SignVision: Real-time American Sign Language Recognition using Deep Learning and Computer Vision

Abinav Anantharaman
College of Engineering
Northeastern University
Boston, USA
anantharaman.ab@northeastern.edu

Satwik Shridhar Bhandiwad
Khoury College of Computer Sciences
Northeastern University
Boston, USA
bhandiwad.s@northeastern.edu

***Abstract*— American Sign Language (ASL) is a visual language used by the deaf and hard-of-hearing community to communicate. In this project, we propose the development of "SignVision," a real-time ASL recognition system that leverages deep learning and computer vision techniques. Our proposed system uses a CNN-based model with transfer learning on a large dataset of ASL images, employing a pre-trained ResNet18 architecture. We compared our proposed system with different custom models to ensure its effectiveness and found that our system outperformed the other models with improved accuracy and real-time performance. SignVision has the potential to facilitate communication between the hearing-impaired and hearing individuals and enable the development of ASL-based virtual assistants, real-time translation systems, and accessibility tools. Our project aims to leverage the power of deep learning and computer vision to build an accurate and efficient ASL recognition system with real-world applications for the deaf and hard-of-hearing community.**

Keywords—American Sign Language, ResNet, Convolutional Neural Networks, Deep Learning, Computer Vision

I. INTRODUCTION

The task of recognizing ASL gestures involves identifying and classifying various hand and arm movements which can be complex and diverse. Deep learning models, particularly Convolutional Neural Networks (CNNs), have shown promising results in image and video classification tasks, including ASL recognition. Transfer learning, which involves using pre-trained models to improve performance on smaller datasets, has also been shown to be effective in ASL recognition.

In this context, we propose the development of a robust ASL recognition system called "SignVision." SignVision utilizes computer vision techniques and deep learning architectures to recognize ASL gestures in real-time with high accuracy. Our proposed system employs a CNN-based model with transfer learning on a pre-trained ResNet18 architecture, which outperforms other custom models. ResNet18 has been shown to be effective in various image classification tasks, including ASL recognition, due to its ability to learn more complex features and its ability to prevent overfitting.

In this project, we aim to classify 29 classes of American Sign Language (ASL) gestures from a dataset of 87,000 images. The classes include A to Z, SPACE, DELETE, and NOTHING. The images are originally of size 200x200 pixels, but for faster training, we resize them to 64x64 pixels and normalize them. We will perform all the necessary steps, from processing the dataset to building a neural network and predicting the class of test set images. The ultimate goal of this project is to develop an accurate and efficient ASL gesture recognition system that can facilitate communication between hearing-impaired and hearing individuals.

Fig 1: 26 Alphabets in ASL

To validate the performance of our model, we compared the results of SignVision with other custom models. Our experimental results demonstrate that our proposed system outperforms these models, achieving high accuracy and real-time performance. SignVision has the potential to facilitate communication between the hearing-impaired and hearing individuals and enable the development of various ASL-based applications.

II. Related work

A. Sign Language Recognition Using ResNet50 Deep Neural Network Architecture [1]

This paper discusses the development of a computer vision-based approach for sign language recognition using ResNet50 Deep Neural Network Architecture. The authors note that while several sensor-based systems exist for sign language recognition, they are often expensive and impractical for widespread deployment.

The authors used an image dataset for American Sign Language containing 60,336 image samples from 36 classes of 10 digits and 26 alphabets to evaluate their approach. They achieved an accuracy of 99.03% on the test set, demonstrating the effectiveness of software-based techniques for classification.

The paper provides a detailed explanation of the ResNet50 architecture and how it was adapted for sign language recognition. The authors also discuss the use of transfer learning, which involves using pre-trained models to improve performance on new tasks with limited data.

Overall, this paper highlights the potential of computer vision-based approaches for sign language recognition and demonstrates the effectiveness of ResNet50 Deep Neural Network Architecture in achieving high accuracy rates. The authors suggest that their approach could be used in a range of applications, including communication devices for individuals with hearing impairments and educational tools for learning sign language.

B. Deep Learning for Sign Language Recognition: Current Techniques, Benchmarks, and Open Issues [2]

This paper provides a comprehensive review of the current techniques, benchmarks, and open issues related to automated sign language recognition based on machine/deep learning methods and techniques. The authors attempt to answer three main research questions: which studies have been conducted addressing automated Sign Language Recognition (SLR), what techniques in Automatic Sign Language Recognition for various languages are applied to date, and which challenges remain unsolved in this scientific field.

