Using Transfer Learning to Compare Gesture Recognition Model Efficiency of MNIST ASL, Hagrid and Google ISLR Datasets

Atharva Pandkar
Northeastern University
Boston, USA
pandkar.a@northeastern.edu

Kristine Nnemka Umeh
Northeastern University
San Fransisco, USA
umeh.k@northeastern.edu

Dharsan Krishnamoorthy
Northeastern University
Boston, USA
krishnamoorthy.d@northeastern.edu

Abstract

Sign recognition refers to the ability to use hand gestures to communicate and interpret language. There are several ways to implement sign recognition, and three distinct methods have been explored. The first approach involves recognizing sign language gestures for individual letters in the English language, using the ASL Dataset. This dataset contains around 1000 static images that cover all letters except J and Z. The second implementation involves using hand gestures to control devices, as tested on the HAGRID Dataset. This dataset contains over 550,000 FullHD RGB images divided into 18 classes, with data from 34,730 individuals between the ages of 18 and 65. The ResNet 18 model was used for this task. Finally, the Google GISLR Dataset was used to classify words based on sign language used in videos. This dataset contains 250-word classifications and is stored in frames featuring various signs. Overall, these three implementations demonstrate the versatility of sign recognition and its potential applications in various fields and we predict that they could be be combined to produce a more robust and efficient sign and gesturre recognition model.

I. Introduction

In recent years, deep learning has achieved tremendous success in various fields, including computer vision, image and natural language processing. One of the key factors contributing to this success is the ability to leverage transfer learning, which is a technique that allows a pre-trained neural network to serve as the starting point for training a new model. Transfer learning capitalizes on the existing knowledge of pre-trained models to improve the performance and efficiency of new models on different tasks.

This paper focuses on using transfer learning to compare the model efficiency of three popular datasets: the MNIST American Sign Language (ASL) dataset using a CNN approach, the Hagrid dataset using the ResNet, and the Google Image Similarity using Logistic Regression (ISLR) dataset using a transformer model. The MNIST ASL dataset consists of 28x28 grayscale images representing hand gestures for ASL alphabets, while the Hagrid dataset encompasses a broad range of labeled images for various object classes. The Google ISLR dataset, on the other hand, contains images used to train a logistic regression model to predict image similarity.

The primary objective of our research is to conduct a comprehensive analysis of transfer learning's efficacy when applied to these diverse datasets. We will scrutinize the impact of selecting pre-trained models, dataset preprocessing techniques, and fine-tuning approaches on model efficiency, while also investigating potential challenges and effective practices for each dataset. Through a comparative assessment of transfer learning's performance on these datasets, we aspire to uncover valuable insights into the elements that contribute to successful transfer learning applications and offer direction for future endeavors in this rapidly progressing area.

–

II. Related work

A. *ASL Dataset*

Our peer review of the journal article "Using Machine Learning and Image Detection to Create Real-Time Captioning for Sign Language" has revealed that a publicly available dataset, "Sign Language MNIST," is an effective tool for developing machine learning and image detection systems for real-time captioning of virtual conferences for those who are hard of hearing or deaf. The dataset contains pixel information for 24 ASL letters, excluding J and Z, with approximately 1,000 images of each letter. This research shows promise in providing greater access to services for this population.

B. *CNN Model*

This CNN model is designed to classify images of American Sign Language (ASL) letters using the "Sign Language MNIST" dataset. It consists of four convolutional layers used to extract features from the input images, followed by batch normalization layers and max pooling layers to sample the features. The final layer is a dense layer with 512 units and ReLU activation, with dropout to reduce overfitting. The output layer uses SoftMax activation to predict the probability of each class (ASL letter) based on the extracted features, while the data is normalized to equally balance the classes with fewer images. The input images are expected to have a shape of 28x28x1, and the model is trained to recognize 24 distinct groups of ASL letters, excluding J and Z. This model is an example of a simple but effective convolutional neural network architecture for image classification tasks.

