International Journal of Mathematical, Engineering and Management Sciences
Vol. 6, No. 5, 1297-1314, 2021
https://doi.org/10.33889/IJMEMS.2021.6.5.078

Fine-Tuned Pre-Trained Model for Script Recognition

Mamta Bisht
Department of Electronics and Communication Engineering,
Jaypee Institute of Information Technology, Noida, India.
Corresponding author: bishtmamta29@gmail.com

Richa Gupta
Department of Electronics and Communication Engineering,
Jaypee Institute of Information Technology, Noida, India.
E-mail: richa.gupta@jiit.ac.in

(Received on April 11, 2021; Accepted on July 25, 2021)

Abstract
Script recognition is the first necessary preliminary step for text recognition. In the deep learning era, for this task two essential requirements are the availability of a large labeled dataset for training and computational resources to train models. But if we have limitations on these requirements then we need to think of alternative methods. This provides an impetus to explore the field of transfer learning, in which the previously trained model knowledge established in the benchmark dataset can be reused in another smaller dataset for another task, thus saving computational power as it requires to train only less number of parameters from the total parameters in the model. Here we study two pre-trained models and fine-tune them for script classification tasks. Firstly, the VGG-16 pre-trained model is fine-tuned for publically available CVSI-15 and MLe2e datasets for script recognition. Secondly, a well-performed model on Devanagari handwritten characters dataset has been adopted and fine-tuned for the Kaggle Devanagari numeral dataset for numeral recognition. The performance of proposed fine-tune models is related to the nature of the target dataset as similar or dissimilar from the original dataset and it has been analyzed with widely used optimizers.

Keywords- Transfer learning, Fine-tuning, Deep learning, CNN, VGG-16 model, Script classification.

1. Introduction

Script identification in documents and scene images is an essential starting point for text recognition under multi-lingual scenarios. It is basically a classification task and has potential applications in documents and scene understanding (Yuan et al., 2016), mobile phone navigation and mobile product search (He et al., 2012), text detection for videos (Khare et al., 2015), and machine translation (Alabau et al., 2014) etc. This research area consists of lots of challenges such as low image quality, complex background, and different text styles etc., which makes script identification a difficult task in natural scene images (Ghosh et al., 2010). Various scripts have similar formations of characters thus make script identification even more difficult. English, Greek, Russian, Chinese, and Japanese are some examples that have similar shapes of characters.

Recently, convolutional neural network (CNN) proves its excellent performance for the classification tasks. In the network, the earlier CNN layers learn low-level features; the middle CNN layers learn complex features and the CNN layers closer to the output part of the network interprets the learned features into the context of a classification task. But the network requires lots of input data in lakhs or millions for its successful deployment in the real world. To solve this small dataset problem, the model weights established for standard datasets can be reused. Transfer learning often comprises taking the pre-trained weights in the first layers and initializing the last layers randomly and training them for the classification tasks. Thus, in the transfer learning approach, learning or back-propagation occurs only at the last layers initialized with random weights.

This paper focuses on identifying the pre-segmented text in natural scene images and pre-segmented Devanagari numerals using a transfer learning approach with small trainable parameters. Transfer learning is particularly useful with a limited computing resource. A lot of state of the art models take several days and weeks in some cases to train even when trained on highly powerful GPU machines. Thus, in order not to repeat the same process over a long period of time, transfer learning allows us to make use of pre-trained weights as a beginning point. After using transfer learning we can take three advantages, firstly, we can overcome the problem of the limited dataset, secondly, we can avoid the requirement of powerful GPU machines, and thirdly, we can save computation time. Fine-tuning is a methodology for transfer learning and in this paper; it is used on openly available CVSI-15 and MLe2e datasets for the script classification task and also used on Kaggle Devanagari numeral dataset for the numeral recognition task with small trainable parameters. We have selected convolutional neural network model VGG-16 described by Simonyan and Zisserman (2014) and the model discussed by Bisht and Gupta (2020) for fine-tuning. The performance of the proposed fine-tune model is also analyzed with widely used optimizers such as stochastic gradient descent (SGD), root mean square propagation (RMSprop), and adaptive moment estimation (Adam).

The major outcomes of this paper are as: (1) Study of two pre-trained models, VGG-16 and model trained on Devanagari handwritten characters dataset (DHCD). (2) Fine-tune these pre-trained models for another datasets as CVSI, MLe2e and Kaggle Devanagari handwritten numeral dataset. (3) Analyze the performances of described fine-tune models with different optimizers as sgd, RMSprop, and Adam. The paper is presented in the following sections: In Section 2, we discuss the importance of transfer learning and script classification-related work that exists in the literature. In Section 3, we discuss the proposed VGG-16 fine-tune network and fine-tune network trained on Devanagari handwritten characters dataset (DHCD) (Bisht and Gupta, 2020). In Section 4, we analyze the obtained results of the proposed fine-tune networks with different optimizers. In Section 5, we conclude the study.

2. Literature Review

The deep learning model requires large training data for training and at testing time the trained model expects identical feature spaces in test data as it learned during training from the training dataset. But in reality, the test data may not always hold this property and this leads to the requirement to rebuild a new model from scratch. It is sometimes difficult to collect new large training data and rebuild the model. In this situation, the transfer learning method is helpful. Nowadays, transfer learning method has been explored in many research areas like military object recognition (Yang et al., 2019), plant disease identification (Chen et al., 2020), human action recognition (Wang et al., 2017), character classification (Pramanik and Bag, 2020), medical image analysis (Khan et al., 2019) and many more. Today, many popular models exist for image recognition tasks which are trained on benchmark datasets like VGG-16, Inception V3, and Resnet-50 etc. The knowledge of these pre-trained models can be reused for another task.

Earlier script identification work started with texts in documents (Ghosh et al., 2010; Ubul et al., 2017) and videos (Sharma et al., 2013; Sharma et al., 2014) where the method mainly focused on the property of shape and visibility of data. Various hand-crafted features have been extracted in literature and used to train classifiers for the script classification tasks. The commonly used classifier included support vector machines, k-nearest neighbor's algorithm and random forest etc. With the time the convolutional neural network (CNN) popularity increases as it is able to automatically extract effective attributes from the input data which is difficult and time taking process in traditional machine learning approach (Li and Tang, 2015; Li et al., 2015; Li et al., 2017) and it is proved in the ICDAR competition on video script identification (CVSI) in 2015 as all the CNN based methods performed well over hand-crafted features based methods (Gomez et al., 2017).

Mei et al. (2016) focused on the spatial dependencies within text line and bring together a CNN and an RNN (Recurrent Neural Network) into one end-to-end trainable network. To work with random dimensions input images, they also integrated an average pooling structure in their model for the script classification tasks. Shi et al. (2015) adopted multi-stage pooling in their CNN network which has capability to handle exceptionally randomize aspect ratios of scene text images. They further worked on the mid-level representations and combined them into a CNN for script classification (Shi et al., 2016). Gomez and Karatzas (2016) and Bhunia et al. (2019) worked on CNN features and then integrate Naive-Bayes nearest neighbor classifier and attention mechanism respectively in their work. Lu et al. (2019) integrated Local and Global CNN together and used well known ResNet-20 architecture for script identification. An effective method focusing on two key components feature extraction and classification is adopted by Ma et al. (2021), the first component is based on hierarchical feature fusion while the second one is a convolutional classifier. Tounsi et al. (2017) worked on script classification using transfer learning on a small dataset.

