VGG16 CNN based Braille Cell classifier model for Translation of Braille to Text.

Vishwanath Venkatesh Murthy [1]* , M Hanumanthappa[2]

[1]Research scholar, Department of Master of Computer Applications, Bangalore University, Bengaluru
[2] Department of MCA, Bangalore University, India.

Abstract: The visually impaired individuals can use only Braille documents as a medium of education. These Braille documents are scripted on metallic plates which are heavy and bulky in nature. Maintaining these documents are very cumbersome due to their wear and tear tendency in unsupported weather conditions. It is also a tedious task to transport them over the globe and chances of damaging the documents is more. The main purpose of the research is to translate the Braille documents to natural text. Once the document is text form, it can be distributed over the globe using world wide web and can be reproduced to Braille document whenever needed. As the document is in digital form, maintaining Braille document for latter reproducing will be very easy. Extracting the Braille Character cell from the Braille document is the central theme of the research. In previous work the paper was published on segmentation and cropping of Braille cells using its physical properties was published. This paper presents the Deep learning based proposed model for classifying the Braille cells using VGG16 Convolution neural network model.
Keywords: Braille, CNN, CV2, Deep learning, Feature extraction, Keras, TensorFlow, Matplotlib, VGG16, YOLO.

I. INTRODUCTION

TheOptical braille recognition (OBR)is a process that translate the Braille document to text. The OBR process uses various steps like(i) preprocessing, (ii) segmentation, and (iii) feature extraction. Braille documents is a collection of Braille cell, where each cell consists of combination of 3x2raised dots embossedon typicallysolid metallic plates or Braille paper [2].

- **Challenges**

Due to metallic nature of Braille plates, they are heavy and bulky to use. Transporting such documents from one place other is tedious task [3]. Preserving the documents also requires lot of space and they tend to wear and tear due to weather conditions. Hence there is a need to Translate the Braille documents to electronic version like in computer Text. The process of translation requires first to scan the documents into image. Defects or uneven light, skew and other kind of noise can be established during the scanning process. Preprocessing and recognizing the presence of embossed Braille dots, mapping each cell and translating it to English sentence is a major challenge.

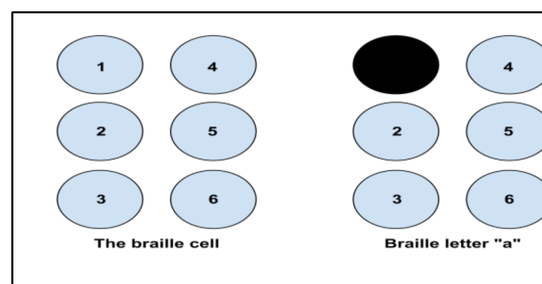


Figure-1: Braille cell arrangement

Preprocessing the image to remove all the noise introduced in Braille image is a challenge. Also, the coding standard used of Braille character is unique, where each cell is formed using combinations of six dots arranged in 3x2 matrix form as shown in figure 1. Braille cells with 3x2 matrix form are embossed on metallic plates. Sample Braille metallic plate scanned image is shown in figure 2.

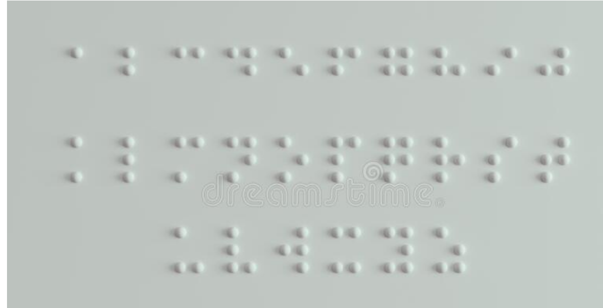


Figure-2: Sample Braille document.

Recognizing the presenceof embossed Braille dots, Marking the cell boundary of each cell is another major Challenge in the research work.
Finally recognized braille cells needs to be mapped to the respective alphabet in English. The recognized braille cells may contain huge variance representing the same alphabet as shown in figure 3.