The paper discusses various deep learning models that have been used for SLR, including convolutional neural networks (CNNs), recurrent neural networks (RNNs), and Residual Networks (ResNets). ResNets are a type of CNN that use residual connections to improve the performance of deep neural networks. The authors note that ResNets have shown promising results in SLR tasks, particularly when combined with other deep learning techniques such as attention mechanisms.

The paper also discusses various datasets that have been used for SLR research, including publicly available datasets such as RWTH-PHOENIX-Weather and American Sign Language Lexicon Video Dataset (ASLLVD), as well as proprietary datasets such as Saudi Sign Language Recognition Dataset (SSLRD) and Chinese Sign Language Recognition Dataset (CSLRD). The authors note that while these datasets have helped advance the field of SLR, there is still a need for more diverse and larger datasets to improve the accuracy and robustness of SLR models.

In addition to discussing current techniques and benchmarks, the paper also highlights several open issues and challenges in the field of SLR. These include the need for more standardized evaluation metrics, the need for more research on local-level sign languages, and the need for more research on multi-modal approaches that combine visual information with other modalities such as audio or haptic feedback.

Overall, this paper provides a comprehensive overview of current techniques, benchmarks, and open issues related to automated sign language recognition based on machine/deep learning methods and techniques. The authors highlight the potential of ResNets for improving the performance of SLR models and emphasize the need for more diverse and larger datasets to improve the accuracy and robustness of SLR models.

C. Classification of Sign-Language Using Deep Learning by ResNet [3]

This paper explores the classification of sign language using deep learning by ResNet. The authors begin by discussing the differences between American Sign Language (ASL) and British Sign Language (BSL) and the challenges of recognizing and classifying sign language gestures. They note that sign languages are complex visual languages consisting of coordinated hand gestures, body movements, and facial expressions, which can vary significantly across different cultures and regions.

The authors then describe their approach to classifying sign language gestures using deep learning algorithms based on ResNet architecture. They compare their results to previous studies that used VGG19 and mobileNet algorithms, finding that ResNet achieved a higher degree of accuracy (93.48%) with a dataset of 43,500 images at 64x64 pixels. They also split the data into training (70%), validation (15%), and testing (15%) datasets over 20 epochs.

The literature review in this work focuses not only on static sign language recognition systems but also on the use of different classifiers and their application in Machine Learning- and Deep Learning-based systems. The authors note that several researchers have worked in the field of sign language recognition over the last two decades, with varying degrees of success.

The authors conclude by discussing potential applications for their research in improving accessibility for the Deaf community. They suggest that their approach could be used to develop more accurate real-time translation tools for ASL or other sign languages, as well as improve communication between Deaf individuals and hearing individuals who do not know sign language.

Overall, this paper provides a valuable contribution to the field of sign language recognition using deep learning algorithms based on ResNet architecture. The authors' approach achieves high accuracy rates with a large dataset, suggesting that it could be applied to real-world scenarios such as developing translation tools or improving communication between Deaf

individuals and hearing individuals who do not know sign language.

D. *American Sign Language Recognition using Deep Learning and Computer Vision [4]*

This paper focuses on the development of a vision-based application that offers sign language translation to text, with the aim of aiding communication between signers and non-signers. The authors highlight the challenges faced by non-sign language speakers when communicating with sign language speakers, particularly those with speech impairments. They note that while sign language is ubiquitous in recent times, there remains a challenge for non-sign language speakers to communicate effectively with signers.

To address this challenge, the authors propose a model that takes video sequences and extracts temporal and spatial features from them. They use Inception, a CNN (Convolutional Neural Network), for recognizing spatial features and a RNN (Recurrent Neural Network) to train on temporal features. The dataset used is the American Sign Language Dataset.

The authors provide an overview of related work in gesture recognition using different methods such as Hidden Markov Models (HMM). They also discuss the dataset used in their work, which consists of 3,000 videos of 24 different signs performed by 10 different signers.

The proposed model is evaluated using various metrics such as accuracy and F1 score. The authors report promising results, with an accuracy of 98% achieved on the test set. They also discuss the cost involved in conducting this research and highlight some of the problems faced by their model.