As deep learning model requires high computational power and a very large dataset for training and if we have less computational power (or fewer resources) and less dataset for training then the performance of trained model is affected by low performance on real world test data. To overcome this issue we may try to use weights of well-trained model on very large benchmark dataset such as ImageNet which consists millions of images for classification task. This gives the motivation to explore transfer learning area where knowledge of pre-trained model can be used on another dataset for another task and this saves computational power and requirement of very large dataset. In this work, we explore the VGG-16 pre-trained model and use it as a feature extractor after removing its top layer on the script identification dataset. We add two dense layers and an output layer at its top and train these newly added layers only and hold all weights of pre-trained VGG-16 network. We also select a pre-trained model trained on DHCD dataset (Bisht and Gupta, 2020) and check its performance for Kaggle Devanagari handwritten numeral dataset for recognition task. We perform training with SGD, RMSprop, and Adam optimizers and analyzed their performance.

3. Methodology

Transfer learning refers to a neural network model trained on one dataset that can be used for another small dataset by fine-tuning the earlier network. In transfer learning, the training dataset (D_s) and the test dataset (D_t) come from a different distribution i.e. $D_s \neq D_t$. Its aim is to use the knowledge of learning task (T_s) in training dataset to help improve the target learning task (T_t) in the test dataset where $T_s \neq T_t$. The fine-tuning of the model depends on the size and similarity of the new dataset to the original dataset. Here we discuss the two cases of transfer learning when the new dataset is different and similar to the original dataset. If the new dataset is different, we need to fine-tune the network somewhere earlier in the network instead, trains only the classifier part at the top of the network. Here VGG-16, one of the popular pre-trained models is used to classify images of the script datasets, CVSI-15 and MLe2e. If the new dataset is similar to original dataset then we assume higher-level features in the pre-trained network to be appropriate to the new dataset as well and therefore, we train only the classifier part at the top of the network. The description of pre-trained models used are as follows:

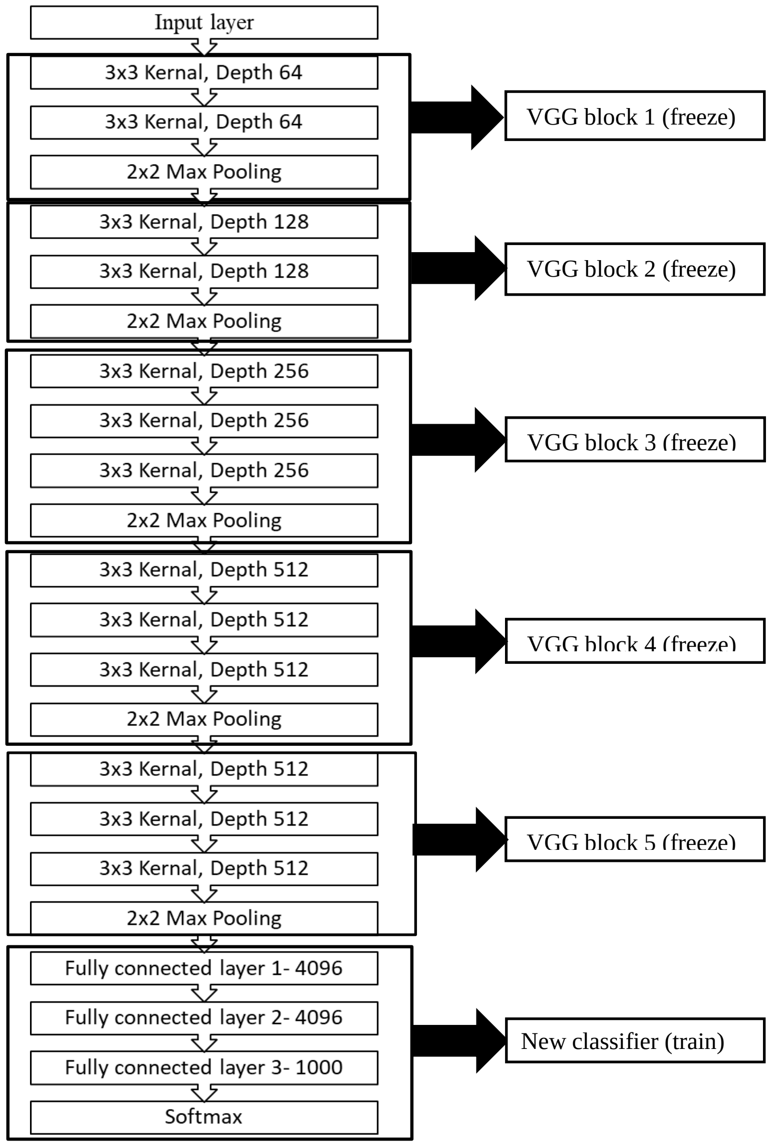

Figure 1. VGG-16 fine tuning.

3.1 Pre-Trained VGG-16 Model

The VGG-16 model, configured to input as 224×224×3 is trained on ImageNet photo classification dataset. The feature extractor part of this model consists of 13 convolutional layers in a group, known as VGG block that includes small filters of size 3×3 followed by a max-pooling layer. It consists of fully connected layers and the output layer in its classifier part. Here, we design fine-tune VGG-16 model. The feature extraction part of this model is loaded from Keras keeping include top as false and then we add two fully connected layers of node 128 on it. Then the output layer is set to the number of unique classes present in datasets as 10 in CVSI-15 dataset and 4 in MLe2e dataset. In chosen fine-tune model only new fully connected and output layers have trainable weights. The training of designed fine-tune model is done for 25 epochs with batch size 64 and learning rate as 0.001. The proposed fine tune model is shown in Figure 1.

3.2 Pre-Trained Model on DHCH

The model is configured to input as 32×32×3 is trained on large dataset DHCD of size 78,200 for training and 13,800 for testing. It contains 36 Devanagari handwritten character classes and 10 Devanagari handwritten digit classes. The model consists of four convolutional layers mixed with nonlinearity (RELU) and max pooling layers with kernel size as 3x3 and stride value as 1. The model also added dropout of 0.2 to avoid overfitting condition. Then flattening is done and further the model uses two dense layers of shape 128, 64. The last dense layer is connected to the output layer of size 46 with softmax activation function. We fine-tune the model for recognition of Kaggle Devanagari numeral dataset (Pant et al., 2012). Since the new dataset is similar to the original dataset i.e. DHCD, we assume that the extracted features from the earlier model are relevant to the new dataset and therefore we only fine-tune the classifier part and hence the model consists of very small trainable parameters. The model is trained for 50 epochs with batch size 32 and learning rate as 0.001 respectively.

The above described pre-trained models consist of a series of layers as convolutional, pooling, flattening, and fully connected layers. The input samples are passed through these layers and the generated output is used to categorize samples. The main parts of the convolutional neural network are as follows:

Convolutional Layer – In the convolutional layer, the convolutional product takes place between the input sample and the kernel, which passes over the input sample. Mathematically, for a given input image 'Img' with dimension as (n_H, n_W, n_C) and kernel 'K', we have convolution as:

$$conv(Img, K)_{(x,y)} = \sum_{i=1}^{n_H} \sum_{j=1}^{n_w} \sum_{k=1}^{n_c} K_{i,j,k} \; Img_{x+i,y+j,k}$$

where n_H is the height, n_w is the width and n_C is the number of channels in the input sample.

