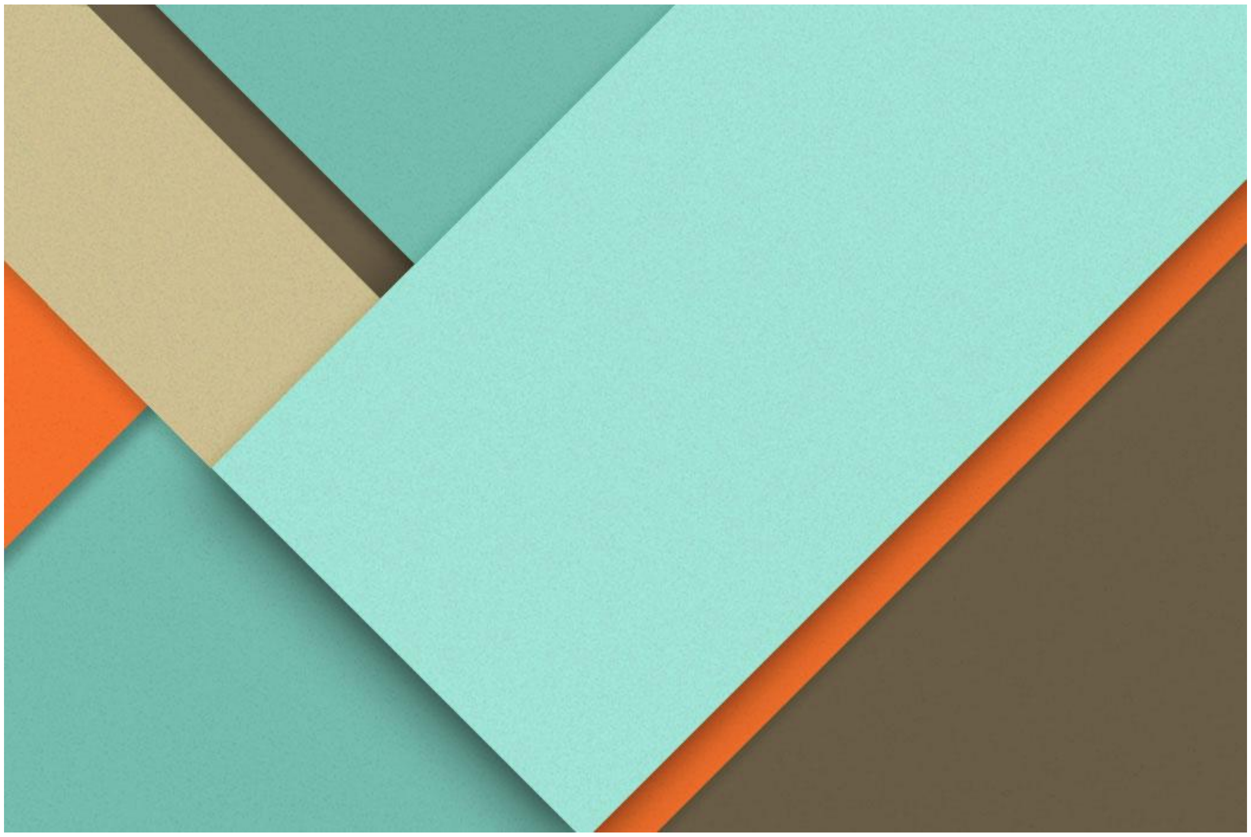


Arabic Hand Gesture Recognition

12.12.2020

—

Mohammed Zakarian

Introduction

The project mainly belongs to deep learning(subdomain of machine learning) involving neural networks to develop a predictive model trained on the collected dataset. The domain of this project is related to sign language used by the deaf and the mute people. The system will use the visual hand dataset based on an Arabic Sign Language and interpret this visual data in textual information. While running the system, each meaningful text is translated using its corresponding hand gesture as an input. So, before giving an insight of the dataset and the deep learning methods used, it is useful to talk about the problem statement described in the succeeding section.