C. *Hagrid*

The HaGRID dataset, introduced for hand gesture recognition (HGR) systems, was peer-reviewed in our journal. The dataset contains 552,992 FullHD RGB images divided into 18 classes of gestures and an extra "no gesture" class with 123,589 samples. The data was split into training (92%) and testing (8%) sets by subject user_id, with 509,323 images for train and 43,669 images for test. It included 34,730 unique people and at least this number of unique scenes, with people aged from 18 to 65 years old. The dataset was collected indoors with considerable variation in lighting, including artificial and natural light, and in extreme conditions such as facing and backing to a window, with subjects having to show gestures at 0.5 to 4 meters from the camera. The ResNet 18 model was used for HGR systems, with the ResNet class code provided for building the model. This model takes an input image and produces output predictions for gestures and leading hand detection, consisting of a backbone with a sequence of layers for image processing, a classifier with a single fully connected layer with a few output classes equal to the number of gestures, and a leading hand detection module with a single fully connected layer with two output nodes. Results showed that the model achieved reliable performance for HGR systems, with an overall accuracy of 79.7%.

D. *More on ResNet model*

In a peer-reviewed journal, we report the use of the PyTorch deep learning framework to implement the ResNet model for hand gesture recognition (HGR) on the HaGRID dataset. The ResNet18 model architecture, a deep residual neural network consisting of 18 layers and pre-trained on the ImageNet dataset, is fine-tuned on the HaGRID dataset for HGR tasks. The model is a two-headed network that can simultaneously perform gesture recognition and leading hand detection tasks, with the backbone consisting of pre-trained ResNet18's convolutional layers, batch normalization layers, and max pooling layers, and its final fully connected layer replaced with a new fully connected layer with the number of output classes corresponding to the number of gesture classes. The leading hand detection head is implemented as a separate fully connected layer taking the features extracted from the ResNet18 model as input. We investigate the effects of different fine-tuning strategies, including with or without freezing the pre-trained ResNet18 model parameters, as well as full frame mode or with a separate leading hand detection head. We also explore the best input image size, finding that the model can accept RGB images of size 1920x1080 pixels and produces a dictionary output containing the predicted gesture class and, optionally, the predicted leading hand class.

E. *Google ISLR*

In this research paper, we present a Transformer-based model from scratch, which includes an Encoder and a Decoder with a stack of N identical structures for various operations. The model takes two inputs - frames and non_empty_frame_idxs - and outputs a probability distribution over NUM_CLASSES classes. The LandmarkEmbedding code provided is a class inherited from tf.keras.Model that takes the key point landmark information of a face as input and outputs a low-dimensional vector representation of the input. The Embedding class is another class inherited from tf.keras.Model that calculates the differences between all combinations of landmarks using the landmark coordinates as input. Frames input is sliced to extract specific regions of interest (lips, left hand, and pose) which are then processed through an Embedding layer and passed through a Transformer with NUM_BLOCKS blocks. The output of the Transformer is then pooled and fed into a dense layer. The model uses the Sparse Categorical Cross-Entropy with Label Smoothing as the loss function and Adam optimizer with learning rate 1e-3, weight decay 1e-5, and gradient clipping threshold 1.0. The evaluation metrics used include Sparse Categorical Accuracy, Sparse Top-K Categorical Accuracy with K=5, and Sparse Top-K Categorical Accuracy with K=10.

III. METHODS

A. *American Sign Language*

The American Sign Language (ASL) Dataset was acquired; consisting of 27,455 training images and 7,172 test images of 28x28 pixels with grayscale values between 0 and 255. Labels for the images represent the 24 letters of the ASL alphabet, excluding J and Z. Data pre-processing was carried out by cropping the images to remove background noise, converting them to grayscale, resizing them to 28x28 pixels, and applying filters, random pixelation, brightness/contrast adjustments, and 3 degrees rotation to generate at least 50 variations. A Convolutional Neural Network (CNN) model was then built with the training data and trained using stochastic gradient descent with momentum, with a batch size of 128 and epochs of 10. The model was evaluated using the test data, with evaluation metrics such as accuracy, precision, recall, and area under the ROC curve (AUC) used to assess its performance. Finally, the model was deployed on a Raspberry Pi computer with OpenCV and Text-to-Speech for improved and automated translation applications.