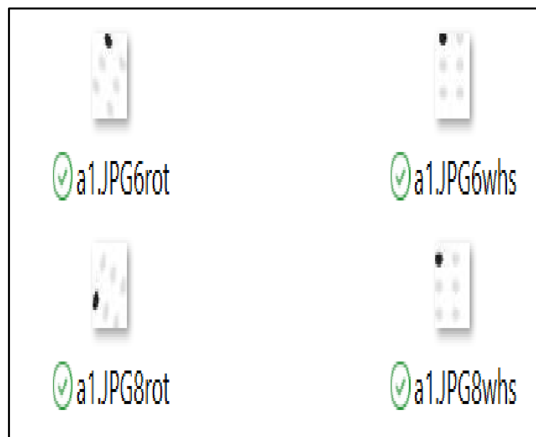


Figure-3: Variance in Sample of Braille Alphabet 'A'

- **Aim of the research**

To provide the better solution to convert Braille scanned documents into computer readable form or any other non-Braille language likeEnglish.

II. PROPOSEDMODEL

In the previous work, the initial two stages of the objective have been achieved successfully. The objective in stage-3 is obtained by the proposed model that usesVGG16 Convolution Neural network for the purpose of mapping the Braille Cells to respective Alphabets.

- **Dataset used**

The dataset needed for training and testing of the CNN model; the standard dataset "Braille Character Dataset" is downloaded from the Kaggle website to the local directory "E:\Braille\DATASET". The Braille character dataset was divided into 26 categories as per the English Alphabets from 'A' to 'Z'. For every category a separate folder is created. Each folder contains set of 60 images that belong to same alphabet. Additional folders were created for special symbols like 'space', 'capital' and 'Number' with 10 images each. With the 26 alphabet

folders and 3 special folders, total images considered for the model are 26x60 + 10x3= 1590 images. Hence our trained model can recognize 60 variations of each of the Braille cell. The Sample dataset with 60 sets for category (Directory) 'A' is shown in figure 4.

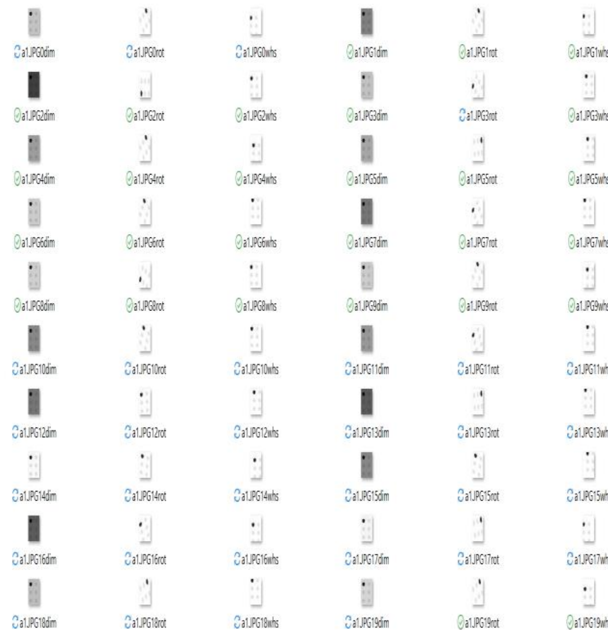


Figure-4: 60 samples images of folder 'A' that belong to same alphabet class 'A'.

It is clearly seen from figure 4, the amount of variance available in each of these samples that belong to same alphabet 'A'.

- **Algorithm of proposed model**

Programming language used for developing the model is Python-3 using Jupyter notebook 6.4.11 which is the latest one. Various other tools and libraries used areNumPy, OpenCv (cv2), random, Matplotlib, TensorFlow, Keras. TFLearn and Pickle.The algorithm for Braille classifier model is given below.