Problem statement

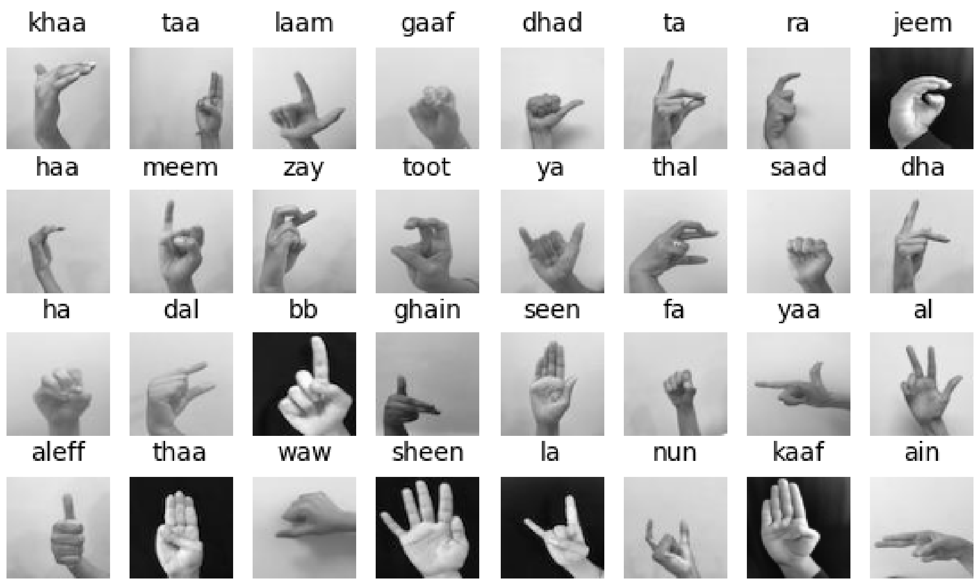
Sign language is very important for the deaf and the mute people to communicate both with the normal people and with themselves. We as normal people tend to ignore the importance of sign language which is the mere source of communication for the deaf and the mute communities. These people are facing major downfalls in their lives because of these disabilities or impairments leading to their unemployment, severe depression, and several other symptoms. One of the services they are using for communication or for us to talk to them are the sign language interpreters. But, hiring these interpreters is very costly and therefore, a cheap solution is required for resolving this big problem of these massive unfortunate people. A thorough analysis and survey to find a way to make these disabled communicable with themselves and the normal people have led to the breakthrough of the Sign Language Recognition System. The system targets to recognize the sign language and translate it into text for some meaningful communication.

Objective

Our aim is to develop a sign language recognition system capable enough to translate the most commonly expressed hand gestures used by deaf or a dumb person into textual data. To make these disabled people communicable is our prime objective.

Dataset description

The provided dataset consists of 54049 images of American sign language alphabets performed by more than 40 people for 32 standard Arabic signs and alphabets. The number of images per class differs from one class to another. Each distinct hand gesture indicates some meaningful information. Estimatedly, there are around 1500 images per class and each class represents a different meaning by its hand gesture or sign. Pictorially, the sample image of each class along with its label is represented in the figure below.



For some storage schemes, 32 folders are created and each folder consists of around 1500 images incorporating differently aged people's hand gestures in different environments. The directories containing these folders are treated as training and validation datasets for the model which will be explained later in this section.

Now, before talking about the model used, it is mandatory to undergo data preprocessing to make the dataset more consistent and compatible to the model as an input. So, how the data preprocessing is done is elaborated in the next section.

Data preprocessing

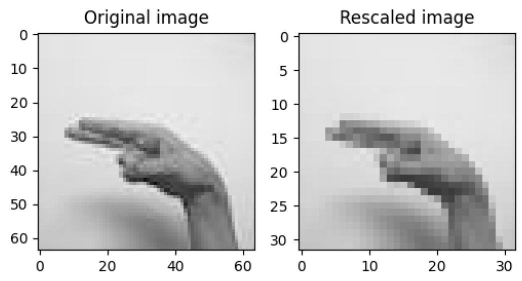
The data preprocessing involves the transformation applied to the data before feeding it to the model for training/testing. So, what changes are performed on the dataset is described below.

Removing data imbalancement

As already mentioned, the number of images per class differs from each other. This imbalancement among the classes may degrade the training performance of the model. Thus, to avoid this imbalancement, there must be an equal number of images among all classes. This imbalancement is removed by looping over each class folder to get the list of filenames of all the images per class. During each iteration, 1000 images are picked randomly from the current class folder and the rest are removed. Resultantly, a total 32000 images are filtered by summing up 1000 images of all the classes.

Data rescaling

The images contained in each class have the dimensions of (64x64). In order to keep the computations while training less complex and fast, the images can be rescaled into (32x32) following the same ratio of dimensionality. Rescaling is explained pictorially by the following figure.



Data augmentation

The data augmentation technique is widely used to increase the size of the training dataset by generating the artificial modified versions of the original images from the training dataset.

The technique results in a more diverse and consistent sequence of images and this may further result in creating more generalized and skillful deep learning models. The technique helps to avoid overfitting and underfitting of the model by applying several optional modifications to the training images. In this case, the following augmented changes are performed on the training images through ImageDataGenerator provided by the keras API.

Width shift range

This augmentation technique includes the horizontal shifting of the object to the left or right upto to the defined limit.

Height shift range

This steps includes the vertical shifting of the targeted object to up and down upto a certain limit.

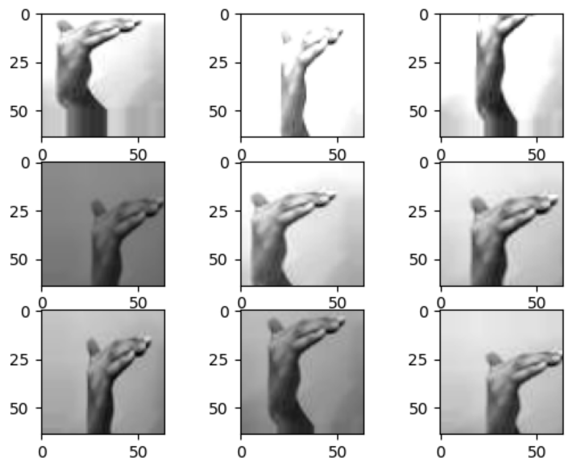
Brightness range

This augmentation technique involves the random darkening and brightening the images upto to a certain limit.

