**SU\_GROUP12 - PROJECT REPORT**

1. **Preprocessing Steps**

Since the task is learning the representation of symbols in American Sign Language through images, the first step is converting the raw image data into its proper format for Machine Learning. Originally, the raw data includes 200x200 pixels images, which are resized as 100x100 pixels. For the images, there are a total of 29 labels: 26 letters of the English alphabet, “space”, “del” and “nothing”; these categorical labels are converted to numeric labels for further steps that involve deep learning and neural networks. Then, the structure of the datasets is checked by observing the shapes, maximum, minimum values in the features and also the unique values in the labeling. The maximum value in features is 255, and the minimum is 0 as expected to be present in RBG images. Therefore, it is understood that there are no anomalies in the datasets, and further processing could be performed. All the RGB values in the training and test sets are converted to float and divided by 255 for scaling and normalization purposes. One-hot encoding for the training labels is performed for all 29 classes. In order to ensure the reproducibility as much as possible, random seeds for the Python environment, NumPy, and TensorFlow are set and the validation data split has been executed using a fixed random state. 20% of the training set is allocated for validation purposes.

1. **Experiments and Hyperparameter-tuning**

We have implemented several architectures for Convolutional Neural Networks and benefitted from different Transfer Learning approaches such as VGG16, ResNet50, and InceptionV3. In all of our models, we used the Checkpoint callback in order to retrieve the model constructed in the best epoch during the runtime, in terms of validation accuracy. Validation accuracy is the preferred performance metric within the context of this project since the class distributions are perfectly balanced.

**2.1.CNN**

In the first model, we add 4 convolutional and pooling layer pairs, with kernel size 7x7, pooling size 2x2, and the number of filters is increasing from 16 to 128, in the order of 2. After flattening, we add 3 dense layers of neuron size 128, 64, and 29. We keep the batch size as 256 and the number of epochs as 100. The maximum validation accuracy that we could obtain is 96.655%, therefore we infer this as an underfitting case and increase the number of convolutional layers in the further models.

In the second model, we add 2 more convolutional and pooling layer pairs, but starting from the number of filters 8 and doubling in each layer, up to 256. Within this new architecture, we tried to explore the hyperparameter space as much as possible. The second model is the best model we have experimented with. Until the last layer, only the non-linear activation function ReLu is used while in the last dense layer of 29 neurons we have used softmax activation function since we have multiple classes. This makes a total number of 3,756,333 parameters that are all trainable. We used a batch size of 128 and 100 epochs have been carried out. We obtain an accuracy of 99.879%.

Then, we compiled the very same model with a lower learning rate of 0.00001, where the iterative learning in the epochs is highly observable since the increase in both training and validation accuracy has been decreased a lot. We didn’t change the number of epochs and the resulting maximum accuracy is 98.362%, which is indeed lower than the previous even if we decreased the learning rate. This may be the result of not increasing the number of epochs, but more epochs might result in overflow in RAM since even with 100 iterations, it was really close to the capacity.

In the next model, we kept the learning rate as default and increased the batch size to 256. The same architecture with the same hyperparameters is trained. The resulting validation accuracy is 99.724%, which we observe an increase due to the batch size.

Then another experiment has been conducted with the same architecture as the batch size is kept as it is and the learning rate has been set to 0.0001. The validation accuracy has been improved to 99.672% since model 3 but hasn’t outperformed the last and very first model described above. The recent increase may be due to the increase in the batch size. Due to the memory issues, since we can’t increase the number of epochs as the learning rate decreases, we didn’t obtain an increase in the validation accuracy when we decrease the learning rate in Convolutional Neural Network models.

**2.2. Transfer Learning**

**2.2.1. ResNet50**

The first model that we tried was the ResNet50 model. After importing and generating the model, we made the last 5 layers trainable then added Flatten layer and Dense layers with neuron count; 1024, 512, 256, 29 in this order. After that, we compiled the model with optimizer Adam and started training the model with batch size 256 and epochs 10. With these hyperparameters, we got the validation accuracy of 0.55207.

Then we decided to change the hyperparameters with the hope of increasing the accuracy rating. We changed the batch size to 128 from 256 and trained a newly created model with these hyperparameters. From this experiment, we got a validation accuracy of 0.6232. With this we decided that 128 batch size was better.

Then we generated the model again and added the flatten layer and Dense layers with neuron count; 1024, 1024, 1024, 29 in this order. After that, we compiled the model with optimizer Adam and trained the model with batch size = 128 and epochs = 100. With this, we got the validation accuracy of 0.9457. Which was the best of them all. Since 94.57% accuracy is not sufficient, we decided to explore other Transfer Learning methods such as InceptionV3 and VGG16.

