Individual Final Report
Daqian Dang
DATS 6303: Deep Learning
Instructor: Dr Amir Jafari
April 23, 2023

Deep Learning-Based ASL Fingerspelling Recognition System for Enhanced Communication and Accessibility

I. An overview of the project

The goal of this project is to design and develop a deep learning-based American Sign Language (ASL) fingerspelling recognition system, with the ultimate aim of bridging the communication gap between the hearing and deaf or hard of hearing communities. By concentrating on ASL fingerspelling, which is comprised of 24 unique hand configurations corresponding to the alphabet's letters, the project strives to establish a robust foundation for more sophisticated applications in the future, such as real-time ASL sign reading, transcription into text, or conversion to speech.

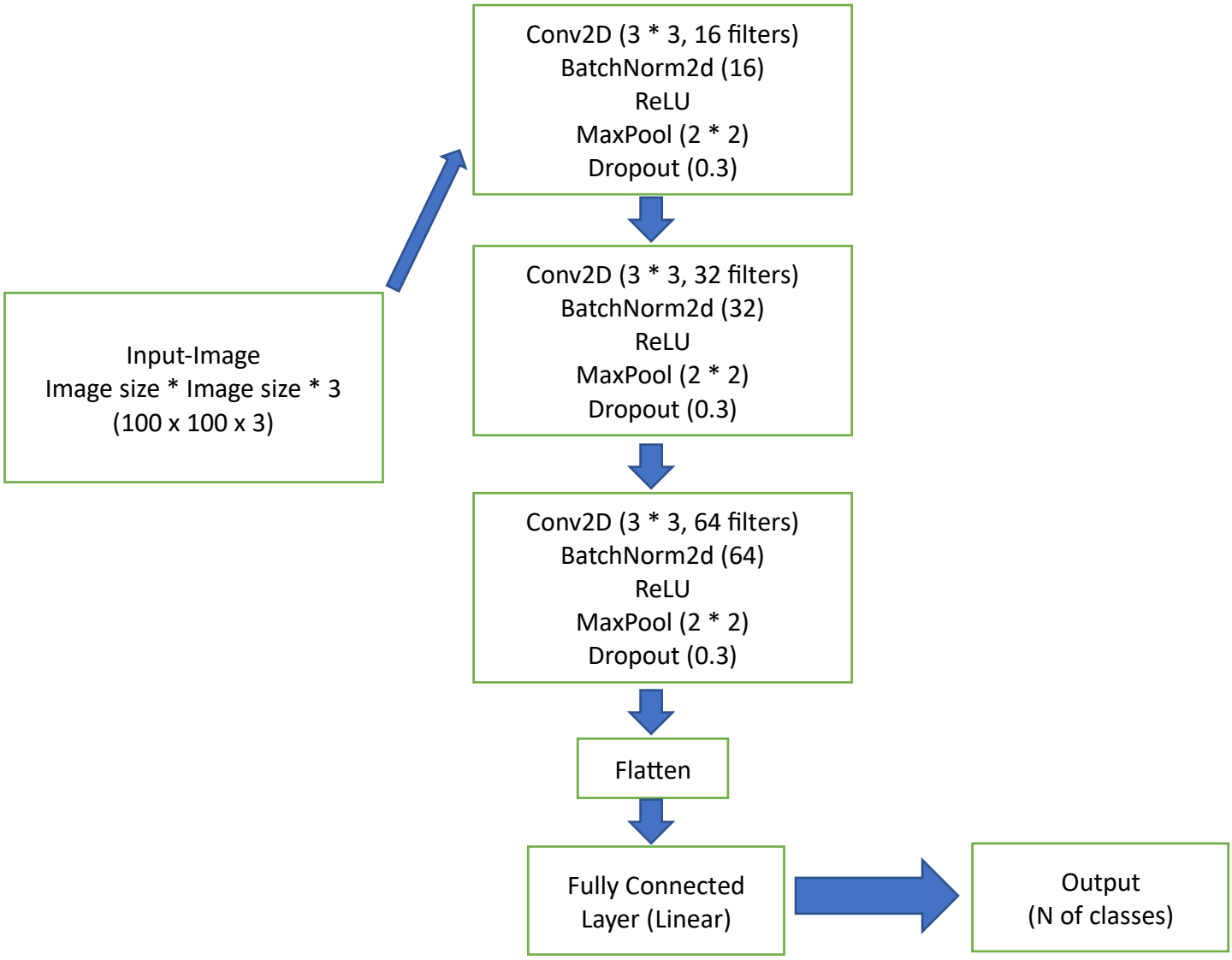
To facilitate the project, Kaggle's "ASL Fingerspelling Images (RGB & Depth)" dataset was utilized, encompassing 131,662 pre-cropped color images featuring signing hands from five distinct individuals. While only the RGB images were used for the project, the dataset spans 24 ASL classes, with the exception of letters J and Z, which necessitate motion for accurate representation. The class distribution is fairly balanced, with a minimum of 5235 images and a maximum of 6220 images per class. This diverse dataset provides a solid basis for training our deep learning model to recognize a wider range of ASL fingerspelling variations, ultimately enhancing the model's performance.