Algorithm:

1. *Download the Braille dataset from Kaggle to local directory.*
2. *Import transfer learning model VGG16*
3. *Preprocess the Images.*
4. *Create the weights and feature for training, testing and validation.*
5. *Add Fully connected layers*
6. *Compile and evaluate the model*

Step-1: In the first stage create thetraining, validation, and testing directory. Each of the directories are added with 29 directoriesrepresenting 29 classes each for 26 alphabets and three special symbols. Each directory contains 60 images belong to same class. The 60 imagesbelonging to class 'A' and stored in directory 'A' are shown in figure-4.

Step-2: To classify the multiclass image we need to import VGG16 architecture which is done using the equation-1.

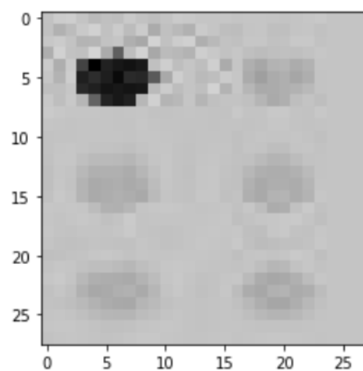
vgg16 = applications.VGG16 (include_top=False, weights='imagenet') --(1)

The VGG16 model summery that uses 16 layers is shown in figure-5.

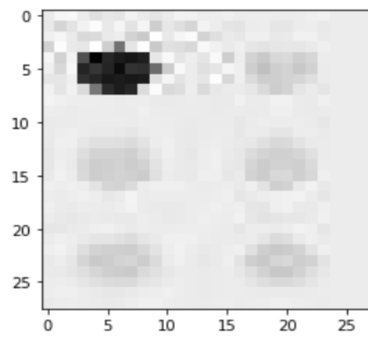
```
vgg16.summary()

Model: "vgg16"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, None, None, 3)]   0

 block1_conv1 (Conv2D)       (None, None, None, 64)    1792

 block1_conv2 (Conv2D)       (None, None, None, 64)    36928

 block1_pool (MaxPooling2D)  (None, None, None, 64)    0

 block2_conv1 (Conv2D)       (None, None, None, 128)   73856

 block2_conv2 (Conv2D)       (None, None, None, 128)   147584

 block2_pool (MaxPooling2D)  (None, None, None, 128)   0

 block3_conv1 (Conv2D)       (None, None, None, 256)   295168

 block3_conv2 (Conv2D)       (None, None, None, 256)   590080

 block3_conv3 (Conv2D)       (None, None, None, 256)   590080

 block3_pool (MaxPooling2D)  (None, None, None, 256)   0

 block4_conv1 (Conv2D)       (None, None, None, 512)   1180160

 block4_conv2 (Conv2D)       (None, None, None, 512)   2359808

 block4_conv3 (Conv2D)       (None, None, None, 512)   2359808

 block4_pool (MaxPooling2D)  (None, None, None, 512)   0

 block5_conv1 (Conv2D)       (None, None, None, 512)   2359808

 block5_conv2 (Conv2D)       (None, None, None, 512)   2359808

 block5_conv3 (Conv2D)       (None, None, None, 512)   2359808

 block5_pool (MaxPooling2D)  (None, None, None, 512)   0

=================================================================
Total params: 14,714,688
Trainable params: 14,714,688
Non-trainable params: 0
_________________________________________________________________
```

Figure-5: VGG16 Model summary.

Step-3: Each image is converted to grayscale using cv2 library before inputting to the model. Theoriginal image and the sample output grayscale image is shown in figure-6.



(a) original image



(b) Grayscale image

Figure-6: Original image and Grayscale image

We must resize the image to establish a base size for all images fed into the model.

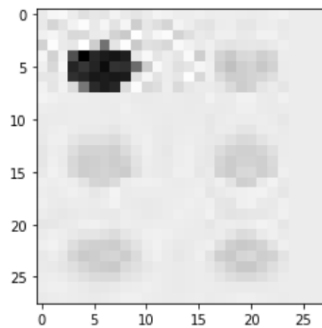


Figure 7(a) before resizing the image (28x28)

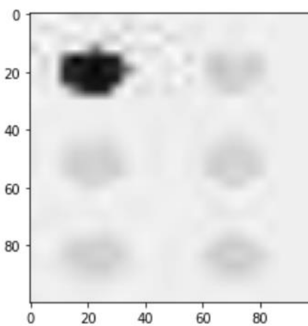


Figure 7 (b) After resizing the image (100x100)

The original image size is found to be 28x28. The new image is resized to 100x100 as a base size for our model. The figure-7 shows the images before and after resizing.