Finally, the authors propose possible improvements to their model such as incorporating more advanced techniques like attention mechanisms or using larger datasets for training. They conclude that their work offers a promising solution to aid communication between sign language speakers and non-sign language speakers.

In summary, this paper presents a vision-based application for American Sign Language recognition using deep learning and computer vision techniques. The proposed model achieves high accuracy on the test set and offers potential benefits for communication between signers and non-signers. The authors suggest possible improvements to their model and highlight some of the challenges faced during their research.

III. METHODOLOGY

A. *Dataset Preparation*

The ASL dataset is a collection of images that depict hand gestures of the American Sign Language alphabet. The dataset has a total of 87,000 images, which are divided into 29 categories: 26 letters of the English alphabet and 3 additional gestures for space, delete, and nothing. The images are 3 channel RGB images and have a resolution of 200 x 200 pixels. The dataset was obtained from Kaggle.

To train and evaluate our models, we divided the ASL dataset into training, validation, and testing sets using an 80:10:10 ratio. This means that 80% of the dataset was used for training, 10% for validation, and 10% for testing.

Fig 2: Sample of Training Dataset from Kaggle

B. *Preprocessing*

The images in the ASL dataset were pre-processed before being fed into the models. This included resizing the images to 64 x 64, normalizing their pixel values to be in the range [0, 1], and augmenting the training set with random horizontal flips and rotations. The pre-processing step helps to improve the accuracy of the models by ensuring that the images are of a consistent size and are normalized to a standard range of values. The augmentation step helps to increase the diversity of the training set and prevent overfitting.

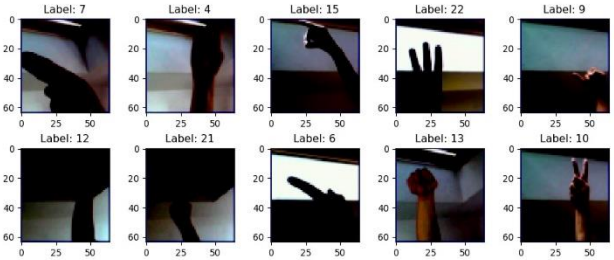

Fig 3: Images after Resizing and Normalization

C. *Model Architecture*

Three models were used in this study: Custom Sequential Model 1, Custom Sequential Model 2, and Resnet18 with transfer learning.

Custom Sequential Model 1 is a convolutional neural network architecture consisting of multiple layers. It begins with a 2D convolutional layer with 3 input channels and 32 output channels, followed by a ReLU activation function. This is then followed by a max pooling layer with kernel size of 2x2 and stride of 2. The next layer is another 2D convolutional layer with 32 input channels and 64 output channels, with a kernel size of

3x3 and stride of 1. This is also followed by a ReLU activation function and a max pooling layer with kernel size of 2x2 and stride of 2. The following layer is another 2D convolutional layer with 64 input channels and 128 output channels, with a kernel size of 3x3 and stride of 1, followed by a ReLU activation function and a max pooling layer with kernel size of 2x2 and stride of 2. A fully connected layer with input size of 2048 and output size of 1024 is then added, followed by a ReLU activation function and a dropout layer with a probability of 0.5. The final fully connected layer has an input size of 1024 and output size of 29, which corresponds to the number of classes in the dataset. The network uses the LogSoftmax activation function in the output layer to produce a probability distribution over the 29 classes.

```
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1          [-1, 32, 64, 64]             896
              ReLU-2          [-1, 32, 64, 64]               0
         MaxPool2d-3          [-1, 32, 32, 32]               0
            Conv2d-4          [-1, 64, 32, 32]          18,496
              ReLU-5          [-1, 64, 32, 32]               0
         MaxPool2d-6          [-1, 64, 16, 16]               0
            Conv2d-7         [-1, 128, 16, 16]          73,856
              ReLU-8         [-1, 128, 16, 16]               0
         MaxPool2d-9           [-1, 128, 8, 8]               0
           Linear-10                 [-1, 1024]       2,098,176
             ReLU-11                 [-1, 1024]               0
          Dropout-12                 [-1, 1024]               0
           Linear-13                   [-1, 29]          29,725
================================================================
Total params: 2,221,149
Trainable params: 2,221,149
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.05
Forward/backward pass size (MB): 3.96
Params size (MB): 8.47
Estimated Total Size (MB): 12.48
----------------------------------------------------------------
```

Fig 4: Model 1 Architecture

Custom Sequential Model 2 is a deeper and more complex version of the original Model 1. It has 5 convolutional layers with an increasing number of output channels (32, 64, 128, 256, and 512), each followed by a batch normalization layer, a ReLU activation function, and a max pooling layer. The max pooling layers have a kernel size of 2 and a stride of 2, which down samples the input by a factor of 2 in both dimensions.