Activation Function - It is commonly used after convolutional layers to add element-wise non-linearity over the input sample. Out of the available activation functions, sigmoid, tanh, and rectified linear unit (ReLU), the ReLU activation function is most popular and defined as:

$$RELU(x) = 0 \; if \; x \leq 0 \; and \; RELU(x) = x \; if \; x \geq 0.$$

Pooling - It is a stage of down-sampling of the image's features through summing up the information. The most common pooling function is the max-pooling. If $\text{Window}(n, n)$ is a window function to the patch of the convolution layer, and a_j is the maximum value in the neighborhood then it is defined as:

$$a_j = \max_{N\times N}(a_i^{n\times n} Window(n,n)).$$

For a kernel 'K' of dimension $n \times n$ with stride '$\boldsymbol{s}$' which is a value of moving steps in the convolutional product, the dimension of this layer is defines as:

$$dimension\left(pooling\left(Img_{(n_H,n_W,n_C)}\right)\right) = \left(\frac{n_H + 2p - n}{s} + 1\,, \frac{n_W + 2p - n}{s} + 1, n_c\right).$$

Where 'p' is the number of elements added on every side of input sample to note the edge information and called padding.

Fully Connected Layer – The output of the final convolutional neural network is connected to the one or more fully connected layers where every k^{th} node in the previous layer is connected to every j^{th} node in the next layer as:

$$y_j = \sum_k w_{jk} x_k + b_j$$

where, w$_{jk}$ is the weight value between the k^{th} node of the previous layer and the j^{th} node of the current layer, b_j is the bias value of the j^{th} node of the current layer.

4. Analysis

Firstly, we have performed fine-tuning with VGG-16 on CVSI -15 and MLe2e dataset with different optimizers for script classification. The CVSI-15 (Sharma et al., 2015) contain 6,412, 1,069 and 3,207 scene text images for training, validation and testing respectively of ten script classes. The MLe2e dataset (Gomez et al., 2017) is comparatively small with 1,178 and 643 word images for training and testing respectively of four different scripts.

We used the feature extraction part of the pre-trained VGG-16 model and add a two new dense layers and output layer in classifier part to the model for the script identification task. We freeze all the weights of the convolutional layers in pre-trained VGG-16 during training as shown in Figure 1 and only train newly added layers in classifier part. We add two fully connected layers of node 128 in fine-tune model. Then we add an output layer of nodes equal to a number of the unique labels in each dataset. We check the performance of this chosen fine-tune network on different optimizer parameters which helps in updating weights and minimizing the loss function. We also add image augmentation in the fine-tune model.

The experimental results are tabulated in Table 1. It is observed that the VGG-16 fine-tune model performs well using Adam optimizer for CVSI-15 dataset and gives approximately 89.64% test accuracy. For MLe2e, fine-tune model performs well using SGD and RMSprop optimizers with best test accuracy approximately as 80.68%.

Secondly, we have performed fine-tuning with the model described in section 3.2 on Kaggle Devanagari numeral dataset (Pant et al., 2012) with different optimizers for numeral classification. We freeze all the weights of the convolutional layers in a pre-trained model during training and only train newly added classifier part with nodes equal to a number of the unique labels in the Devanagari numerals i.e. 10. The experimental results are tabulated in Table 2. It is observed that the DHCD fine-tune model performs well using SGD, RMSprop, and Adam optimizers for Kaggle Devanagari numeral dataset.

Table 1. VGG-16 fine-tuning on CVSI-15 and MLe2e dataset.

Dataset (Training/ validation/ test) Input: (224*224*3)	**Optimizer**	**Accuracy (%)**		
		Training	Val	Test
CVSI-15 (6412 / 1069/3207) Total parameters: 17,943,882 Trainable parameters: 3,229,194	SGD(lr=0.001, momentum=0.9)	88.83	83.25	82.28
	RMSprop(lr=0.001)	91.96	88.58	87.99
	Adam(lr=0.001)	96.29	90.45	89.64
MLe2e (1174/-/642) Total parameters: 17,943,108 Trainable parameters: 3,228,420	SGD(lr=0.001, momentum=0.9)	83.60		80.68
	RMSprop(lr=0.001)	84.68	-	80.21
	Adam(lr=0.001)	41.17	-	24.92

Table 2. Fine tuning on Kaggle Devanagari numeral dataset.

Dataset (Training/ validation/ test) Input: (32*32*3)	**Optimizer**	**Accuracy (%)**		
		Training	Val	Test
Kaggle Devanagari numeral dataset (2304/-/576) Total parameters: 307,082 Trainable parameters: 650	SGD (lr=0.001, momentum=0.9)	96.27	-	96.52
	RMSprop (lr=0.001)	97.18	-	96.70
	Adam (lr=0.001)	96.48	-	98.09

During training, we use optimizer to find the optimized weights and to minimize the loss function which tells the optimizer if it is moving in the correct way to reach the bottom of the valley, the global minimum. These multiple iterations on complete training dataset update the weights and bias known as Gradient descent. Since it works on the whole training dataset, it becomes slow and computationally expensive if the training dataset is very big. One of the advance selections for the Gradient descent method is SGD which is used for only partial data to train every time, called mini-batch size. But it requires forward and backward propagation for every entry and makes the process noisy and slow too. If we add momentum in SGD, it supports gradients vectors to speed up in the correct way, thus leading to faster convergence. This algorithm uses exponentially weighted averages which deal with sequences of numbers to compute gradient and use this gradient to update parameters. For a new sequence V, exponentially weighted averages define as:

$$V_t = \beta V_{(t-1)} + (1-\beta) S_t .$$

where β is a hyper-parameter, β € [0,1], for this study we have selected $\beta = 0.9$ and S_t is a noisy sequence at instant t.

The second optimizer used is Root Mean Square Propagation and abbreviated as RMSprop optimizer. It restricts the oscillations in the vertical direction and it takes bigger step-size in the horizontal direction for converging faster. The third optimizer used is Adaptive Moment Estimation and abbreviated as Adam optimizer. It combines the properties of SGD with momentum and RMSprop.

The performance of first proposed VGG-16 fine-tune model for CVSI-15 and MLe2e dataset with SGD with momentum is shown in Figure 2 and Figure 3, with RMSprop it is shown in Figure 4 and Figure 5, and with Adam it is shown in Figure 6 and Figure 7 with a learning rate of 0.001. These figures show the loss and accuracy curves for 25 epochs. Figure 8 and Figure 9 present their confusion matrixes for dataset used by proposed VGG-16 fine-tune model.

The performance of second proposed fine-tune model on Kaggle Devanagari numeral dataset with SGD, RMSprop, and Adam optimizers is presented in Figure 10, Figure 11 and Figure 12. These figures show the loss and accuracy curves for 50 epochs. Figure 13 present a confusion matrix with Adam optimizer.

It is observed that if the dataset used in transfer learning is different from the original dataset then it requires fine tuning with some layers prior to classification. Here, firstly the VGG-16 pre-trained model is used as a feature extractor and then fine-tuned by adding new layers to the model. These freshly added new layers were trained for publically available CVSI-15 and MLe2e datasets for script recognition. This proposed fine-tune model has a total 17,943,882 parameters and 3,229,194 trainable parameters for CVSI-15 dataset. Similarly, it consists of total 17,943,108 parameters and 3,228,420 trainable parameters for the MLe2e dataset as presented in Table 1. A well-performed model on Devanagari handwritten characters dataset has also been adopted and fine-tuned for Kaggle Devanagari numeral dataset for recognition. This proposed fine-tune model has a total 307,082 parameters and 650 trainable parameters as presented in Table 2. This proposed fine-tune model for the Kaggle Devanagari numeral dataset performs very well with very small trainable parameters. All experiments are performed on a computer with i5 - 9300H CPU (2.40 GHz), 8 GB RAM and NVIDIA GeFORCE GTX 1650.

