**CS412 Spring 2021 Project**

**GROUP 1**

**Team Members**

**Aleyna Buse Sümer – 25429**

**Tolga Baydar – 25128**

**Doğancan Çelik – 25470**

**Zeynep Tüfekçi - 25109**

Preprocessing:

Instead of rewriting everything by hand, we chose to make use of the preprocessing tools available in Keras. We also used the ImageDataGenerator class to read the images and split the sets. We rescaled (normalized) the images by dividing by 255 so that the pixel values are floats between [0,1], and resized the images to be 224x224 using a nearest interpolation method since we used a pertained architecture. We also used 20% of the training set as the validation split, and shuffled the images (23200 for training, 5800 for validation). We tried to use different data augmentation techniques such as introducing rotations and brightness tweaks but we did not improve the performance using them, so in our final model, we chose not to. We used a batch size of 256 images for the ImageDataGenerator.

In all of the following experiments, we use categorical cross entropy as our loss function since we are dealing with a multi-class classification problem. For the training hyperparameters, we used the Adam optimizer, which defaults to an initial LR of 1e-4. We reduce the LR by a factor of 0.05 when the learning stagnates, until we hit a minimum LR of 1e-7. The patience for early stopping is usually one more than the patience for reduce LR due to the following logic: if even after reducing the learning rate, there was no improvement in learning, then the model is done learning. We also read the images as float32.