**2.2.2. InceptionV3**

Two InceptionV3 models were trained for our classification task. The first model used the weights of InceptionV3 without making any layers of it trainable, and it had 3 Dense layers with 1024 neurons. For the second model, the last 5 layers of InceptionV3 were made to be trainable, and it had 3 Dense layers with 128, 256, and 512 neurons. For both models, non-linear activation function ReLU, batch size of 128, Adam as an optimizer, and default learning rate of 0.01 were chosen as hyper-parameters. The first model was run for 100 iterations, while the second model was run for 50 iterations. The checkpoint accuracies of both models are as follows: Model 1: checkpoint accuracy: 0.9996 - val\_accuracy: 0.9798; Model 2: checkpoint accuracy: 0.9946 - val\_accuracy: 0.9610. Considering the fact that both models seemed to overfit when we observed the difference between the training set and validation set accuracies (or validation set accuracy did not seem to improve even though training set accuracy reached 100%) even with the best weights saved as a checkpoint, and also the fact that VGG16 model happened to be a better performer for our task within our time constraints; we decided to move our attention to trying out and tuning as many VGG16 models as possible.

**2.2.3. VGG16**

The last model we tried using Transfer Learning was VGG16. After importing and generating a model, we created our own input format using the generated model. For the hyperparameter tuning, we added the fully connected layers consisting of a Flatten layer and 5 dense layers with neurons; 1024, 512, 256, 128, 29 and with activation functions; ReLu, ReLu, ReLu, ReLU, Softmax in this order. First we tuned to find out how many pretrained layers to use for feature extraction. First, freezed all the layers and then unfreeze the last 5, 10, 20 and 30 layers and see the results with this unfreezing. For these models we compiled our models with learning rate 0.0001 and trained our models using batch size 256, 10 epochs. When we freezed all layers, we got validation accuracy of 0.9947 = 99.47% and validation loss 0.0362, when we unfreezed last 5 layers we got validation accuracy 0.9931 with validation loss 0.0392, when we unfreezed last 10 layers, we got validation accuracy 0.9933 and validation loss 0.0404, when we unfreezed last 20 layers we got validation accuracy 0.9940 and validation loss 0.393 and when we unfreezed last 30 layers we got validation accuracy 0.9940 and validation loss 0.0391. So when we unfreeze the last layers it did not improve the accuracy and loss much. From this point we decided not to unfreeze the layers. After that we tuned the batch size and instead of training our models with batch size 256 we trained with batch size 128 with freezing all the trainable layers and compiled with learning rate 0.0001. The validation accuracy of this model was 0.9931 and validation loss was 0.0290. Changing batch size did not give much improvement so we trained our models with batch size 256. Lastly we tuned our models for learning rate. We first compiled our model with learning rate 0.001 with the same architecture as before and trained with batch size 128, 10 epochs. It gave a validation accuracy of 0.9752 and validation loss of 0.0702. So when we changed the learning rate from 0.0001 to 0.001, the validation accuracy and loss did not improve. Then we tried to reduce the learning rate to 0.00001, it converged slowly compared to other models and after 10 epochs, it gave validation accuracy of 0.9452 and validation loss of 0.3595 which is worse than other learning rates.

After that we created 4 different models with increased epochs. First the first one we added the fully connected layers consisting of a Flatten layer and 4 dense layers with neurons; 1024, 512, 256, 29 and with activation functions; ReLu, ReLu, ReLu, Softmax in this order. Then we created our own model, compiled it with a learning rate of 1e^-5, because with larger epoch size smaller learning rate might have given good results compared to the 10 epoch models we have trained so far. Then trained it with batch size = 256, epoch = 150 while using ModelCheckpoint as a callback to save the best result. With this model, our best validation accuracy was 0.9993.

For the second model, we conducted another experiment in the hope of improving the result we got. This time, while we were adding the fully connected layers we decided to add one more dense layer between dense layers with neurons 256 and 29 with a neuron count of 128 and with the activation function ReLu. Then we compiled it with a learning rate of 0.0001 and trained it with batch size = 256 and epoch = 200 without early stopping and with this model, we got the validation accuracy of 0.9995.

For the third model, we conducted another experiment with different dense layers, this time we added 6 fully connected layer to the pretrained model with neurons; 512, 256, 256, 256, 128, 29 and with activation functions; ReLu, ReLu, ReLu, ReLu, ReLu, Softmax respectively. For this model we freezed all the trainable layers and compiled the model with learning rate 0.0001 and trained the model with batch size 256 and epoch 150 while using ModelCheckpoint as a callback to save the best result. With this model, our best validation accuracy was 0.99931. This model gave an accuracy score of 0.99908 on Kaggle.

For the fourth model, we took the exact same model with unfreezing last 5 layers and compiled the model with learning rate 0.0001 and trained with batch size 256 and epoch 100 while using ModelCheckpoint as a callback to save the best result. With this model, our best validation accuracy was 0.99931, which is not an improvement nor reduction in the validation accuracy.

The second model we trained with validation accuracy 0.9995 gave the best results in the validation data, so we decided to train this model using all the training data and see the results in the test data in Kaggle. We compiled the same model with learning rate 0.0001 and trained with batch size 256 and epoch 100. This model gave a test accuracy of 0.99908 in the Kaggle.

1. **Best Model**

The best model is the second model explained in the CNN section, which can also be found in Model 2 of the CNN section in the Google Colab, as well as the “Predictions with the best model” section. The model is trained with the whole training data for prediction stage, and the checkpoints are added with respect to training loss. The best model’s test set 1 accuracy (or should we say the accuracy that is seen on Kaggle leaderboards) is 0.99938.