Step-4: In step 4, vgg16.predict_generator () method is used to extract the features from training, test and validation dataset and is coveted to NumPy array as shown in equation-2.

bottleneck_Train_features = vgg16.predict_generator (generator, predict_Train_size) – (2)

Step-5: In this stage convolution base is added to the last Fully connected layer. First the sequential model is initialized. Fully connected layers added as convolution base which include flatten layer, Dense1 layer, Dropout layer, Dense2 layer followed by again Dropout layer. The LeakyReLu activation function was used in these layers. Finally, the SoftMax activation Dense layer is added as output layer with 29 classes.

The Convolution model summary is shown in figure-8.

```
: model.summary()

Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten (Flatten)           (None, 25088)             0

 dense (Dense)               (None, 100)               2508900

 dropout (Dropout)           (None, 100)               0

 dense_1 (Dense)             (None, 50)                5050

 dropout_1 (Dropout)         (None, 50)                0

 dense_2 (Dense)             (None, 29)                1479

=================================================================
Total params: 2,515,429
Trainable params: 2,515,429
Non-trainable params: 0
_________________________________________________________________
```

Figure 8:Braille Convolution model Summary

Step6: In step6, the model is compiled with 20 epochs and 32 batch size. The output of the epochs is shown in figure-9.

```
32/32 [==============================] - 2s 45ms/step - loss: 3.2787 - acc: 0.0854 - val_loss: 2.9372 - val_acc: 0.2038
Epoch 2/20
32/32 [==============================] - 1s 30ms/step - loss: 2.9023 - acc: 0.1842 - val_loss: 2.5456 - val_acc: 0.4295
Epoch 3/20
32/32 [==============================] - 1s 35ms/step - loss: 2.6654 - acc: 0.2441 - val_loss: 2.3051 - val_acc: 0.5392
Epoch 4/20
32/32 [==============================] - 1s 35ms/step - loss: 2.4463 - acc: 0.2983 - val_loss: 2.0588 - val_acc: 0.5799
Epoch 5/20
32/32 [==============================] - 1s 35ms/step - loss: 2.2965 - acc: 0.3627 - val_loss: 1.9407 - val_acc: 0.6238
Epoch 6/20
32/32 [==============================] - 1s 35ms/step - loss: 2.1974 - acc: 0.3901 - val_loss: 1.8380 - val_acc: 0.6458
Epoch 7/20
32/32 [==============================] - 1s 34ms/step - loss: 2.0677 - acc: 0.4149 - val_loss: 1.6738 - val_acc: 0.6552
Epoch 8/20
32/32 [==============================] - 1s 35ms/step - loss: 1.9386 - acc: 0.4608 - val_loss: 1.6224 - val_acc: 0.6959
Epoch 9/20
32/32 [==============================] - 1s 35ms/step - loss: 1.8547 - acc: 0.4786 - val_loss: 1.4634 - val_acc: 0.7398
Epoch 10/20
32/32 [==============================] - 1s 35ms/step - loss: 1.7912 - acc: 0.4959 - val_loss: 1.3921 - val_acc: 0.7273
Epoch 11/20
32/32 [==============================] - 1s 35ms/step - loss: 1.7050 - acc: 0.5322 - val_loss: 1.3112 - val_acc: 0.7618
Epoch 12/20
32/32 [==============================] - 1s 35ms/step - loss: 1.6529 - acc: 0.5424 - val_loss: 1.1874 - val_acc: 0.7806
Epoch 13/20
32/32 [==============================] - 1s 35ms/step - loss: 1.5450 - acc: 0.5685 - val_loss: 1.1584 - val_acc: 0.7868
Epoch 14/20
32/32 [==============================] - 1s 35ms/step - loss: 1.4915 - acc: 0.5825 - val_loss: 1.1696 - val_acc: 0.7492
Epoch 15/20
32/32 [==============================] - 1s 35ms/step - loss: 1.4508 - acc: 0.5940 - val_loss: 1.0478 - val_acc: 0.8276
Epoch 16/20
32/32 [==============================] - 1s 30ms/step - loss: 1.3681 - acc: 0.6163 - val_loss: 0.9730 - val_acc: 0.8307
Epoch 17/20
32/32 [==============================] - 1s 31ms/step - loss: 1.3348 - acc: 0.6233 - val_loss: 0.9281 - val_acc: 0.8370
Epoch 18/20
32/32 [==============================] - 1s 33ms/step - loss: 1.2623 - acc: 0.6590 - val_loss: 0.8642 - val_acc: 0.8401
Epoch 19/20
32/32 [==============================] - 1s 30ms/step - loss: 1.2780 - acc: 0.6399 - val_loss: 0.8526 - val_acc: 0.8495
Epoch 20/20
32/32 [==============================] - 1s 35ms/step - loss: 1.1855 - acc: 0.6756 - val_loss: 0.7911 - val_acc: 0.8777
```