After the convolutional layers, the output is flattened and passed through two fully connected layers. The first fully connected layer has 1024 output features and is followed by a dropout layer with a dropout probability of 0.5. The second fully connected layer has 512 output features and is also followed by a dropout layer with a dropout probability of 0.5. Finally, the output of the second dropout layer is passed through a third fully connected layer with output size equal to the number of output classes.

Overall, Custom Sequential Model 2 has more layers, more channels, and more parameters compared to the original Custom Sequential Model 1. It is also designed to be more robust and less prone to overfitting due to the use of batch normalization and dropout layers.

```
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1          [-1, 32, 64, 64]             896
       BatchNorm2d-2          [-1, 32, 64, 64]              64
              ReLU-3          [-1, 32, 64, 64]               0
         MaxPool2d-4          [-1, 32, 32, 32]               0
            Conv2d-5          [-1, 64, 32, 32]          18,496
       BatchNorm2d-6          [-1, 64, 32, 32]             128
              ReLU-7          [-1, 64, 32, 32]               0
         MaxPool2d-8          [-1, 64, 16, 16]               0
            Conv2d-9         [-1, 128, 16, 16]          73,856
      BatchNorm2d-10         [-1, 128, 16, 16]             256
             ReLU-11         [-1, 128, 16, 16]               0
        MaxPool2d-12           [-1, 128, 8, 8]               0
           Conv2d-13           [-1, 256, 8, 8]         295,168
      BatchNorm2d-14           [-1, 256, 8, 8]             512
             ReLU-15           [-1, 256, 8, 8]               0
        MaxPool2d-16           [-1, 256, 4, 4]               0
           Conv2d-17           [-1, 512, 4, 4]       1,180,160
      BatchNorm2d-18           [-1, 512, 4, 4]           1,024
             ReLU-19           [-1, 512, 4, 4]               0
        MaxPool2d-20           [-1, 512, 2, 2]               0
           Linear-21                 [-1, 1024]       2,098,176
          Dropout-22                 [-1, 1024]               0
           Linear-23                  [-1, 512]         524,800
          Dropout-24                  [-1, 512]               0
           Linear-25                   [-1, 29]          14,877
================================================================
Total params: 4,208,413
Trainable params: 4,208,413
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.05
Forward/backward pass size (MB): 6.32
Params size (MB): 16.05
Estimated Total Size (MB): 22.42
----------------------------------------------------------------
```

Fig 5: Model 2 Architecture

ResNet18 model is a convolutional neural network architecture that consists of 18 layers, including convolutional layers, batch normalization layers, ReLU activation functions, and fully connected layers. It is a variant of the ResNet (Residual Network) architecture that introduces residual connections between the layers, which allows for the training of much deeper networks than was previously possible.

In this implementation, the ResNet18 model uses a pre-trained ResNet18 model as a feature extractor and replaces the original fully connected layer with a new fully connected layer with 29 output classes, corresponding to the number of classes in the dataset. During training, only the weights of the new fully connected layer are updated, while the weights of the pre-trained ResNet18 model are kept fixed.