Our analysis of the results on CVSI-15 and MLe2e datasets using the proposed fine-tune VGG-16 model relates to its distinct nature from the original dataset. On the other hand, the results on the Kaggle Devanagari numeral dataset using proposed fine-tune DHCD model relates to its similar nature from the original dataset. Few samples of above datasets are shown in Figure 14, Figure 15 and Figure 16. In CVSI-15 dataset many samples are identical in font style and background because they are cropped word images from long sentences but in MLe2e dataset, samples consist of large variation in font style and background.

Although the highest accuracy of 98.91% is achieved by Google using deep convolutional network with data-augmentation (Sharma et al., 2015) on CVSI-15, our proposed fine-tuned pre-trained VGG-16 model is a shallower network that involves a faster training procedure. The highest accuracy of 97.20% is achieved by Ma et al. (2021) using Residual attention-based model but our proposed fine-tuned pre-trained VGG-16 model has limited performance due to huge variability in dataset and distinct nature from the original dataset. The proposed fine-tuned pre-trained DHCD model performs well on Kaggle Devanagari numeral dataset because of huge similarity in source and target dataset in transfer learning. The comparison of the proposed fine-tune models with few other methods is presented in Table 3.

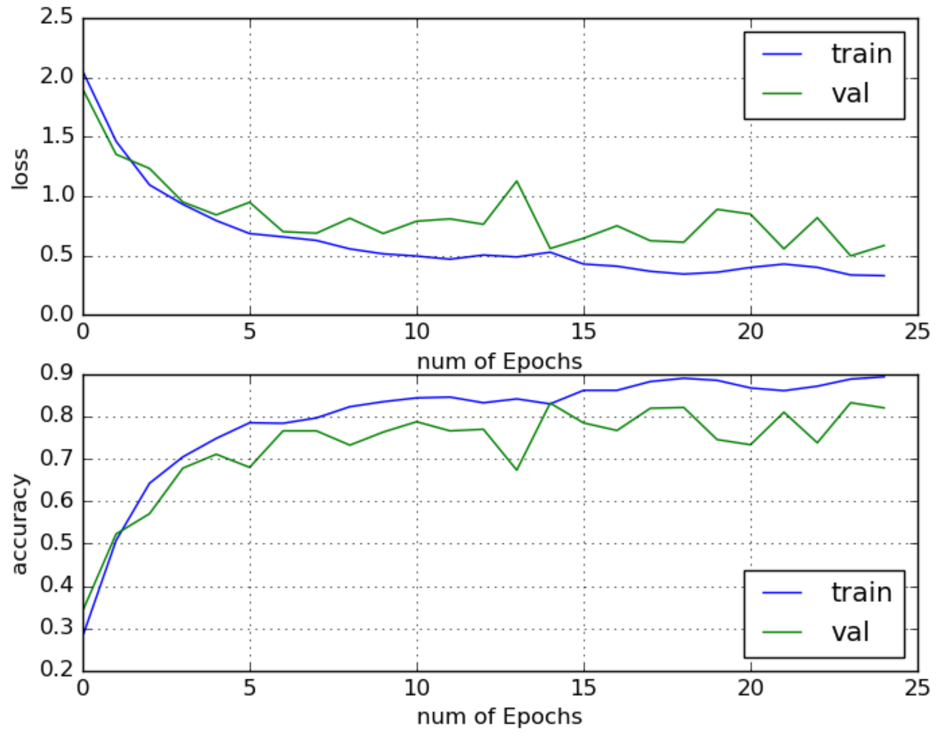

Figure 2. VGG-16 fine tune model on CVSI-15 using SGD optimizer.

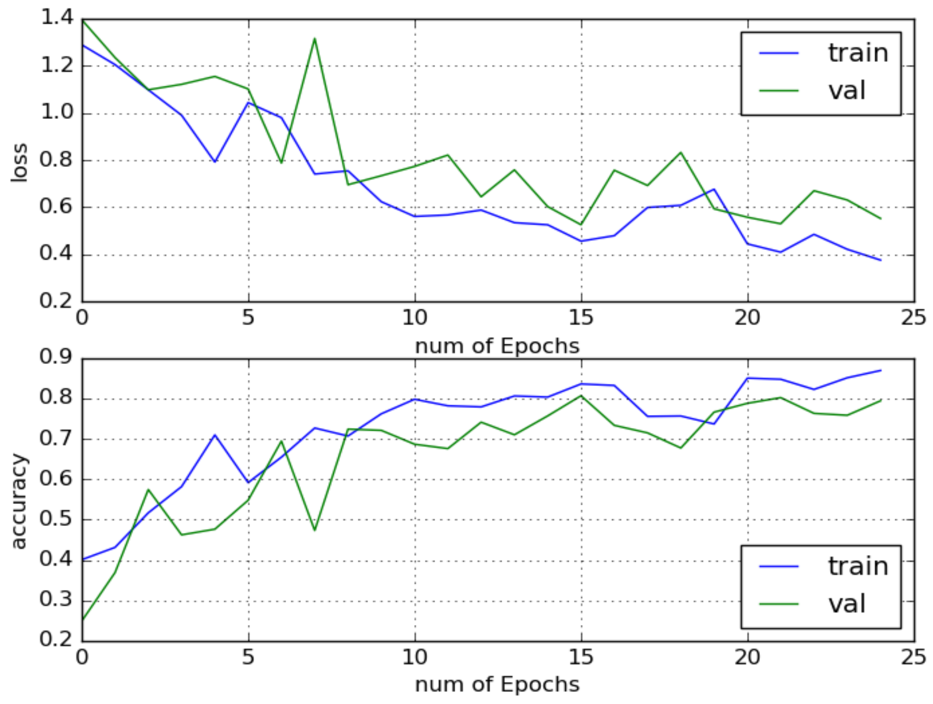

Figure 3. VGG-16 fine tune model on MLe2e using SGD optimizer.

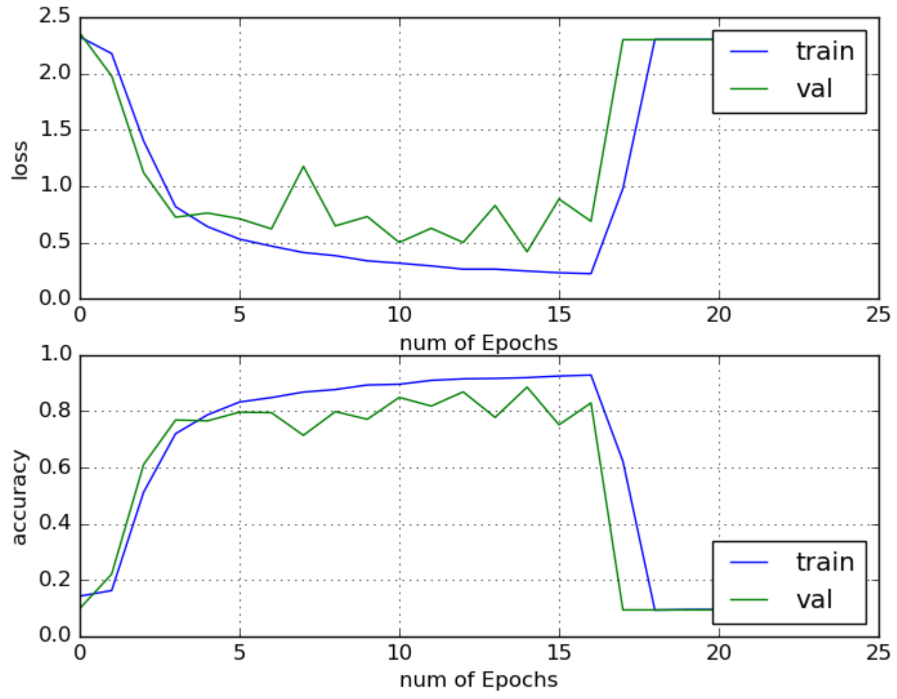

Figure 4. VGG-16 fine tune model on CVSI-15 using RMSprop optimizer.