Figure 9: evaluation of Model with 20 epochs

It is quick to note that the accuracy of thesystem improves with each pass of the training dataset.

III. COMPARATIVE STUDY OF TOOLS

Many authors have worked on the improvisation of OBR process. The table 1 shows the various tools used by the researchers and the accuracy obtained by them. The available tools include MATLAB, OpenCV, SimpleCV, Mahotas, GNU Octave, ScikitImage, Scilab, CVIPTools, Matplotlib, TensorFlow, CUDA, Keras, CAFFE, YOLO, PyCharm, RStudio, Spyder, Pillow etc.

Table-1: Tools used by various authors

No	Authors	Tools used	Accuracy
1	Shokat, S., Riaz	MATLAB	80%
2	DiegoGoncalves, Gabriel G. Santos	YOLO	NA
3	Tasleem Kausar, Sajjad Manzoor	TensorFlow,MATLAB R2018a, Keras	95.2%
4	Robert de Luna	SimpleCV	85%
5	S A Hariprasad, Kishore Kumar N	NumPy, matplotlib, cv2	89%
6	Tanmay R	TensorFlow	96%
7	Sudhir Rao Rupanagudi, Sushma Huddar	MATLAB	94%
8	TingLi, Xiaoqin Zeng	MATLAB version 2012	92%
9	Toshiaki Okamoto, Tomoyuki Shimono	TensorFlow 1.5.0, Keras 2.1.3, OpenCV 4.1	94%
10	N.D.S.M.K.De Silva	Emgu CV 3.0 (Open CV library)	98%
11	Banumathi. K. L, Jagadeesh Chandra	MATLAB	82 to 100%

IV. RESULTS AND DISCUSSION

The dataset is downloaded from the Kaggle website and stored in the local directory. The dataset of all alphabets was divided into 26 folderseachfor English Alphabets from ‘A’ to ‘Z’. Each folder hasa set of 60 images that belong to same alphabet with variations. Total images considered for the model are 26x60 = 1560 images. The said model was run on 20 epochs.The output accuracy of the model is shown in figure-10.