Models

1. CNN Model 1: This was the first model that we tried. For this model, we downsized the images to be 64x64 pixels. It consisted of a series of three convolutional blocks (each consisting of a convolutional layer, an activation layer, and a max pool layer), followed by flattening layer and three dense layers, where all the layers used the ReLU activation function and Glorot Uniform initialization. The last layer used a softmax activation and had 29 outputs units, since we were working on a classification task of 29 classes. In training, we used a batch size of 256 images and enough steps so that the model would go through the entire training dataset in each epoch. We ran the learning model with the reduce learning rate and the early stop callbacks provided in Keras. We used a patience of 4 for early stopping, and a patience of 3 for reduce LR. We let the model train for at most 100 epochs. The model finished training at epoch 61 due to early stop, and the ending metrics were: loss: 0.0033, accuracy: 1.0000, val\_loss: 0.0031, val\_accuracy: 1.0000. We submitted the prediction results for this model and obtained an accuracy of 89.73%.
2. CNN Model 2: This model was largely inspired by the first model we tried. We downsized the images to be 64x64 pixels using the nearest interpolation method and used 30% of the data in our validation split (we were trying to get more realistic validation metrics since 30% of 29000 is 8700, roughly the same size of the test set). In this model, we used the same convolutional block structure in CNN Model 1. However, instead of three, we used 4 convolutional blocks with more filters. We used the same training scheme described in CNN Model 1 with the same callbacks. The model finished training on epoch 100 with the following metrics: loss: 3.5002e-04, accuracy: 1, val\_loss: 3.5814e-04, val\_accuracy: 1. We submitted the prediction results for this model and obtained a higher accuracy of 94.92%.
3. CNN Model 3 (regularized): After the results obtained in CNN Model 2, we started to suspect that the model needed t be more complex to predict better. We also checked the pictures and wondered if we have been losing information (such as the lines between the fingers) while resizing. Hence, we changed our preprocessing scheme slightly here. Here, we augmented the data by randomly changing the images’ brightnesses to 30-100% of their original brightnesses (to try to avoid overfitting) and downsized the images to be 80x80 pixels. We used the same convolutional structure in CNN Model 2, and added another dense layer. For all the layers, we also added regularizers for the kernel, bias and activity (l1-l2, l2, l2 respectively). We also tried to use the He Uniform initializer for the weights. This model had roughly 1M trainable (and total) parameters, which is about 4x as much parameters as the previous models. The model finished training on epoch 30 with the following metrics (due to early stopping): loss: 0.2106, accuracy: 0.9993, val\_loss: 0.2102, val\_accuracy: 0.9997. Note that we started to monitor the validation accuracy with early stopping and counted a change more than 1e-3 to be an “improvement”. We submitted the prediction results and were shocked to see that the performance actually went down to 88.38%. We decided not to pursue these CNN models any more and started brainstorming for another approach.
4. Pretrained Model 1: After brainstorming and trying different versions of the previous models (with different filter sizes, layers), we decided to attempt to use Transfer Learning as mentioned in the assignment PDF. We researched about which infrastructure to use and decided to use MobileNetV2 (MN). We used the MobileNetV2 network, pertained on the ImageNet dataset. We resized the images to be 224x224 pixels (so it works with MN). We also normalized the pixels to be in the [0,1] range (Note: we also tried rescaling the pixels to [-1,1], but we attained slightly lower accuracy). We did not use the top layers of MN; instead, we followed MN with a global average pooling layer, and two dense layers followed by a sense (softmax) layer. We also used dropout layers (to turn off 20% of the units) between the dense layers to avoid overfitting. Note that this network had 2.4M parameters, 174K of which were trainable (from the dense layers), and the rest were inside MN. We chose to not tweak the weights of MN, and focus on interpreting the feature maps created by it using the dense classification layers. In the first epoch, the loss was 3.41, which is almost -ln(1/29) (approx. 3.36), which was nice to see. Training finished in 36 epochs, and the final metrics were:loss: 0.2014, accuracy: 0.9678, val\_loss: 0.1142, val\_accuracy: 0.9986. Note that generally, during this run, the validation loss and validation accuracy were *lower* than the training loss and training accuracy. This behavior is expected since we used dropouts - it creates a more robust model. We submitted the prediction using this model and attained 97.45% prediction accuracy, which is our highest. When we normalized the pixels to [-1,1], we attained 97.34% accuracy. We chose this model, since it performed the best.
5. Pretrained Model 2 (Augmentation): By manual checking, we found that some of the darker images are harder to classifcy; hence, we resorted to trying to see if using random brightnesses would make the model more robust. We also added in some rotations (up to 10 degrees), and random brightness effects (50% - 100%). This made the learning task more difficult for the model, and we had hoped that it would generalize better to new datasets. We used the same exact structure as Pretrained Model 1 and trained the model. The model converged after 55 epochs (20 more than the previous model) with the following statistics: loss: 0.3252, accuracy: 0.9310, val\_loss: 0.1734, val\_accuracy: 0.9845. We ran the model on the test set and submitted the result, and we got 95.77%.
6. Pretrained Model 2 (Augmentation, Deeper): We realize that the task has become harder after adding the different random edits to the photos (slight rotations and dimming). Hence, in an attempt to deal with this, we decided to give our model more freedom (and hence more parameters). In this model, instead of two dense and one classification layer, we used three dense layers and one classification layers, with the exact same setting as previously done. The model had 190K trainable parameters (distributed over more dense layers for nonlinearity). Our model converged after 60 epochs and the final metrics were: loss: 0.2446, accuracy: 0.9659, val\_loss: 0.1423, val\_accuracy: 1. We submitted the result and the accuracy on the test set was 94.8%. We realized that this model began to overfit and/or that the brightness and rotation augmentations don’t help that much. We decided to drop the more complex models and investigate the ones with less dense layers.
7. Pretrained Model 3 (Shallower 1): We do not any augmentation or randomization techniques here, and we normalize the pixels to be in the [-1,1] range. We use the same exact architecture as previous: MN as base, and dense layers afterwards. We used one dense layer here with 128 units followed by a softmax classification layer. The model was shallower, and had about 160K trainable parameters. We trained the model with the same settings and callbacks, and it converged after 14 epochs with the following metrics: 0.3773, accuracy: 0.9168, val\_loss: 0.1697, val\_accuracy: 0.9916. We predicted the test set, and obtained an accuracy of 95.47%. This was an improvement from Pretrained Model 2, hence we decided to investigate an even shallower model.
8. Pretrained Model 4 (Shallower 2): This is the same exact setting as in Pretrained Model 3. The only difference is that after the global average pooling layer (after MN), we had no dense layers, and only the softmax classification layer. In this model, we had about 37K parameters. This model converged fast, after only 15 epochs. The final metrics were: loss: 0.1323, accuracy: 0.9885, val\_loss: 0.1216, val\_accuracy: 0.9936. We ran the model on the test set and obtained an accuracy of 96%. In the end, Pretrained Model 1 was still the best; hence, it is the model that we choose in the end.

Note(s)

1. The required submission format is not accepted on Kaggle. That is why our first submission gave an error. The format requested is “Id,Predictions”; however, Kaggle only accepts “Id,Prediction”. We assume that the test2 submission will be evaluated with the same script on Kaggle, hence we submit using the header names “Id,Prediction” too.

We only provide the notebook for Pretrained Model 1. The other notebooks are all saved (as different versions of the same script on Kaggle) and can be given upon request. We only submit one notebook to decrease the clutter.