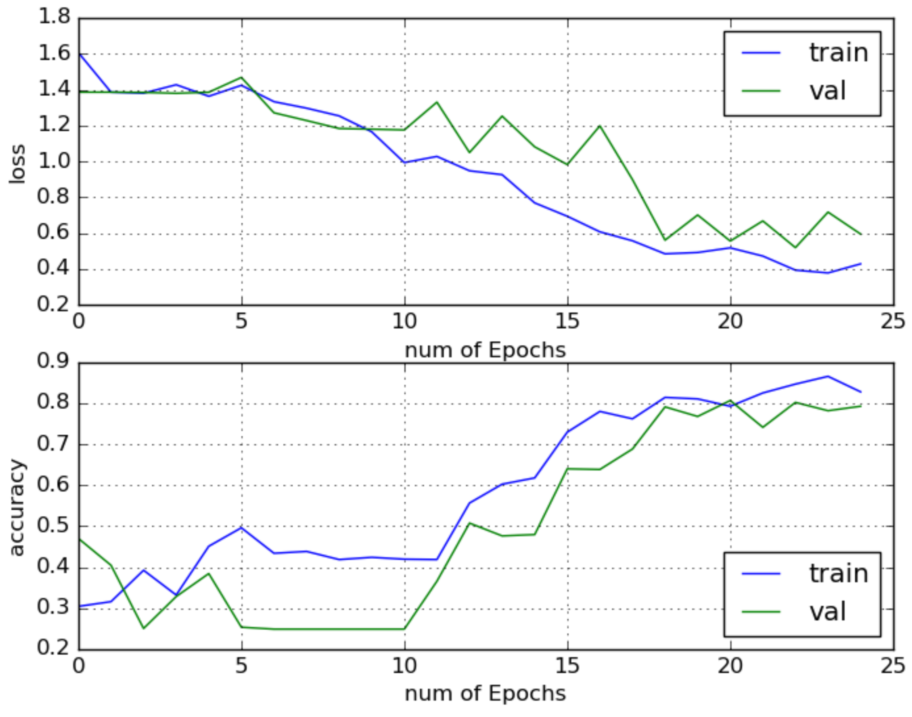

Figure 5. VGG-16 fine tune model on MLe2e using RMSprop optimizer.

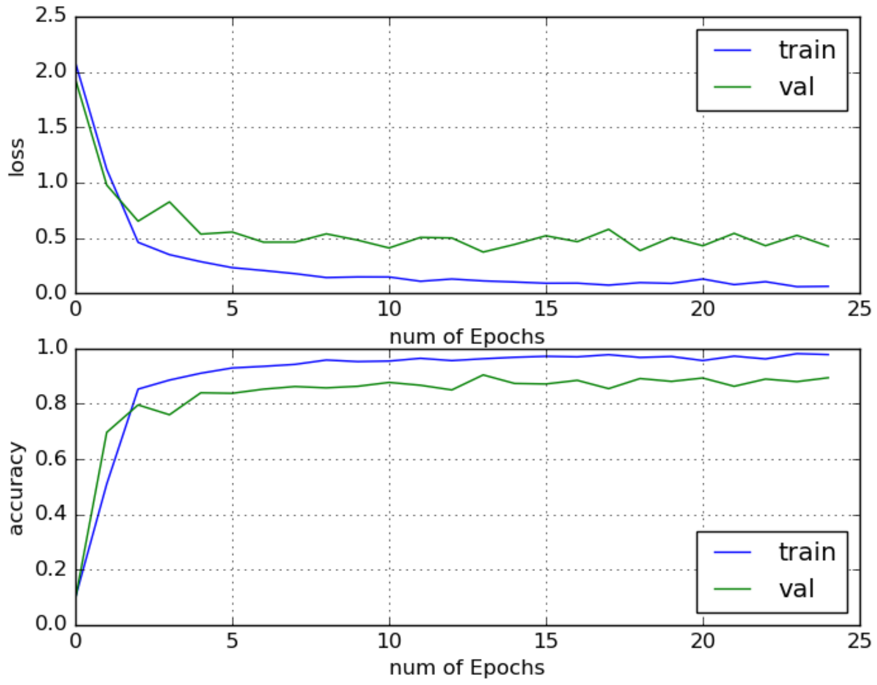

Figure 6. VGG-16 fine tune model on CVSI-15 using Adam optimizer.

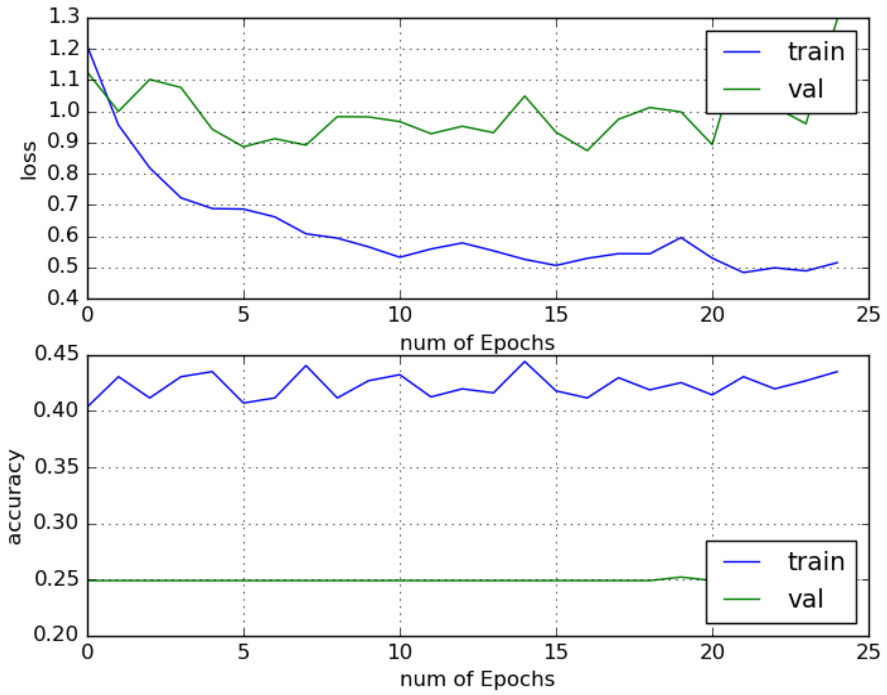

Figure 7. VGG-16 fine tune model on MLe2e using Adam optimizer.

