Font Recognition System Using DeepFont Architecture

Bobby Pehtrus and Caecilia C. Lestari

***Abstract*—Typography is an important aspect in visual designs, so that font recognition on an image or photo has been on demand among designers. But, font recognition without any help of technology or expert is proven hard. A survey conducted on Visual Communication Design students in Universitas Ciputra suggest that font recognition application is still on high demand. DeepFont is known as a leading deep learning architecture for font recognition. Finally, the purpose of this research is to implement DeepFont architecture on font recognition system. Experiment to determine unknown parameters on the DeepFont architectuce based on the output shape produce training parameters (3,3) kernel size, 10 epochs with a mixture of same and valid padding. The model shows an accuracy rate of 86% on training set using Tensorflow and Keras. The test results are carried out by generation of three types of datasets namely dataset with the same augmentation of training set, partial augmentation and no augmentation followed by testing each dataset with accuracy, precision and recall metrics using confusion matrix. The datasets have an average accuracy of 78.6%, an average precision of 80.7% and a recall of 78.5%. The font recognition application was successfully tested with Alpha Test on 5 test scenarios and 13 test cases. Based on the test results, it can be concluded that the resulting application can recognize fonts well.**

***Index Terms*—autoencoder, convolution neural network, DeepFont, deep learning, font recognition.**

I. Introduction

Typography is a work of art and the process of organizing a visual form (typeface) to provide a better appearance [1], [2]. Typography is an important factor in visual design, both print and digital. Typography can make it easier for readers to understand or interpret written messages through typeface or font [2].

A designer has certainly met with "fonts in the wild" and wants to find out the type of the fonts. With so many font choices to choose from, the error-prone process of recognizing fonts from text images requires special precision and expertise that is difficult even by font experts [3], [4]. To strengthen the validation, the researchers conducted a survey of 38 students of the Visual Communication Design Program at Universitas Ciputra with graphic design specialization. Survey results show that font selection in graphic design process is important (4.76). In addition, the process of recognizing fonts from an image also considered difficult without any help from experts (3,24). The survey results also state that internet search is the main method to recognize font types (68%) and shows that technological intervention in font recognition is quite helpful for designers in recognizing fonts (4.0). However, the existence of applications to recognize fonts directly is still a high demand (4.79). Therefore, the existence of an application to recognize fonts will be very helpful for designers for their visual design needs.

Research on font recognition has been carried out on many scripts and languages. Machine learning techniques that have been used are Local Feature Embedding and NCM, Gabor Filter [5], [6] and Support Vector Machine [5], [7]. In addition, some websites such as Identifont, MyFonts, WhatTheFont and Fontspring have introduced font recognition based on similarity, but are very dependent on human intervention in *preprocessing* images and do not have satisfactory accuracy [3], [4]. This is due to the shortcomings of machine learning methods, which is to manually determine feature points thereby reducing flexibility in the real environment [8]. On the contrary, deep learning methods can provide such flexibility with domain adaptation techniques such as the use of autoencoders and transfer learning [9] because of their non- *task specific* nature [10].

One of the deep learning implementations in font recognition for Latin alphabet is the Convolution Neural Network (CNN) architecture "DeepFont" which was introduced by [4]. The advantage of this method is it has been complemented by domain adaptation to minimize the domain mismatch problem which causes a decrease in accuracy on the machine learning based font recognition system that has been mentioned by [3].

The operating system that is often used by students to do graphic design at Universitas Ciputra is Windows (68%). Thus, the application operating system demand also leads to the same operating system is Windows (63%).

With 59% penetration of internet users in the world and 64% penetration of internet users in Indonesia [11], websites become one of the popular technology bases. Even so, the application in this study is not web-based because font recognition model can take a large amount of memory. The DeepFont model requires a storage capacity of 700MB, too large to be loaded or downloaded by users [4]. To load the model into RAM for each user will require a large fee. RAM in cloud services is also limited as in the Hostinger provider only 1 GB of RAM for the Enterprise package and Dewaweb provides 1 GB of RAM for the Guardian package so that to load various instances of the deep learning model will consume RAM memory greatly.

To meet the demands of graphic designers, this study will use DeepFont architecture as a font recognition model. A font recognition application will be made on the Windows operating system (63 %). Testing of the model is done using

Manuscript received 1 June, 2020.

The authors are with Department of Informatics, Faculty of Information Technology, Universitas Ciputra, Surabaya 60219, Indonesia (e-mail: bpehtrus@student.ciputra.ac.id, caecilia.citra@ciputra.ac.id).

accuracy, precision and recall metrics with confusion matrix.