Zoom range

This augmentation technique randomly removes or add the pixels into the images for zoom in or zoom out upto the provided limitation.

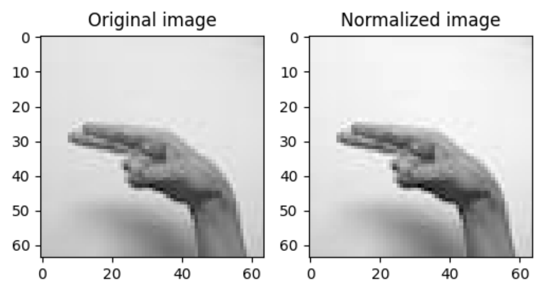
All the above-mentioned augmentation techniques are performed by passing parameters with their limitations to the ImageDataGenerator class provided by keras API. The transformations of the original image can be seen in the following figure.



Considering the preceding figure, it includes various augmented images generated from the one original image belonging to class 'khaa'. These images are then converted into normalized images and this normalization process is explained in the next section.

Data normalization

This step performs the normalization process on each image of the dataset. Usually, the pixel values in the image range from 0 to 255. But, these values must be rescaled before providing these images to the model as an input. So, the normalization will rescale these pixel values in the range of (0, 1). This rescaling will keep the model easy to learn and train in a fast way. So, this process is explained by the figure below.



Considering the above figure, there is some contrast difference between the two images. Normalized image is a bit more clear and bright than the original image. So, normalized images are more adaptable and easy to learn for the model to train.

Data splitting

The data used to build the model comes from multiple types of datasets. There are three different purposeful datasets for any computer vision project to analyze, make some comparisons and improve the performance of the model. In particular, these three different types of datasets are used in different stages of the creation of any machine learning model. These three distinct datasets are stated below:

Training dataset

The training set is a dataset on which the model is trained for learning weights or features. Initially, the model is fitted on the training dataset and in our case specifically, 80 percent of the whole dataset is used for the training dataset which is approximately 25600 images.

Validation dataset

The model is fitted on the validation dataset for the unbiased evaluation of itself during training. It validates the performance of the model based on how well the model is learning its weights before it is used for the real-time testing on the testing dataset. In our scenario of sign language recognition, 20 percent of the dataset is used which is equivalent to 6400 images.

Test dataset

After the completion of training and validation phenomena, the test dataset is used for testing the model and to measure the goodness of how well the model is trained. For this, 960 samples are used for the test dataset since there are 30 test images for each sign alphabet. This self-generated test set is created in order to measure the model's ability to generalize. More importantly, this test set is not collected from the 32000 images.

Model selection

After the dataset is fully preprocessed, it is then fed to the model network in the form of compatible input fashion for training. But before starting this time-consuming process, it is necessary to ensure the best possible selection of the deep neural network considering the problem domain. There are various frameworks which can be used in this case like tensorflow, keras, pytorch etc. Each framework has its own pros and cons considering the problem domain, keras is used. So, to ensure the best possible fit, there are several pretrained models available in the keras library. Those pretrained models are trained at the ImageNet dataset to provide state of the art results in the domain of image classification. So, here the question arises that what is ImageNet dataset and what classes the ImageNet dataset constitutes is explained briefly in the next section

ImageNet dataset

ImageNet is a large collection of annotated images publicly made available for computer vision research. This large-scale collection of images is a critical resource for analyzing, training and testing the machine learning algorithms. There are around 14 million images and 1000 categories or classes in this dataset and this dataset is also used for large-scale visual recognition challenge competitions. The pretrained models provided by the keras.applications python package are also called complex functional models because these functional models are trained on the ImageNet dataset. These pre-trained models are capable enough to classify any image that falls into these categories of images.

Keras pretrained models

As mentioned before briefly, keras applications constitute several pre-trained deep learning models available in its repository. The pretrained weights are also available alongside these models. So, these models are further used for custom object detection, image classification etc. But considering the domain problem, image classifiers are filtered from these pre-trained models and not the object detectors because the case requires performing the image classification. The selection is made considering the complexity and the nature of the hand gesture dataset. The selected pretrained models with their results are mentioned in the below table.

Model	**Size(MB)**	**Top-1 acc**	**Top-5 acc**	**Parameters**	**Depth**
Xception	88	0.790	0.945	22,910,480	126
VGG16	528	0.713	0.901	138,357,544	23

ResNet50	98	0.749	0.921	25,636,712	-
InceptionV3	92	0.779	0.937	23,851,7841	159
MobileNet	16	0.704	0.895	4,253,864	88
EfficientNet	29	0.810	0.922	19,466,823	-

In the above table, it is important to note that all the models are trained using ImageNet dataset. Every model has its own size, accuracy and number of parameters along with the architecture depth. These models are retrained further on the arabic hand gesture classification dataset comprising 32 classes. After training, the best possible fit is considered for the case. So, the final selection is made after custom-training these models on the given dataset. So, the custom training begins in the succeeding section.

Model compilation

This section includes the detailed analysis about how to perform the complex training process to produce state of the art results. To custom train the selected keras pretrained models, transfer learning is the only choice. So, what is transfer learning and how to perform it is explained in the below section.