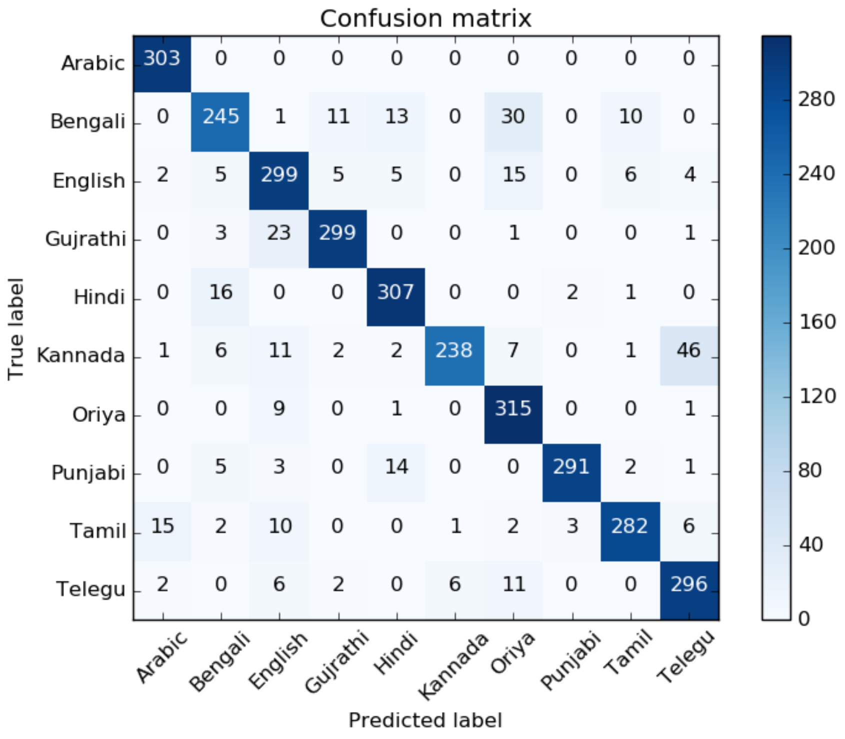

Figure 8. Confusion matrix on CVSI-15 dataset using VGG-16 fine-tune model with Adam optimizer.

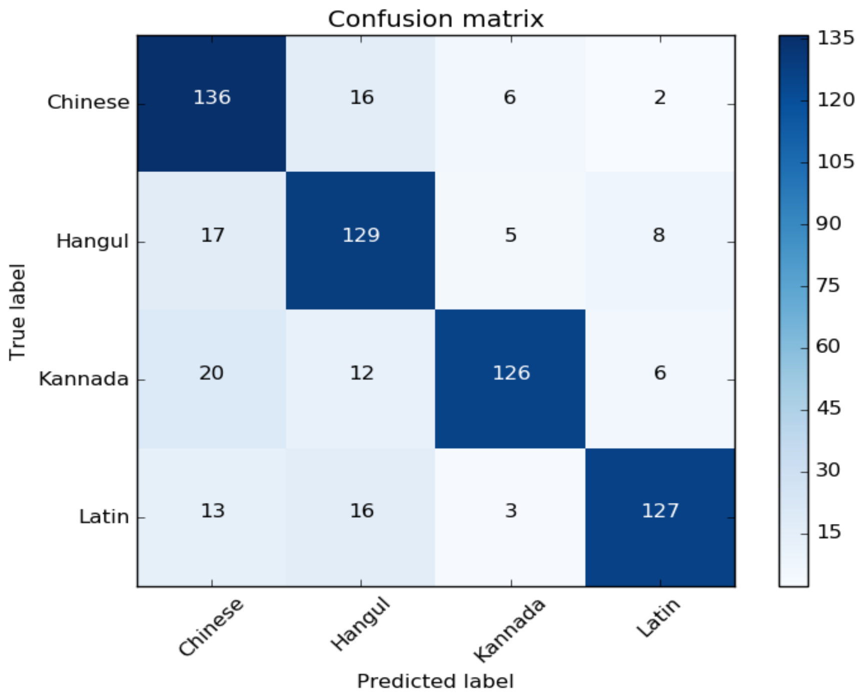

Figure 9. Confusion matrix on MLe2e dataset using VGG-16 fine-tune model with SGD with momentum optimizer.

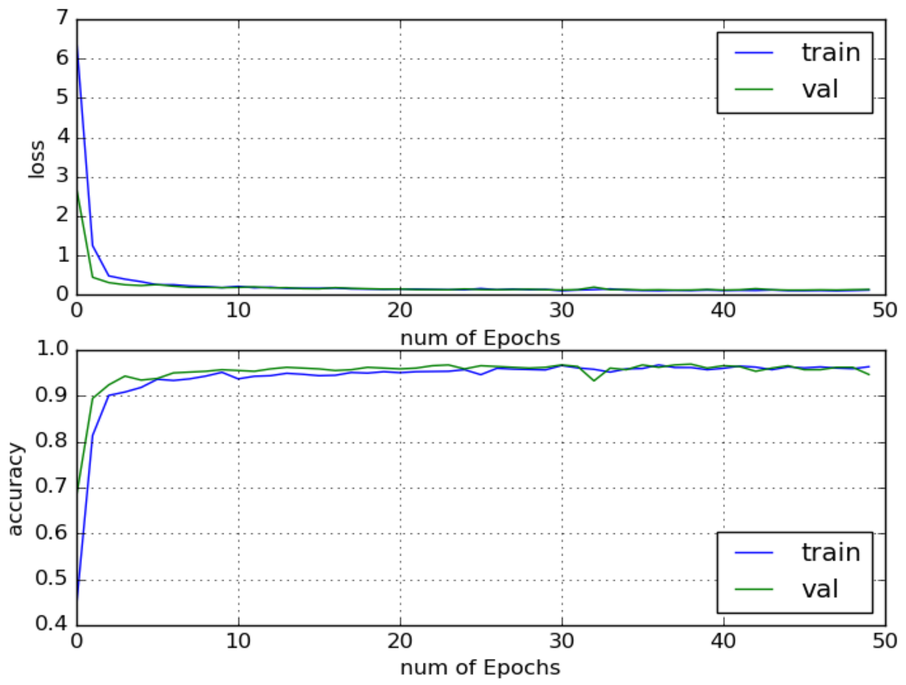

Figure 10. Fine tune DHCD model on Kaggle Devanagari numeral dataset using SGD optimizer.

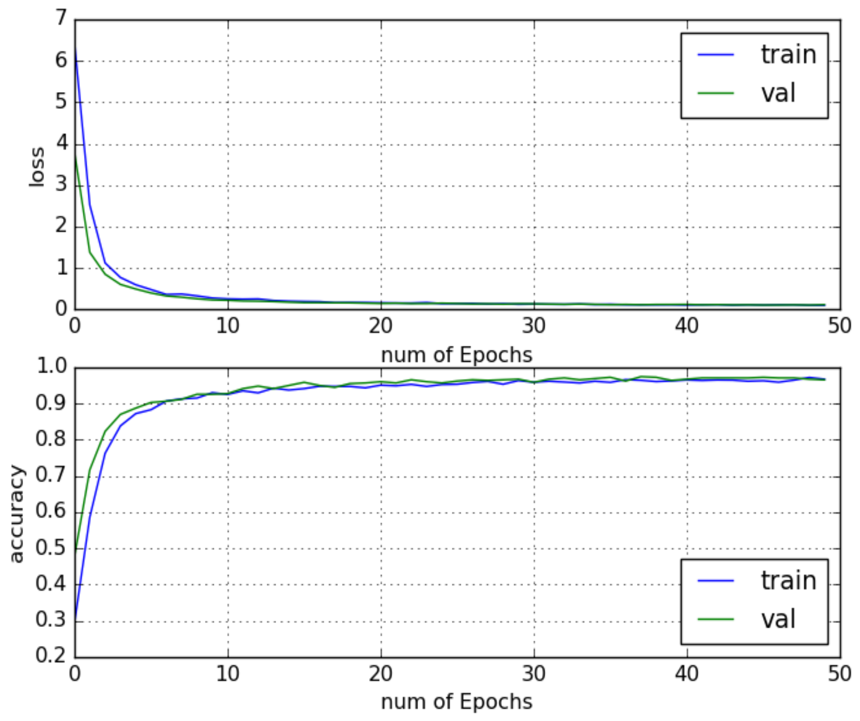

Figure 11. Fine tune DHCD model on Kaggle Devanagari numeral dataset using RMSprop optimizer.