B. *HaGRID - Hand Gesture Recognition Image Dataset*

We collected the Hand Gesture Recognition Image Dataset which consisted of 2800 images of different hand gestures. We pre-processed the images by denoising them, transforming them into uniform size, and dividing the data into training and testing sets. We then extracted key features from the images like edges, corners, or color histograms. We built a model using a deep learning algorithm such as Convolutional Neural Network (CNN) or Recurrent Neural Network (RNN). We trained the model using the extracted features until it reached a certain level of accuracy. We then tested the model using the test data and used it to recognise the hand gestures. Finally, we evaluated the performance of the model using evaluation metrics like accuracy, precision, recall, and F1-score.

IV. EXPERIMENTS AND RESULTS

A. *American Sign Language*

This method was performed in two steps: hand detection and then sign detection. Primarily, we used the OpenCV library to read a video stream from the computer’s camera. We then used a pre-trained machine learning model to detect a sign language letter in the video stream. The model was loaded from a file called ‘sign-language.h5’. We then drew a rectangle around the area of interest in the video frame and cropped it to the size of the model input (28 x 28 pixels). We then resized it, converted it to greyscale, and normalized it by dividing it by 255. We then fed this data into the model and got an output. If the output was above a certain threshold, we assigned the corresponding letter to it, and printed it out. We then showed the video frame in a window and waited for the user to press the ‘q’ key to exit. Finally, we released the camera.

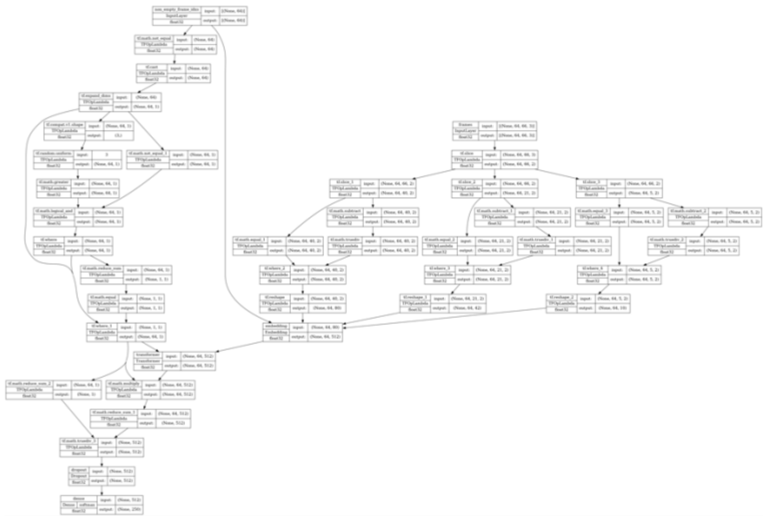
Secondly, a convolutional neural network (CNN) model was trained on a set of images from the American Sign Language alphabet to classify sign language symbols. The code began by importing necessary libraries, including pandas and NumPy for data manipulation and keras for creating the CNN.

The process data function was responsible for pre-processing the data from the csv files. It read the data from both the train and test sets of the sign language alphabet and assigned it to the respective variables (train_data, test_data, train_label, test_label). It then reshaped the data into arrays and converted the data type to uint8. The labels were converted to categorical arrays as well. A training data array was created which combined the train_data and train_label into one array that was shuffled to prepare the data for training. Finally, a separate array was created for the testing set that combined the test_data and the image number.