Transfer learning

Transfer learning refers to the situation when the knowledge learned in one task or domain is reused to improve the generalization in another domain. From machine learning's perspective, it can be defined as reusing the saved weights of any pre-trained model to improve the accuracy or to custom-train your own model. In order to use the weights of any pre-trained model e.g. VGG16, EfficientNet, some modifications have to be made to make the model compatible to train on another dataset. The modifications performed on the neural network are elaborated in the next part.

Modifications in pretrained models

Basically, three modifications are made to make the model ready to train in the given case. Those modifications are briefly explained below.

Input layer modification

Input layer is changed considering the dimensions and size of the input images. In the current case, the images are of the size (64, 64) and the ImageDataGenerator class receives the input shape parameter to automatically prepare the input layer of the model to initialize the training process.

Output layer modification

The output layer is modified depending upon the number of classes. Number of neurons is equivalent to the number of classes at the output layer.

Addition of layers

Some dense(fully-connected) layers are added at the bottom of these ready-made architectures just before the output layer to make the model more effective and suitable for use in accordance with the complexity and the format of the dataset

Optimizer

An optimizer is one the final arguments required to compile the model before training phenomena. There are different variants of optimizer available in the keras library like SGD(stochastic gradient descent), RMS(root mean square) and adam etc. For hand gesture recognition, adam is used. The ‘adam’ optimizer is used to reduce the loss calculated after each epoch while training. This optimizer uses the stochastic gradient descent method that is based on adaptive estimation of first-order and second-order moments. This method is computationally efficient, occupies less memory, invariant to diagonal rescaling of gradients and is best-suited for the problems that require complex processing in terms of data/parameters.

Loss function

The loss function is the necessary argument used in the model compilation. The loss function calculates the training or/and validation losses after each epoch during the training phenomena. This measurement provides the level of goodness that shows how well the model is being trained. Increase in loss degrades the model performance and decrease in loss optimizes the model performance. There are several built-in classes available in the keras library for calculating the loss during training. The selection depends upon the nature of the dataset. In case of image classification having more than two classes to predict, ‘categorical_crossentropy’ class is used. This class computes the cross entropy loss between the ground truth values and the predicted values resulting from model predictions.

Training callbacks

The callbacks are used to perform certain actions at different stages of the training process. These are useful when a developer wants to save some kind of model information during or after the training process. These callbacks can be performed before and after the single batch, start or end of an epoch etc. There are various callbacks provided by the keras library but in this case, few callbacks are considered to be used during training of the model. These callbacks have their own distinct functionality and purpose which are explained briefly below.

ModelCheckpoint

This callback is used to save the keras model or the model weights after some intervals during the training process. The save model file can be used further to load and start the training again or for some testing or evaluation purposes. This callback can be used several ways, providing the optional arguments to the callback class. The options are described precisely below.

- Whether to save the model possessing the best performance or to save the model file after each epoch, disregarding the model performance. In the specific case, the best model file is saved if the model is improved as compared to past performances.
- The callback can be only used based on the monitored quantity. The quantity to be monitored and whether it should be maximized or minimized. The monitored quantity can have four options: train_accuracy, train_loss, validation_accuracy and validation loss. In this case, validation_accuracy is termed as a monitored quantity.
- The callback also provides the option of at what frequency it should save the model file. The model checkpoint file is saved after each epoch analyzing the validation accuracy.

So, there are several other options available to use this callback but the above mentioned options are used in the given case.

EarlyStopping

This callback is used to stop the training process automatically when model performance stops improving upto a certain limit based on some monitored quantity. As previously mentioned, the monitored quantity in the given case is validation accuracy. The training process terminates when the validation accuracy stops improving upto to a certain number of epochs. There are several optional arguments used to perform this early stopping and those options are explained briefly below.

- The validation accuracy(monitored quantity) qualifies to be improved when increased with the minimum change. This minimum change can be passed as a parameter to the early stopping class as a min_delta argument. This min_delta option controls the threshold of change to be qualified as improved validation accuracy.

- Patience option controls when the training process is terminated automatically. When the model performance starts degrading for the defined number of epochs, the training automatically terminates. This termination is caused when model degradation crosses the patient value defined in the early stopping class.

CSVLogger

This callback is used to save the training statistics in a file at runtime during the execution of training phenomena. The result of each epoch is saved in that file. In the given case, a comma-separated log file is used to save the results after each epoch.

So, the above-mentioned callbacks are passed as an array to fit() function in order to apply these callback operations for better hunting the most optimal model and saving the evaluation matrices.