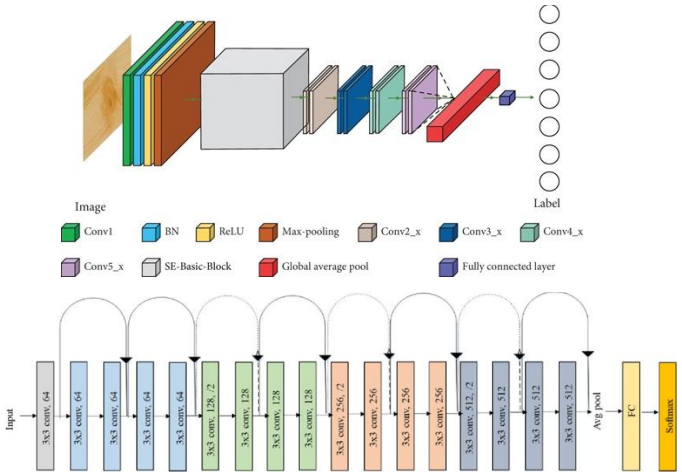

Fig 6: ResNet18 Architecture

D. Training

We used the PyTorch framework to train the models on the ASL dataset. The models were trained for 10 epochs with a batch size of 64 and a learning rate of 0.001. We used the cross-entropy loss function and the Adam optimizer to optimize the

models. The training process was done on an NVIDIA Geforce GTX-1650 GPU.

The SDM optimizer (Stochastic Diagonal Levenberg-Marquardt) is a variant of the classic Levenberg-Marquardt algorithm that is commonly used for non-linear optimization problems. It is used in the Model 1 to update the weights of the neural network during the training process. The SDM optimizer is known for its ability to efficiently converge to the optimal solution while avoiding the problem of overfitting.

The Adam optimizer, on the other hand, is a popular algorithm used for stochastic gradient descent optimization. It is a variant of the classic gradient descent optimization algorithm that uses adaptive learning rates to update the weights of the neural network during the training process. The Adam optimizer is used in Model 2 and ResNet18.

The key difference between the two optimizers is in their approach to updating the weights of the neural network. The SDM optimizer uses a second-order method that computes the curvature of the loss function to update the weights, while the Adam optimizer uses a first-order method that calculates the gradient of the loss function to update the weights.

IV. EXPERIMENTS AND RESULTS

In this study, we have trained three different models on the Kaggle dataset, namely Custom Sequential Model 1, Custom Sequential Model 2, and ResNet18 Model with transfer learning. Each model has undergone 10 epochs of training, and we have evaluated their performance based on various metrics.

Below are the results of the training for each model:

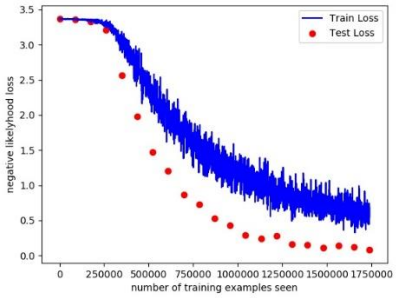

Fig 7: Train and Test Loss for Model 1

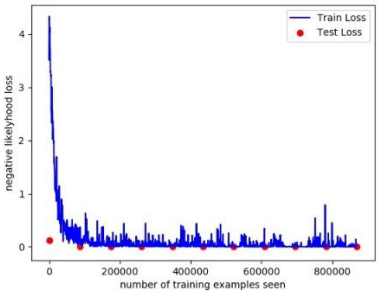

Fig 8: Train and Test Loss for Model 2

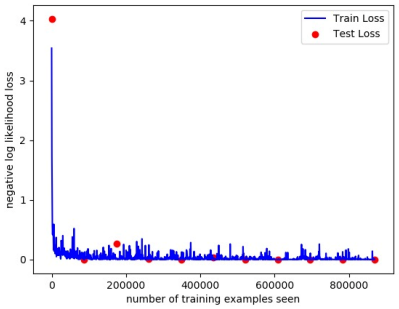

Fig 9: Train and Test Loss for ResNet18

To evaluate the performance of each model, we computed several metrics such as precision, recall, and accuracy. The results showed that the ResNet18 model outperformed the other two models in all aspects, with a precision and recall of 1.0 and an accuracy of 100%. The Custom Sequential Model 2 also performed reasonably well, with an accuracy of 82.8%. However, the Custom Sequential Model 1 showed poor performance, with an accuracy of only 10.3%.

After analysing the results of the three models on the ASL dataset, ResNet18 outperformed the other two models significantly. The Precision and Recall values of ResNet18 were both 1.0, which indicates that the model made accurate predictions for all classes. This shows that the ResNet18 model is better at correctly identifying hand gestures in the ASL dataset than the other two models.