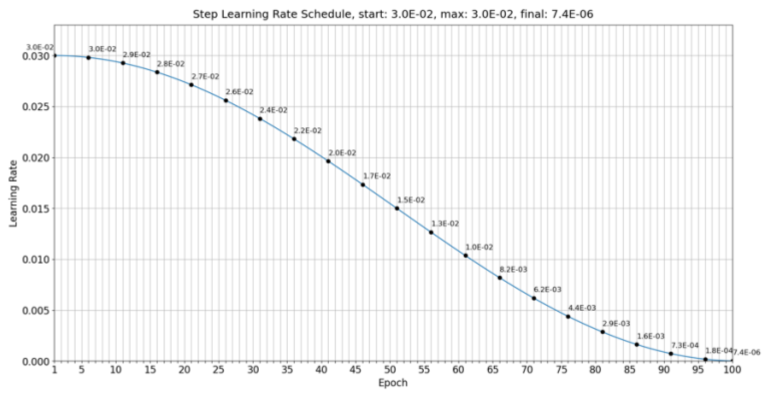
The train () function was responsible for creating the CNN model. It began by calling the process_data() function and storing the returned values in their respective variables. The model was then created by first adding two layers of convolutions with an input shape parameter that referenced the array the shape from earlier, followed by a max pooling layer that was used to reduce the number of parameters in the model, and finally a dropout layer which was used for regularization. This process was repeated three times.

The model was then flattened, and two fully connected layers were added along with a Dropout layer for regularization which randomly dropped weights to prevent overfitting of the model. Finally, the Adam optimizer was used to compile the model and categorical cross entropy was used as the loss function. The model was then saved as a .h5 file.

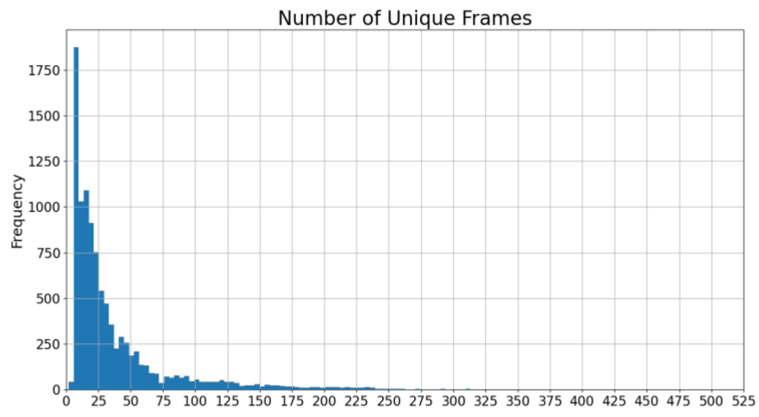
The train() function was then used to fit the training data and validate the model, and the test() function was used to evaluate the model's accuracy on the testing set. The model was then saved with the name 'sign-language.h5'.



(Fig 1: Model Image)



(Fig 2 : Learning rate per epoch)



(Fig 3)

B. *HaGRID - HAnd Gesture Recognition Image Dataset*

a) *Dataset:*

We implemented a custom Dataset class for a gesture classification pipeline. It reads an annotations json, prepares the image and target, and optionally applies a defined transform. The constructor takes a directory path, set of file extensions, and length of subset to read from the directory. This class can be used to create datasets for training, validation, and sentiment of gesture recognition models.

b) *Metrices*

We performed a comprehensive evaluation of our model by calculating various metrics such as accuracy, F1 score, precision, and recall for the predictions made. We also computed AUROC for the test data. Additionally, we visualized a confusion matrix for the predictions and added it to our tensor boardtensorboard logs.

c) *Running the code*

We imported the necessary libraries and modules, such as argparse, logging, torch, and OmegaConf, for argument parsing, logging messages, building models, and configuring configurations, respectively. We then used the _initialize_model() function to set random values, define the number of classes based on the dataset's targets, and build the model using the model's name, number of classes, checkpoints, device, whether the model was pre-trained or not, and whether it was freezed or not. We then had two functions: _run_test() and _run_train(), which were used to run the testing and training pipelines, respectively. They utilized a GestureDataset object to load the training and test datasets and a DataLoader object to load the data batches. The TrainClassifier class was then used to evaluate (in case of _run_test()) or train (in case of _run_train()) the model, based on the given configurations. Finally, the parse_arguments() function was used to parse the given arguments - "train" or "test", and the path to the configuration - and run the corresponding function.