An outline of the shared work:

1. Rajkumar and I collaborated on preprocessing the code, which included data augmentation.
2. Rajkumar was responsible for setting up the baseline model.
3. Andrew reviewed the initial code and made revisions to improve it.
4. Once the preprocessing and baseline model were cleaned up, I experimented with and modified the CNN model architecture from the baseline.
5. I added some code such as early stopping, learning rate scheduling, and saving the model to find the best results. These modifications were slightly different from Andrew's work.
6. Each team member tried to optimize the code on their own to find the best results.

II. Describe the portion of my work on the project.

Model Architecture: My model for this project is a straightforward convolutional neural network (CNN) with three convolutional layers and a final fully connected layer for output. Each convolutional layer includes padding, batch normalization, ReLU activation, max pooling, and 30% dropout. The second and third convolutional layers use two groups for the convolution operation. After the third convolutional layer, the output is flattened and passed through a fully connected layer to produce the final output for the distinct ASL classes. The Softmax activation is not included in the model definition since the loss function integrates it during training. Below is the block diagram for the model:



Code Modification on "fit" section: The code defines a "fit" method for training and evaluating a deep learning model. It sets up the optimizer, loss function, and learning rate scheduler, and runs the training loop for a specified number of epochs. The method calculates the training and validation losses, accuracies, and F1 scores, and adapts the

learning rate based on validation loss. It also saves the best model, implements early stopping, and evaluates the model's performance on a test dataset. Some of added codes that I have put in the “fit” section are …

1. Initializes the best validation F1 score, best model, and best results.
   best_val_f1 = 0
   best_model = None
   best_results = None

2. Sets up early stopping parameters, including patience and a counter.
   patience = 5 - Number of epochs to wait before stopping if there's no improvement
   min_val_loss = np.inf - Initialize minimum validation loss as infinity
   counter = 0 - Initialize early stopping counter

3. Calculates the validation F1 score for each epoch.
   val_f1 = f1_score(vl_labels.cpu().numpy(), val_pred_labels.cpu().numpy(), average='macro')

4. Prints epoch statistics, including training and validation losses and accuracies.
   print("Epoch {} | Train Loss {:.5f}, Train Acc {:.5f} - Val Loss {:.5f}, Val Acc {:.5f}".format(epoch + 1, loss_train / len(self.tr_loader), acc(train_pred_labels, tr_labels), loss_val / len(self.vl_loader), acc(val_pred_labels, vl_labels)))

5. Updates the learning rate scheduler based on validation loss.
   scheduler.step(loss_val)

6. Saves the best model and results if the validation F1 score improves.
   ```
   if val_f1 > best_val_f1:
      best_val_f1 = val_f1
      best_model = copy.deepcopy(model.state_dict())
      best_results = {
         'epoch': epoch + 1,
         'train_loss': '{:.5f}'.format(loss_train / len(self.tr_loader)),
         'train_acc': '{:.5f}'.format(acc(train_pred_labels, tr_labels)),
         'val_loss': '{:.5f}'.format(loss_val / len(self.vl_loader)),
         'val_acc': '{:.5f}'.format(acc(val_pred_labels, vl_labels)),
         'val_f1': '{:.5f}'.format(val_f1),
      }
   ```

7. Implements early stopping if validation loss doesn't improve for a specified number of epochs (determined by the patience variable).
   ```
   # Early stopping
   if loss_val < min_val_loss:
      min_val_loss = loss_val
   ```

```
        counter = 0
    else:
        counter += 1
        print(f'Early stopping counter:{counter} out of {patience}')
        if counter >= patience:
            print("Early stopping")
            break
```

8. Saves the best model to a file.
   ```
   torch.save(best_model, "best_model.pt")
   ```

9. Prints the best results (epoch, train loss, train accuracy, validation loss, validation accuracy, and validation F1 score).
   ```
   print('Best results:')
   for key, value in best_results.items():
       print(f"{key}: {value}")
   ```

10. Calculates and prints the F1 score for the test dataset.
    ```
    test_f1 = f1_score(ts_labels, ts_pred, average='macro')
    print(f"Test F1 score: {test_f1: .5f}")
    ```

Overall, the fit method provides a complete solution for training and evaluating a deep learning model.