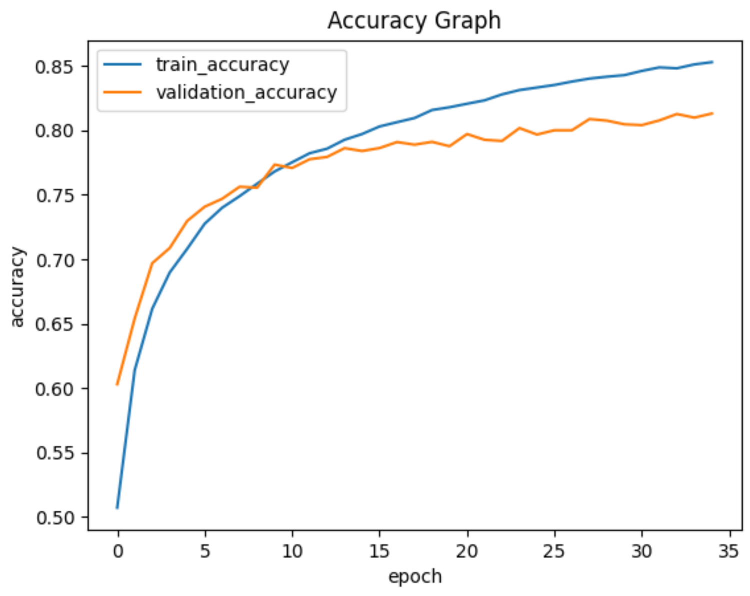
After defining the training and validation generators to make the dataset ready-to-train, finalizing the optimizer, loss function and applying the training callbacks, the model compiles successfully. After successful compilation, the keras model is now ready to train, which is explained in the next section.

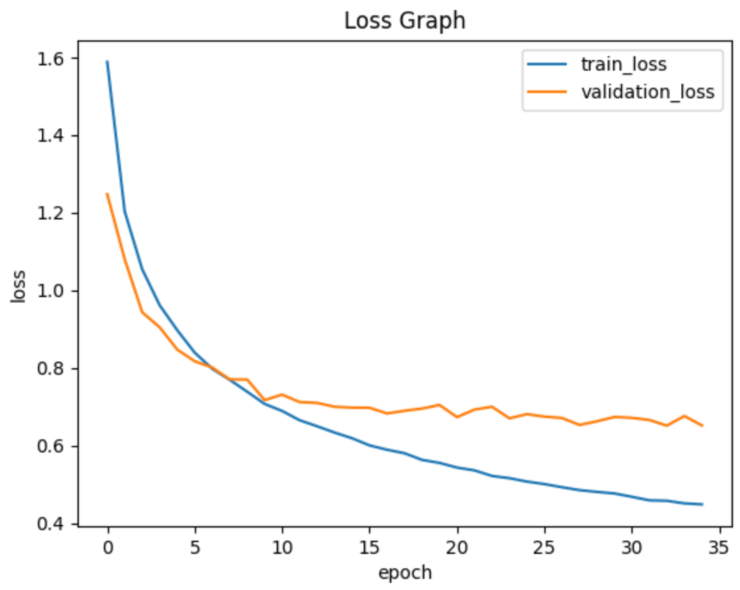
Model training

At this stage, the model instance is fitted to the fit() function to start the training process. This function trains the model for a fixed number of epochs (iterations on a dataset) using training and validation generators incorporating the pre-processed images and some other required attributes as mentioned in the preceding sections. The model had been trained for 25-30 epochs in almost 10 hours and each epoch took 2000 steps to complete. The number of steps per epoch depends upon two factors: batch size and the training number of images. The number of training images is 128000 and the batch size is 64. Resultantly, the number of steps per epoch is calculated by dividing the number of training images by batch size which is equivalent to 2000 steps in the given case.

Best performant

Several keras pretrained models are taken into trial to find the best suited model for the given case. Eventually, EfficientNetB4 has the best performance in terms of accuracy and loss, So, to analyze the best-suited model for the given case, it is recommended to plot the graphs for accuracy as well as loss. So, the graphical approach is used to represent the whole training history of the model with epoch count on the x-axis and the accuracy/loss on the y-axis. The parameters on the y-axis include validation loss, validation accuracy, training loss and training accuracy. The following graphs provide deep insight about how well the EfficientNetB4 model is trained. It can be seen that the accuracies in the first graph and the losses in the second graph are converging towards each other steadily upto 10th epoch. After the 10th epoch, both the accuracies and losses diverge from each other and thus, indicating that the model has learned the weights well. So, the model is stopped till the 25th epoch to avoid overfitting because the model has learned the input features to better classify the unforeseen hand gestures. The behavior can be seen graphically as below.





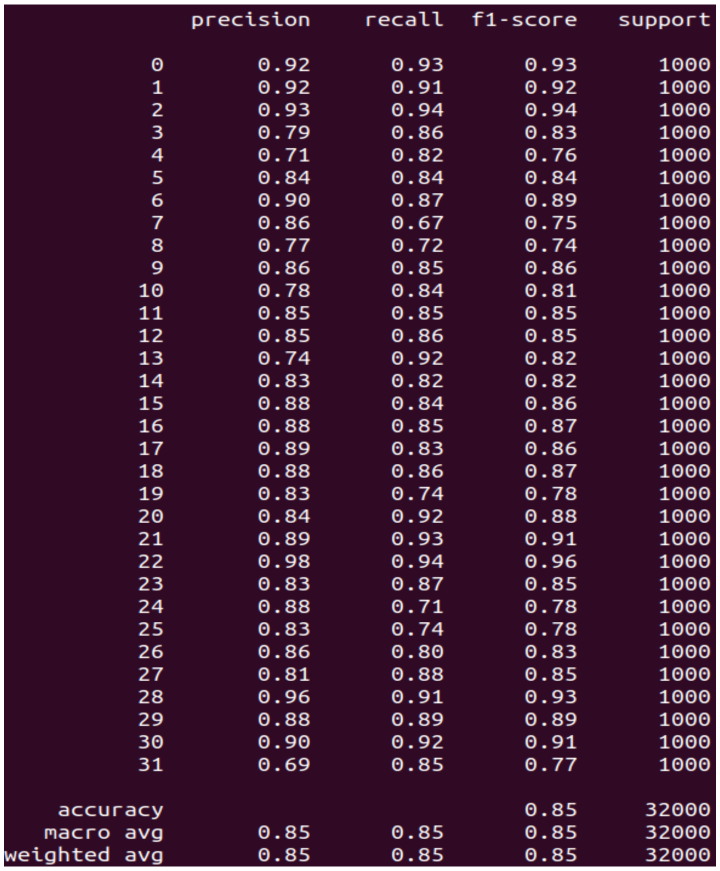
After having the graphical analysis on the training and the validation performance of the model, the next general step is to test the model on unforeseen data. The random and unforeseen evaluation tests the real intelligence of the trained model about how well the model has learned from the input information. This unforeseen behavior is explained precisely in the next section.