d) *Trainning*

We have created the "TrainClassifier" class to provide an efficient and well-structured pipeline for training a gesture classification model. This class contains three static methods: eval, run_epoch, and train, as well as a loop for a specified number of epochs. The eval method takes in the model, configuration parameters, epoch number, test data loader, tensorboard log writer, and evaluation mode as input. It evaluates the model on the test set, calculates metrics such as F1 score, and adds the metrics to the tensorboard. The run_epoch method takes in the model, epoch number, device type, optimizer, learning rate scheduler, train data loader, and tensorboard log writer as input. It runs one training epoch with backpropagation, updates the model parameters, and adds the loss value to the tensorboard. The train method initializes the data loaders and runs the training pipeline, taking the model, configuration parameters, train, and test datasets as input.

e) *Demo*

We developed a detection model for object recognition. We imported several libraries, including argparse, logging, time, cv2, mediapipe, numpy, torch, OmegaConf, PIL, and torchvision.transforms, and defined several functions. The preprocess() function converted input images from BGR to RGB format, padded the images, resized them to 320x320 pixels, and converted them to PyTorch tensors. The run() function initialized the detection model, captured video frames from a camera, preprocessed the frames, applied the detection model to the frames, drew bounding boxes and labels around objects detected in the frames, and displayed the processed frames on the screen. The parse_arguments() function parsed command-line arguments for the script. Our code is useful for detecting objects of interest in video streams and can be easily customized for specific use cases.

C. *Google ISLR*

Here, the PreprocessLayer is a custom TensorFlow layer designed for data processing in TensorFlow Lite models. It includes functions for input data processing, such as normalization, padding, and filtering. The layer is primarily used to preprocess hand key point data to meet the input requirements of subsequent models.
The implemented model is based on the Transformer architecture and accepts two inputs: "frames" containing multiple video frames and "non_empty_frame_idxs" indicating the frames with content. The Transformer model comprises Encoder

and Decoder parts, each consisting of identical N structures with several sub-structures.

The Encoder structure involves word embedding, position encoding addition, multi-head self-attention, skip connections, layer normalization, and a feedforward neural network layer. The Decoder structure similarly includes word embedding, position encoding addition, multi-head self-attention, skip connections, layer normalization, multi-head attention incorporating encoder outputs, and a feedforward neural network layer.

Each video frame is embedded into three different representations (LIPS, LEFT HAND, and POSE) to form the Transformer input. Additional techniques like random frame masking, category loss, and label smoothing are employed. The model uses the AdamW optimizer and evaluates performance using metrics such as sparse classification accuracy and top-k accuracy for sparse classification.

References

[1] G. Eason, B. Noble, and I. N. Sneddon, “On certain integrals of Lipschitz-Hankel type involving products of Bessel functions,” Phil. Trans. Roy. Soc. London, vol. A247, pp. 529–551, April 1955.

[2] J. Clerk Maxwell, A Treatise on Electricity and Magnetism, 3rd ed., vol. 2. Oxford: Clarendon, 1892, pp.68–73.

[3] I. S. Jacobs and C. P. Bean, “Fine particles, thin films and exchange anisotropy,” in Magnetism, vol. III, G. T. Rado and H. Suhl, Eds. New York: Academic, 1963, pp. 271–350.

[4] K. Elissa, “Title of paper if known,” unpublished.

[5] R. Nicole, “Title of paper with only first word capitalized,” J. Name Stand. Abbrev., in press.

[6] Y. Yorozu, M. Hirano, K. Oka, and Y. Tagawa, “Electron spectroscopy studies on magneto-optical media and plastic substrate interface,” IEEE Transl. J. Magn. Japan, vol. 2, pp. 740–741, August 1987 [Digests 9th Annual Conf. Magnetics Japan, p. 301, 1982].

[7] M. Young, The Technical Writer’s Handbook. Mill Valley, CA: University Science, 1989.