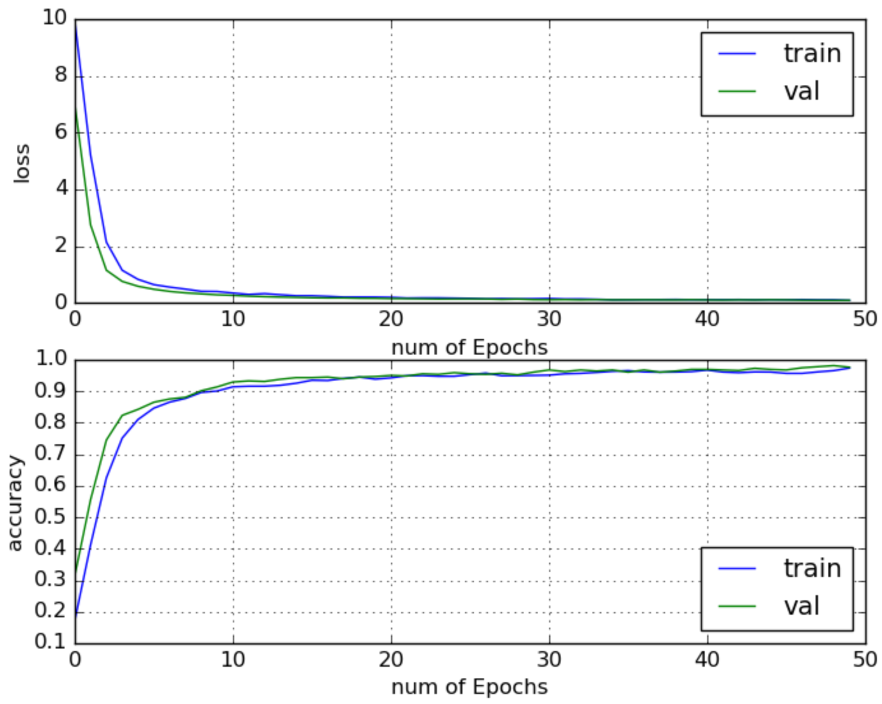

Figure 12. Fine tune DHCD model on Kaggle Devanagari numeral dataset using Adam optimizer.

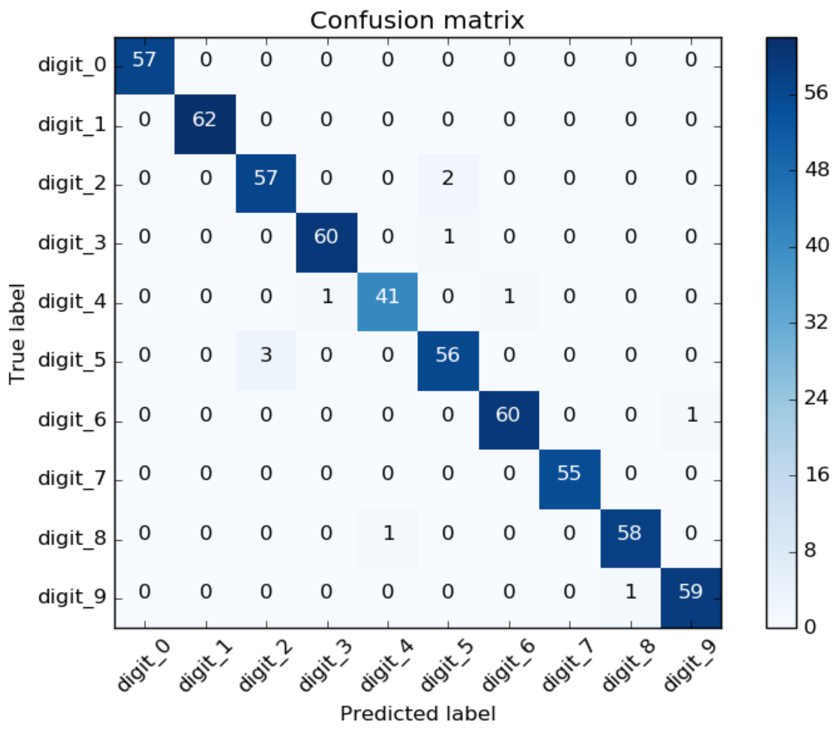

Figure 13. Confusion matrix on Kaggle Devanagari numeral dataset using fine-tune DHCD model with Adam optimizer.

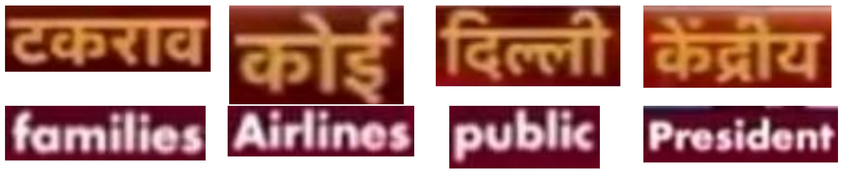

Figure 14. Example of samples shares exactly the same font and background in CVSI-15 dataset.

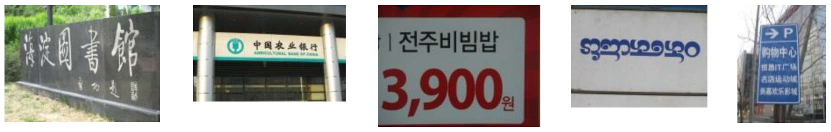

Figure.15. Variability in MLe2e dataset.

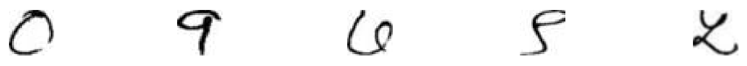

Figure 16. Samples of Kaggle Devanagari numeral dataset.

Table 3. Overall classification performance comparison with some other methods in three different datasets: CVSI-15, MLe2e, and Kaggle Devanagari numeral dataset.

Method	**CVSI-15**	**MLe2e**	**Kaggle Devanagari numeral dataset**
C-DAC (Sharma et al., 2015)	84.66	-	-
CUK (Sharma et al., 2015)	74.06	-	-
Baseline SIFT + Bag of Words + SVM (Gomez et al., 2017)	84.38	**86.45**	-
SRS-LBP + KNN (Gomez et al., 2017)	-	82.71	-
Multilayer Perceptron (MLP) (Pant et al., 2012)	-	-	87.50
Radial Basis Function (RBF) (Pant et al., 2012)	-	-	94.44
Proposed fine-tuned pre-trained VGG-16 model with Adam(lr=0.001)	**89.64**	-	-
Proposed fine-tuned pre-trained VGG-16 model with SGD (lr=0.001, momentum=0.9)	-	80.68	-
Proposed fine-tuned pre-trained DHCD model with Adam (lr=0.001)	-	-	**98.09**

5. Conclusions and Future Work

In this article we worked on a script recognition task using transfer learning where the feature extraction part is chosen from an existing pre-trained model. Here, two pre-trained models are taken, first pre-trained model is VGG-16 trained on ImageNet dataset and second pre-trained model is trained on DHCD. First model is fine-tuned for CVSI-15 and MLe2e datasets which is different from the original dataset for script classification tasks. Second pre-trained model is fine tuned for the Kaggle Devanagari numeral dataset which is similar to the original dataset for numeral classification. Both fine tune models are tested on SGD, RMSprop and Adam optimizers and their performance has been analyzed. The advantage of this transfer learning is that we can take pre-trained model weights, trained on benchmark datasets and only train the last added few layers for fine tuning. This saves time and reduces the requirement of a highly configured machine for training. This also overcomes the need for a very large dataset and is useful in solving real-world problems. In future more fine-tuning work can be explored for better performance of described pre-trained models for script classification tasks.