Model evaluation

After the model is trained, testing is required to measure the real performance of the model on unseen data which the model has not encountered yet. Different programmatic approaches are provided by the scikit-learn library to test the performance of the trained model. The statistical evaluation is done using two methods and that are confusion matrix and classification report. These two approaches are explained in detail in next two succeeding sections.

Classification report

To describe the performance of the classification model on a test dataset, classification report is used as shown under:



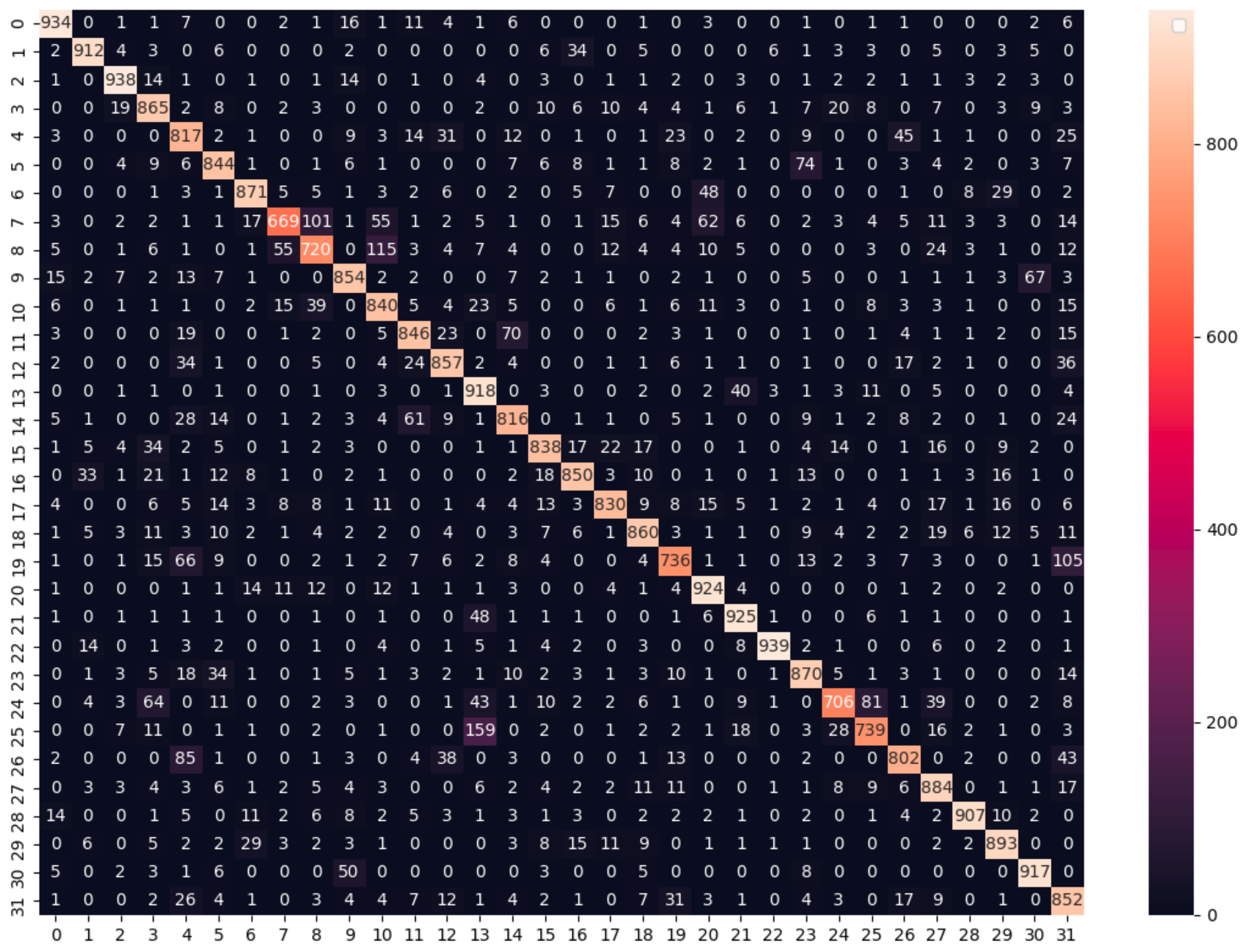
```
              precision    recall  f1-score   support

           0       0.92      0.93      0.93      1000
           1       0.92      0.91      0.92      1000
           2       0.93      0.94      0.94      1000
           3       0.79      0.86      0.83      1000
           4       0.71      0.82      0.76      1000
           5       0.84      0.84      0.84      1000
           6       0.90      0.87      0.89      1000
           7       0.86      0.67      0.75      1000
           8       0.77      0.72      0.74      1000
           9       0.86      0.85      0.86      1000
          10       0.78      0.84      0.81      1000
          11       0.85      0.85      0.85      1000
          12       0.85      0.86      0.85      1000
          13       0.74      0.92      0.82      1000
          14       0.83      0.82      0.82      1000
          15       0.88      0.84      0.86      1000
          16       0.88      0.85      0.87      1000
          17       0.89      0.83      0.86      1000
          18       0.88      0.86      0.87      1000
          19       0.83      0.74      0.78      1000
          20       0.84      0.92      0.88      1000
          21       0.89      0.93      0.91      1000
          22       0.98      0.94      0.96      1000
          23       0.83      0.87      0.85      1000
          24       0.88      0.71      0.78      1000
          25       0.83      0.74      0.78      1000
          26       0.86      0.80      0.83      1000
          27       0.81      0.88      0.85      1000
          28       0.96      0.91      0.93      1000
          29       0.88      0.89      0.89      1000
          30       0.90      0.92      0.91      1000
          31       0.69      0.85      0.77      1000

    accuracy                           0.85     32000
   macro avg       0.85      0.85      0.85     32000
weighted avg       0.85      0.85      0.85     32000
```

