used decaying LR/cyclic LR). The graph below shows that against epochs our loss is exponentially decreasing and imagining that if we ran our model for a greater number of epochs, the loss would decrease more (in about 20-25 epochs).

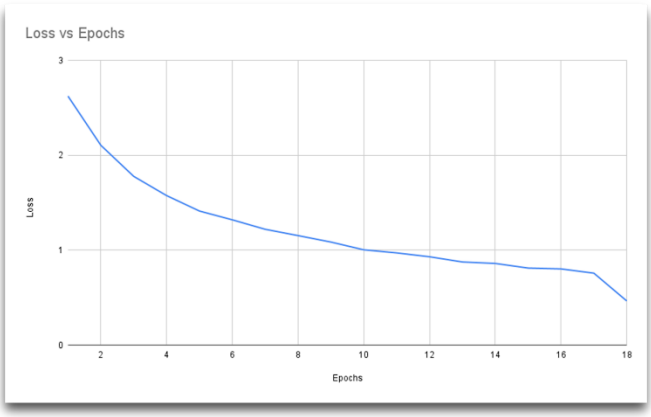

Fig6. Loss vs Epoch curve

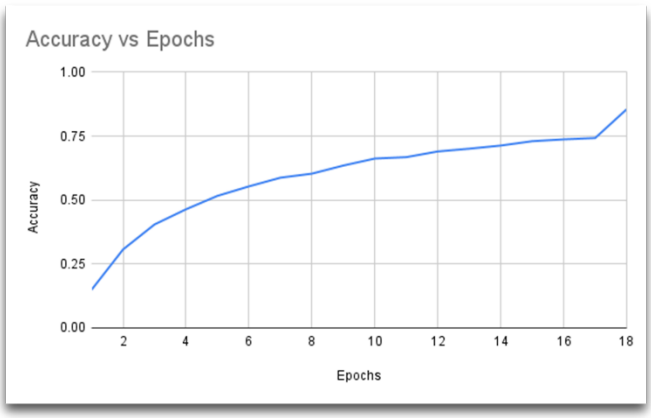

Fig7. Accuracy vs Epoch curve

Accuracy was another such thing we were striving for. Previously, we created a model and ran it for all the 27 classes for about 10 epochs, which gave us a mere 4% accuracy (without any preprocessing of images like background removal / resizing the image). Later after implementing certain changes the model ran successfully for 17 epochs with an accuracy of 85%. We have plotted a graph based on these values against the number of epochs.

4.3 GTTS

GTTS (or Google Text to Speech) was used in our project to make it simpler and easier for users to make out the alphabets. It is a Python library and CLI tool to interface with Google Translate's text-to-speech API. When we input an image of a signed alphabet to our model, it will predict the output and print the alphabet on the screen. This helps if the user is having difficulty in reading the alphabets presented. In order to take our project to the next level, we have used GTTS to deliver our output in the form of audio, which makes it reach a bigger audience. To deliver our output as an audio stream we have used iPython's Audio library[4.3.1] to create an audio stream which can be playable in the output window. By just a click of play button our predicted class would be in audio format. This can further be developed more and can be automated with better User Interface and maybe training a more diverse dataset which includes words and phrases.

5. RESULTS

After training our model, we are trying to input an image from our default camera and sending that as input to our model. Our model pre-processes the image, tries to find what the signed alphabet is and predicts the output alphabet in text and audio format. Using GTTS we are trying to communicate the output in audio for a larger audience.

Input Image(from webcam):

Fig8. Image captured from webcam

After Pre-processing:

Fig9. Preprocessed Image from the webcam

Output:

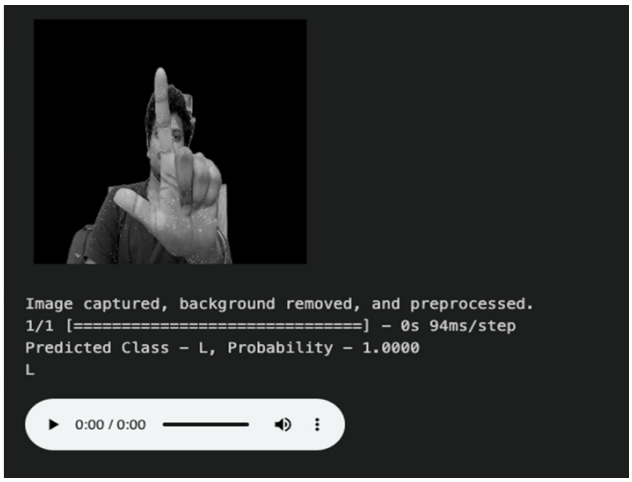

Fig10. User input and model prediction

The signed alphabet in the above image decodes to the alphabet 'L'. Our model is able to predict the output, also when we play the audio, we can hear that our model has predicted the letter 'L' accurately.

5. FUTURE IMPLEMENTATIONS

Our project is trying to convert the signed alphabet images into text/audio. We have used a default camera to upload the images. But, in future we can implement our project along with leading technologies like AiPin, Smart Glasses, or Snapchat lenses. It can be implemented along with smart wearables.

6. CONCLUSIONS

In conclusion, our project is trying to address the challenge of multi-class classification within the realm of American Sign Language (ASL). We have tried to address this by using Machine Learning algorithms. We have worked with CNN, which is showcasing results by accurately interpreting and classifying a set of American sign language gestures. The model's adaptability to various signed alphabets in our dataset establishes a strong foundation for its practical application in real world scenarios. While our project is able to detect various classes, there is still room for improvement. Our project represents a stepping stone towards breaking down communication barriers for individuals with hearing impairments.

7. REFERENCES

[4.3.1]
https://colab.research.google.com/drive/1wMg9ZV2WH2ugAC-6iZLUkEH3V6XxI3H-#scrollTo=9cXLNvYv7LYz

GitHub Link: https://github.com/AdityaPatel1068/ASL

Research Papers:
https://ieeexplore.ieee.org/abstract/document/9399594

https://scholar.google.com/scholar?hl=en&as_sdt=0%2C31&q=American+Sign+Languae++ml&btnG=

A Review on Sign Language Recognition (SLR) System: ML and DL for SLR | IEEE Conference Publication | IEEE Xplore
https://primo.njit.edu/discovery/search?query=any,contains,AMERICAN%20SIGN%20LANGUAGE%20MACHINE%20LEARNING%20MODEL&tab=Everything&search_scope=MyInst_and_CI&vid=01NJIT_INST:NJIT&mfacet=rtype,include,articles,1&mfacet=rtype,include,dissertations,1&mfacet=rtype,include,conference_proceedings,1