Optimization Experiments for my model: The experiments conducted to develop the ASL fingerspelling recognition system resulted in the model in terms of accuracy and F1 score. By employing a combination of hyperparameters, such as learning rate, batch size, kernel size, number of layers, dropout, and batch normalization, the model effectively learned intricate patterns and generalized well to new data. Also, I experimented with L2 regularization which added to optimization ADAM for preventing overfitting. The table below presents the hyperparameters used for my model:

Hyperparameter for Daqian's model	
Learning rate	0.001
minibatch size	512
Kernel size	3 x 3
N of layers	3
Dropout implemented in each layer	0.3

Batch normalization implemented in each layer	16, 32, 64
Epoch Number	30
Optimizer	ADAM
L2 regularization (weight decay)	0.001

III. Results for Daqian's model

The model achieved a train accuracy of 0.950 and an even higher validation accuracy of 0.966, showing its potential for recognizing ASL fingerspelling on unseen data. The test F1 score of 0.965 further supports the model's effectiveness in this task. See result table below.

Results for Daqian's model	
Best N of Epoch	29
Train loss	0.137
Train accuracy	0.950
Validation loss	0.145
Validation accuracy	0.966
F1 score for test set	0.965

During my experimentation with ADAM optimization and L2 regularization, I closely monitored the performance of the model to identify any improvements. However, despite my efforts, I did not observe any significant enhancements in the model's accuracy or performance. I analyzed the results thoroughly and explored various strategies to fine-tune the model and overcome this issue. Nonetheless, the outcomes remained the same, and there was no observable improvement.

In addition, when experimenting with a pretrained model, like EfficientNet, it was found that the computation time was much slower compared to the simpler model. This slower performance is due to the complexity of the EfficientNet architecture, which might not be necessary for the ASL fingerspelling recognition task. In fact, the results show that the simpler model was more effective for this specific application.

This finding highlights the importance of selecting the right model architecture for the problem. While complex models like EfficientNet are effective in many computer vision tasks, they may not always be the best choice, particularly for less complex tasks or when computational resources are limited. In such cases, simpler models with well-tuned hyperparameters can deliver better performance and faster training times, as demonstrated in this project.

IV. Summary and conclusions

The results obtained from this group project demonstrate the effectiveness of the proposed deep learning model in recognizing ASL fingerspelling. The model achieved a train accuracy of 0.950, a validation accuracy of 0.966, and a test F1 score of 0.965. These performance metrics indicate that the selected architecture and hyperparameters are suitable for the ASL fingerspelling recognition task.

Throughout the project, I learned the importance of choosing an appropriate model architecture and optimizing hyperparameters for the specific problem at hand. While complex models like EfficientNet might perform well in various computer vision tasks, they may not always be the optimal choice, especially for less complicated tasks or when computational resources are limited. In this case, the simpler model with well-tuned hyperparameters delivered better performance and faster training times.

For future improvements and research directions, the following suggestions can be considered:

- Expanding the dataset: Including more diverse samples of fingerspelling from a larger number of individuals and different lighting conditions can help improve the model's generalization capability.
- Incorporating motion: Since ASL includes dynamic hand gestures, such as letters J and Z, incorporating motion information could help recognize a more comprehensive set of ASL signs.
- Real-time recognition: Developing a real-time ASL fingerspelling recognition system can facilitate communication between the hearing and deaf or hard of hearing communities more effectively.
- Multi-modal input: Combining RGB images with depth data or other sensory information can potentially improve the recognition accuracy and robustness of the system.

In conclusion, this project successfully developed a deep learning-based ASL fingerspelling recognition system that performs well on unseen data. The insights gained from this project can be utilized to further enhance the system and explore more advanced applications in the future, such as real-time ASL sign reading, transcription into text, or conversion to speech.

V. Calculation of the percentage of my code in the project

(71-30)/(71+34) *100 = 39%

VI. References:

Geislinger, M. (2021). ASL Fingerspelling Images (RGB & Depth) [Data set]. Kaggle. https://www.kaggle.com/datasets/mrgeislinger/asl-rgb-depth-fingerspelling-spelling-it-out

Pugeault, N., & Bowden, R. (2011). Spelling It Out: Real-Time ASL Fingerspelling Recognition. In Proceedings of the 1st IEEE Workshop on Consumer Depth Cameras for Computer Vision, in conjunction with ICCV 2011 (pp. 1114-1119).

Pugeault, N. (n.d.). ASL Fingerspelling Dataset. https://empslocal.ex.ac.uk/people/staff/np331/index.php?section=FingerSpellingDataset

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 770-778).

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., ... & Chintala, S. (2019). PyTorch: An imperative style, high-performance deep learning library. In Advances in Neural Information Processing Systems (pp. 8026-8037).

Tan, M., & Le, Q. V. (2019). EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. In Proceedings of the 36th International Conference on Machine Learning (ICML 2019) (pp. 6105-6114).