The classification report shows the representation of the main classification matrix on a per-class basis. This visual report is giving better and deeper intuition about the behavior of the classifier showing the functional weaknesses of the trained model in many analytical aspects.

Here, the support column shows the count of test images as per class e.g. all classes constitute a total of 1000 test images. F1-score is the mean of precision and recall such that the best score is 1.0 and the worst is 0.0. The ability of the classifier to find all positive instances (correct predictions) is defined by the recall column numerically. In the recall column, all the classes are showing the true predictions of more than 80 percent except five classes. The report is showing the testing accuracy to be 85 percent which is the real predictive result of our classifier.

Confusion matrix

To describe the per-class performance of the classification model on a test dataset in a more insightful way, the confusion matrix is used as shown under.



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	934	0	1	1	7	0	0	2	1	16	1	11	4	1	6	0	0	0	1	0	3	0	0	1	0	1	1	0	0	0	2	6
1	2	912	4	3	0	6	0	0	0	2	0	0	0	0	0	6	34	0	5	0	0	0	6	1	3	3	0	5	0	3	5	0
2	1	0	938	14	1	0	1	0	1	14	0	1	0	4	0	3	0	1	1	2	0	3	0	1	2	2	1	1	3	2	3	0
3	0	0	19	865	2	8	0	2	3	0	0	0	0	2	0	10	6	10	4	4	1	6	1	7	20	8	0	7	0	3	9	3
4	3	0	0	0	817	2	1	0	0	9	3	14	31	0	12	0	1	0	1	23	0	2	0	9	0	0	45	1	1	0	0	25
5	0	0	4	9	6	844	1	0	1	6	1	0	0	0	7	6	8	1	1	8	2	1	0	74	1	0	3	4	2	0	3	7
6	0	0	0	1	3	1	871	5	5	1	3	2	6	0	2	0	5	7	0	0	48	0	0	0	0	0	1	0	8	29	0	2
7	3	0	2	2	1	1	17	669	101	1	55	1	2	5	1	0	1	15	6	4	62	6	0	2	3	4	5	11	3	3	0	14
8	5	0	1	6	1	0	1	55	720	0	115	3	4	7	4	0	0	12	4	4	10	5	0	0	0	3	0	24	3	1	0	12
9	15	2	7	2	13	7	1	0	0	854	2	2	0	0	7	2	1	1	0	2	1	0	0	5	0	0	1	1	1	3	67	3
10	6	0	1	1	1	0	2	15	39	0	840	5	4	23	5	0	0	6	1	6	11	3	0	1	0	8	3	3	1	0	0	15
11	3	0	0	0	19	0	0	1	2	0	5	846	23	0	70	0	0	0	2	3	1	0	0	1	0	1	4	1	1	2	0	15
12	2	0	0	0	34	1	0	0	5	0	4	24	857	2	4	0	0	1	1	6	1	1	0	1	0	0	17	2	1	0	0	36
13	0	0	1	1	0	1	0	0	1	0	3	0	1	918	0	3	0	0	2	0	2	40	3	1	3	11	0	5	0	0	0	4
14	5	1	0	0	28	14	0	1	2	3	4	61	9	1	816	0	1	1	0	5	1	0	0	9	1	2	8	2	0	1	0	24
15	1	5	4	34	2	5	0	1	2	3	0	0	0	1	1	838	17	22	17	0	0	1	0	4	14	0	1	16	0	9	2	0
16	0	33	1	21	1	12	8	1	0	2	1	0	0	0	2	18	850	3	10	0	1	0	1	13	0	0	1	1	3	16	1	0
17	4	0	0	6	5	14	3	8	8	1	11	0	1	4	4	13	3	830	9	8	15	5	1	2	1	4	0	17	1	16	0	6
18	1	5	3	11	3	10	2	1	4	2	2	0	4	0	3	7	6	1	860	3	1	1	0	9	4	2	2	19	6	12	5	11
19	1	0	1	15	66	9	0	0	2	1	2	7	6	2	8	4	0	0	4	736	1	1	0	13	2	3	7	3	0	0	1	105
20	1	0	0	0	1	1	14	11	12	0	12	1	1	1	3	0	0	4	1	4	924	4	0	0	0	0	1	2	0	2	0	0
21	1	0	1	1	1	1	0	0	1	0	1	0	0	48	1	1	1	0	0	1	6	925	1	0	0	6	1	1	0	0	0	1
22	0	14	0	1	3	2	0	0	1	0	4	0	1	5	1	4	2	0	3	0	0	8	939	2	1	0	0	6	0	2	0	1
23	0	1	3	5	18	34	1	0	1	5	1	3	2	1	10	2	3	1	3	10	1	0	1	870	5	1	3	1	0	0	0	14
24	0	4	3	64	0	11	0	0	2	3	0	0	1	43	1	10	2	2	6	1	0	9	1	0	706	81	1	39	0	0	2	8
25	0	0	7	11	0	1	1	0	2	0	1	0	0	159	0	2	0	1	2	2	1	18	0	3	28	739	0	16	2	1	0	3
26	2	0	0	0	85	1	0	0	1	3	0	4	38	0	3	0	0	0	1	13	0	0	0	2	0	0	802	0	2	0	0	43
27	0	3	3	4	3	6	1	2	5	4	3	0	0	6	2	4	2	2	11	11	0	0	1	1	8	9	6	884	0	1	1	17
28	14	0	0	1	5	0	11	2	6	8	2	5	3	1	3	1	3	0	2	2	2	1	0	2	0	1	4	2	907	10	2	0
29	0	6	0	5	2	2	29	3	2	3	1	0	0	0	3	8	15	11	9	0	1	1	1	1	0	0	0	2	2	893	0	0
30	5	0	2	3	1	6	0	0	0	50	0	0	0	0	0	3	0	0	5	0	0	0	0	8	0	0	0	0	0	0	917	0
31	1	0	0	2	26	4	1	0	3	4	4	7	12	1	4	2	1	0	7	31	3	1	0	4	3	0	17	9	0	1	0	852