On the other hand, the Model 1 had a low accuracy of 0.103, which indicates that the model had a hard time predicting correctly for most classes. Furthermore, the Recall and Precision values were 0, indicating a lack of model confidence in predicting for most classes.

The Model 2 also had relatively good Precision and Recall values, but they were not as high as those of ResNet18. The accuracy of the model was 0.827, which is lower than that of ResNet18, indicating that it still had some difficulties in correctly classifying some of the hand gestures.

In conclusion, the ResNet18 model performed better than the other two models because it was able to accurately identify and classify all hand gestures in the ASL dataset. This indicates that ResNet18 is a suitable model for ASL recognition tasks, as it can achieve high accuracy and precision, making it a viable option for real-world applications.

Fig 10: Results for Custom test data using Model 2

Fig 11: Results for Custom test data using ResNet18

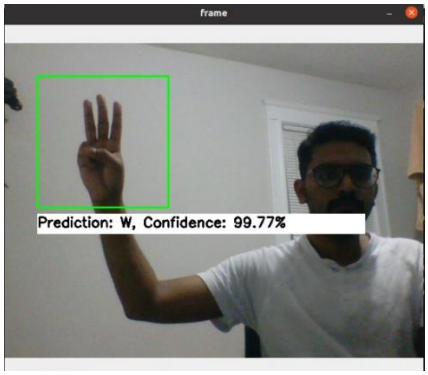

Fig 12: Real-time result for Aplhabet W using ResNet18

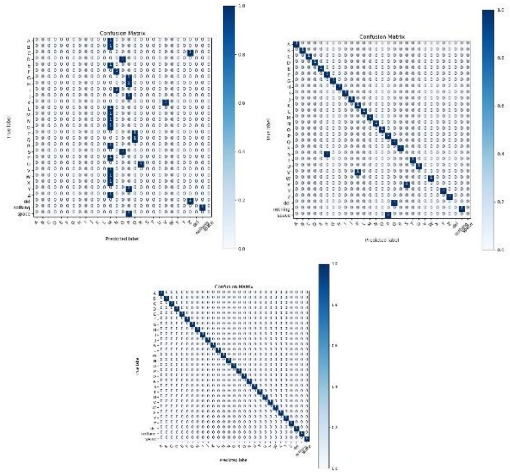

Fig 12: Confusion Matrix for Model 1, 2 and ResNet18 (clockwise)

V. DISCUSSION AND SUMMARY

The Resnet18 model works better than the Custom Sequential Model 1 and 2 on the ASL Kaggle dataset training for several reasons:

- Deeper Architecture: Resnet18 has a deeper architecture. Deeper neural networks tend to perform better on complex tasks such as image classification, as they can learn more complex features and patterns in the data.
- Pre-trained weights: Resnet18 comes with pre-trained weights that were trained on the large ImageNet dataset. This means that the Resnet18 model has already learned a lot of useful features and patterns in images that are likely to be relevant to the ASL Kaggle dataset, and it can leverage these pre-trained weights to improve its performance on this task.
- Skip Connections: Resnet18 uses skip connections to address the vanishing gradient problem, which can be a common issue in deeper neural networks. These skip connections allow the gradients to flow more easily through the network during training, which can help the model converge faster and perform better.

REFERENCES

[1] Rathi, Pulkit and Kuwar Gupta, Raj and Agarwal, Soumya and Shukla, Anupam, Sign Language Recognition Using ResNet50 Deep Neural Network Architecture (February 27, 2020). 5th International Conference on Next Generation Computing Technologies (NGCT-2019).

[2] M. Al-Qurishi, T. Khalid and R. Souissi, "Deep Learning for Sign Language Recognition: Current Techniques, Benchmarks, and Open Issues," in IEEE Access, vol. 9, pp. 126917-126951, 2021, doi: 10.1109/ACCESS.2021.3110912.

[3] Abu-Jamie, Tanseem N. & Abu-Naser, Samy S. (2022). Classification of Sign-Language Using Deep Learning by ResNet. International Journal of Academic Information Systems Research (IJAISR) 6 (8):25-34.

[4] K. Bantupalli and Y. Xie, "American Sign Language Recognition using Deep Learning and Computer Vision" 2018 IEEE International Conference on Big Data (Big Data), Seattle, WA, USA, 2018, pp. 4896-4899, doi: 10.1109/BigData.2018.8622141.