Conflict of Interest
The authors declare that there is no conflict of interest regarding the publication of this work.

Acknowledgements
The authors are grateful to the editor and reviewers for their helpful suggestions.

References

Alabau, V., Sanchis, A., & Casacuberta, F. (2014). Improving on-line handwritten recognition in interactive machine translation. *Pattern Recognition*, *47*(3), 1217–1228. Doi: 10.1016/j.patcog.2013.09.035.

Bhunia, A.K., Konwer, A., Bhunia, A.K., Bhowmick, A., Roy, P.P., & Pal, U. (2019). Script identification in natural scene image and video frames using an attention based convolutional-LSTM network. *Pattern Recognition*, *85*, 172–184. Doi: 10.1016/j.patcog.2018.07.034.

Bisht, M., & Gupta, R. (2020). Multiclass recognition of offline handwritten Devanagari characters using CNN. *International Journal of Mathematical, Engineering and Management Sciences*, *5*(6), 1429–1439.

Chen, J., Chen, J., Zhang, D., Sun, Y., & Nanehkaran, Y.A. (2020). Using deep transfer learning for image-based plant disease identification. *Computers and Electronics in Agriculture*, *173*, 105393.

Ghosh, D., Dube, T., & Shivaprasad, A. (2010). Script recognition—a review. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *32*(12), 2142–2161. Doi:10.1109/TPAMI.2010.30.

Gomez, L., & Karatzas, D. (2016). A fine-grained approach to scene text script identification. In *2016 12th IAPR Workshop on Document Analysis Systems* (pp. 192–197). IEEE. Santorini, Greece.

Gomez, L., Nicolaou, A., & Karatzas, D. (2017). Improving patch-based scene text script identification with ensembles of conjoined networks. *Pattern Recognition*, *67*(1), 85–96.

He, J., Feng, J., Liu, X., Cheng, T., Lin, T.H., Chung, H., & Chang, S.F. (2012). Mobile product search with bag of hash bits and boundary reranking. In *2012 IEEE Conference on Computer Vision and Pattern Recognition* (pp. 3005–3012). IEEE. Providence, Rhode Island, USA. Doi: 10.1109/CVPR.2012.6248030.

Khan, S., Islam, N., Jan, Z., Ud Din, I., & Rodrigues, J.J.P.C. (2019). A novel deep learning based framework for the detection and classification of breast cancer using transfer learning. *Pattern Recognition Letters*, *125*, 1–6. Doi: 10.1016/j.patrec.2019.03.022.

Khare, V., Shivakumara, P., & Raveendran, P. (2015). A new histogram oriented moments descriptor for multi-oriented moving text detection in video. *Expert Systems with Applications*, *42*(21), 7627–7640.

Li, Z., & Tang, J. (2015). Unsupervised feature selection via nonnegative spectral analysis and redundancy control. *IEEE Transactions on Image Processing*, *24*(12), 5343–5355. Doi: 10.1109/TIP.2015.2479560.

Li, Z., Liu, J., Tang, J., & Lu, H. (2015). Robust structured subspace learning for data representation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *37*(10), 2085–2098.

Li, Z., Tang, J., & He, X. (2017). Robust structured nonnegative matrix factorization for image representation. *IEEE Transactions on Neural Networks and Learning Systems*, *29*(5), 1947–1960.

Lu, L., Yi, Y., Huang, F., Wang, K., & Wang, Q. (2019). Integrating local CNN and global CNN for script identification in natural scene images. *IEEE Access*, *7*, 52669–52679. Doi: 10.1109/ACCESS.2019.2911964.

Ma, M., Wang, Q.F., Huang, S., Huang, S., Goulermas, Y., & Huang, K. (2021). Residual attention-based multi-scale script identification in scene text images. *Neurocomputing*, *421*, 222–233.

Mei, J., Dai, L., Shi, B., & Bai, X. (2016). Scene text script identification with convolutional recurrent neural networks. In *2016 23rd International Conference on Pattern Recognition* (pp. 4053–4058). Cancun, Mexico. Doi: 10.1109/ICPR.2016.7900268.

Pant, A.K., Panday, S.P., & Joshi, S.R. (2012, November). Off-line Nepali handwritten character recognition using multilayer perceptron and radial basis function neural networks. In *2012 Third Asian Himalayas International Conference on Internet* (pp. 1-5). IEEE. Kathmundu, Nepal.

Pramanik, R., & Bag, S. (2020). Segmentation-based recognition system for handwritten Bangla and Devanagari words using conventional classification and transfer learning. *IET Image Processing*, *14*(5), 959–972. DOI:10.1049/iet-ipr.2019.0208.

Sharma, N., Chanda, S., Pal, U., & Blumenstein, M. (2013). Word-wise script identification from video frames. In *2013 12th International Conference on Document Analysis and Recognition* (pp. 867–871). IEEE. Washington, DC, USA. Doi: 10.1109/ICDAR.2013.177.

Sharma, N., Mandal, R., Sharma, R., Pal, U., & Blumenstein, M. (2015). ICDAR2015 competition on video script identification (CVSI 2015). In *2015 13th International Conference on Document Analysis and Recognition* (pp. 1196–1200). IEEE. Tunis, Tunisia. Doi: 10.1109/ICDAR.2015.7333950.

Sharma, N., Pal, U., & Blumenstein, M. (2014). A study on word-level multi-script identification from video frames. In *2014 International Joint Conference on Neural Networks* (pp. 1827–1833). IEEE. Beijing, China. Doi: 10.1109/IJCNN.2014.6889906.

Shi, B., Bai, X., & Yao, C. (2016). Script identification in the wild via discriminative convolutional neural network. *Pattern Recognition*, *52*, 448–458. Doi: 10.1016/j.patcog.2015.11.005.

Shi, B., Yao, C., Zhang, C., Guo, X., Huang, F., & Bai, X. (2015). Automatic script identification in the wild. In *2015 13th International Conference on Document Analysis and Recognition* (pp. 531–535). IEEE. Tunis, Tunisia. Doi: 10.1109/ICDAR.2015.7333818.

Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *ArXiv Preprint ArXiv:1409.1556*.

Tounsi, M., Moalla, I., Lebourgeois, F., & Alimi, A.M. (2017). CNN based transfer learning for scene script identification. In *International Conference on Neural Information Processing* (pp. 702–711), Springer, Cham. Guangzhou, China. https://doi.org/10.1007/978-3-319-70136-3_74.

Ubul, K., Tursun, G., Aysa, A., Impedovo, D., Pirlo, G., & Yibulayin, T. (2017). Script identification of multi-script documents: a survey. *IEEE Access*, *5*, 6546–6559. Doi: 10.1109/ACCESS.2017.2689159.

Wang, T., Chen, Y., Zhang, M., Chen, J., & Snoussi, H. (2017). Internal transfer learning for improving performance in human action recognition for small datasets. *IEEE Access*, *5*, 17627–17633.

Yang, Z., Yu, W., Liang, P., Guo, H., Xia, L., Zhang, F., Ma, Y., & Ma, J. (2019). Deep transfer learning for military object recognition under small training set condition. *Neural Computing and Applications*, *31*(10), 6469–6478. Doi: 10.1007/s00521-018-3468-3.

Yuan, Z., Wang, H., Wang, L., Lu, T., Palaiahnakote, S., & Tan, C.L. (2016). Modeling spatial layout for scene image understanding via a novel multiscale sum-product network. *Expert Systems with Applications*, *63*, 231–240. Doi: 10.1016/j.eswa.2016.07.015.