The above error matrix or table allows the statistical visualization of the testing performance of the trained model. Actual outputs and predictive outputs are plotted as x-axis and y-axis. It allows easy identification of confusion between the classes and gives better insight as compared to other reports.

Comparative analysis with the past work

The previous work done with the aim of hand gesture recognition is not so generalized and authentic to use in different environments.

Dataset comparison

Previous papers published include the dataset consisting not more than 20 classes but in the given case, the dataset contains around 32 classes constituting nearly 160000 images. In most cases, considering the previous work, the dataset is converted into grayscale images and this dataset transformation sometimes degrades the model performance. Secondly, several data augmentation techniques are applied in the given case to make the solution adaptable to different environments.

Generalization vs specialization

The model trained in the current case is more generalized as compared to the models analysed in past papers. The datasets and approaches analyzed in the previous work indicate that the solution is not adaptable and generalized. So in the given case, the specialization problem is now resolved because the model is trained in a diverse environment.

Applicability

Considering the given case, the solution is more applicable compared to the past work. It can be adapted in real-time environments as well. Briefly, the comparative analysis is summarized in the table below.

Current work	**Past work**
More generalized solution	Specialized solution(specific to given case)
Big dataset(constituting more than 30 classes)	Limited dataset(in most cases, less than 20 classes)
Model trained on RGB images(improves the model performance)	Model trained on grayscale images(change of degrading the model performance.)
Applied data augmentation	No data augmentation techniques(specific

techniques(brightness, zoom, object positioning) improves the performance	to one environment), Due to this, may not be performed in other environments.
Can be applicable in real-time	In most cases, not applicable in real-time

Tools and Techniques

The programming language and python packages used for data preprocessing, model training and model evaluation are mentioned below.

- **Programming language:** python.
- **Data preprocessing:** opencv, os, shutil, tqdm, numpy, pandas
- **Model training:** tensorflow, keras
- **Model evaluation:** scikit-learn, matplotlib, seaborn

References

- https://stackoverflow.com/questions/54631583/how-can-i-use-imagedatagenerator-class-to-generate-either-train-and-label-as-ima
- https://machinelearningmastery.com/how-to-configure-image-data-augmentation-when-training-deep-learning-neural-networks/
- https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/image/ImageDataGenerator
- https://scikit-learn.org/stable/modules/model_evaluation.html#classification-metrics
- https://machinelearningmastery.com/what-is-generalization-in-machine-learning/
- https://www.tensorflow.org/tutorials/keras/save_and_load
- https://keras.io/api/models/sequential/
- https://keras.io/api/callbacks/
- https://keras.io/api/preprocessing/
- https://keras.io/api/optimizers/adam/
- https://keras.io/api/applications/