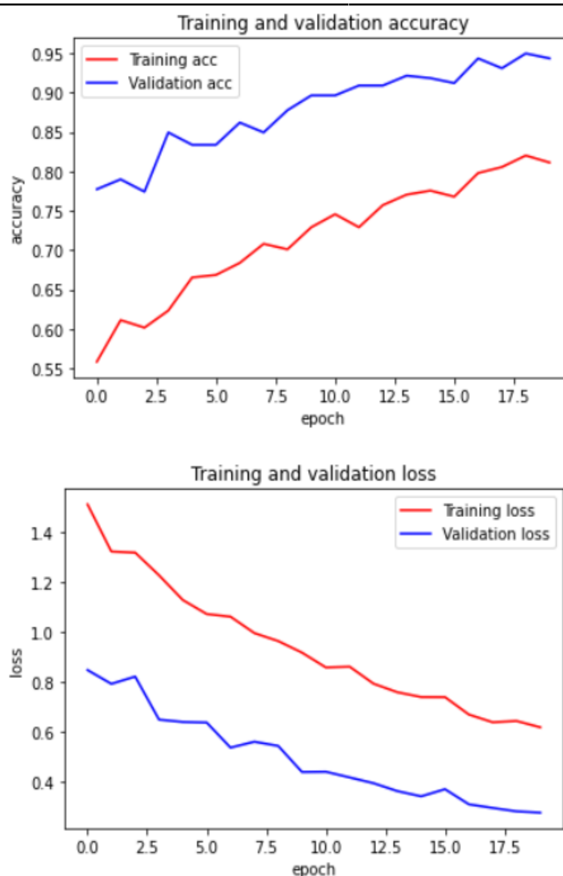


Figure-10: Validation accuracy of model.

The figure 10 shows the validation accuracy curve. It is proved by graph that the training and validation accuracy increases by increasing the no of epochs on the model while the training and validation loss decreases to great extent. The validation accuracy of the model for 20 epoch is 95%.The model is developed using python program on TensorFlow and Keras environment.

V. CONCLUSION AND FUTURE ENHANCEMENT

The proposed work classifies the segmented cell fromthe inputted image and the output is shown in a text editor.The reproduction of the English letters from the braille cellshas been faithful. To test the performance of the system,either the accuracy or the loss can be checked since one is the inverse of the other. The amount of loss wouldeventually show the accuracy of the system. For thereviewing the performance of the system, accuracy isconsidered.

The proposed work has been applied on single side Braille documents. The table 2 shows the Precision, Recalland F1 score values for sample 5 documents.

Table-2: Results of precision, recall and F1 score.

# Braille Doc	Input cells (x)	Precision	Recall	F1 score
1	545	0.99	0.96	0.97
2	861	0.98	0.95	0.96
3	996	0.97	0.93	0.94
4	327	0.99	0.97	0.97
5	734	0.96	0.95	0.96

Accuracy rate for the given Braille documents observed from table-3 is between 91% to 98% while error rate is below 9%.

Table-3: Results ofAccuracy and Error rate.

Doc No	Input cells (x)	Recognized (y)	Accuracy %	Error %
1	545	525	96.33	3.67
2	861	801	93.03	6.97
3	996	910	91.37	8.63
4	327	320	97.86	2.14
5	734	702	95.64	4.36

The proposed work has been applied on single side Braille documents. This model can be extended to recognize the double side Braille documents as a future work.

References

[1] Shokat, S., Riaz, R., Rizvi, S. S., Khan, K., Riaz, F., & Kwon, S. J. (2020). "Analysis and Evaluation of Braille to Text Conversion Methods". 2020, 3461651.

[2] Shreekanth T & V. Udayashankara, "An Application of Eight Connectivity based Two-pass Connected-Component Labelling Algorithm For Double Sided Braille Dot Recognition", International Journal of Image Processing (IJIP), Volume (8) : Issue (5) : 2014

[3] Diego Goncalves, Gabriel G. Santos, "Braille character detection using deep neural networks for an educational robot for visually impaired people",

[4] TASLEEM KAUSAR 1 , SAJJAD MANZOOR1, "Deep Learning Strategy for Braille Character Recognition", Digital Object Identifier 10.1109/ACCESS.2021.3138240, VOLUME 9, 2021

[5] Shokat, S., Riaz, R., Rizvi, S.S. et al. Deep learning scheme for character prediction with position-free touch screen-based Braille input method. Hum. Cent. Comput. Inf. Sci. 10, 41 (2020). https://doi.org/10.1186/s13673-020-00246-6