II. Literature Review

A. Font Recognition

Fonts can be recognized by typeface, weight, font size and slope (italic) [5], [12]. In this study, the term font is used as a variant of a typeface (e.g. Helvetica is a typeface while Helvetica Italic is font name) [13].

Various methods have been introduced such as Local Feature Embedding to draw features and Nearest Class Mean (NCM) as a classification method [3], Transfer Learning for mandarin characters [14], Gabor Filters [5], [6] and Deep Learning [4], [15].

Deep Learning has been implemented in font recognition in many ways. [4] continued the research of [3] by implementing CNN with decomposition approach. To patch the domain mismatch weaknesses from the method of [3], DeepFont used domain adaptation techniques with Stacked Convolutional Auto-Encoder (SCAE) resulting in more than 80% accuracy in its Top-5. CNN also used to recognize Latin manuscript fonts from the CLaMM dataset (ICFHR2016) and Arabic fonts from the KAFD dataset with AlexNet and ResNet architecture [15]. The CNN was assisted by data augmentation and got an accuracy of 98.8% at the row level.

B. Deep Learning

The taxonomy of artificial intelligence according to [10] can be seen in Fig. 1. Deep Learning is a subfield of machine learning that focuses on the use of multiple hidden layers in a neural network. Deep Learning can yield faster results than machine learning because deep learning can take important features by itself without the need for data experts [8] it can be said that deep learning is not task specific [10].

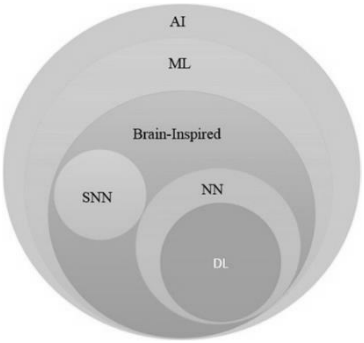


Fig. 1. AI's taxonomy of science. AI: Artificial Intelligence; ML: Machine Learning; NN: Neural Network; DL: Deep Learning; SNN: Spiking Neural Network [10].

The approach of deep learning can be categorized as follows: Supervised, unsupervised and Reinforcement Learning which is often referred to as semi-supervised learning [10]. Supervised methods that are often used are Deep Neural Network, Recurrent Neural Network, Long Short-Term Memory (LSTM) and Convolution Neural Network. The unsupervised method underwent several developments, for example there are Autoencoders, Restricted Boltzmann Machines (RBM) and most recently Generative Adversarial Network (GAN). The supervised and unsupervised methods used in this study are Convolution Neural Networks and Autoencoder which are explained in the next section.

C. Convolution Neural Network

CNN was inspired by the structure of the primate visual cortex as explained by Hubel and Wiesel in [16]. The most beneficial aspect in using CNN is optimizing parameters of Artificial Neural Network (ANN) and being able to take abstract features as input enters deeper layers, so that the features do not depend on the spatial position in the input [17]. The main layers on CNN consists of convolution layer, the pooling layer and the activation layer [16], [18], [19]. The convolution layer consists of a filter that can be "shifted" as far as the length and height of the input image and compute the dot product and the weights [18]. The pooling layer aims to down-sample the input while reduce the complexity of the subsequent layers [17] as shown in Fig. 2.

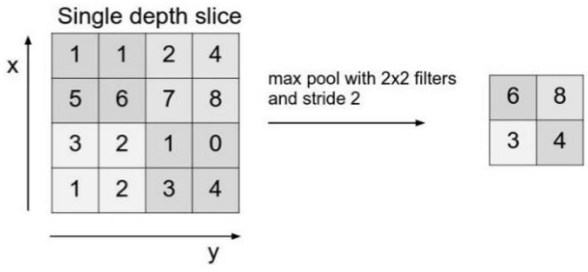


Fig. 2. Max Pooling.

The activation layer aims to provide non-linear computing capabilities. According to [18] and [17], the Rectified Linear Unit (ReLU) activation function is very popular to use because it can facilitate the achievement of *convergence* on Stochastic Gradient Descent (SGD), a simple and easy to understand algorithm. Equations (1) and (2) are representations of the ReLU algorithm.

$$ReLU(x) = \max(0, x) \tag{1}$$

$$\frac{d}{dx} ReLU(x) = \{ 1 \; if \; x > 0; 0 \; otherwise \} \tag{2}$$

D. Autoencoders

Autoencoders is a type of neural network architecture such as CNN or MLP. The specialty of autoencoders are that the output dimensions are the same size as the inputs. The results of the data that came out can be compared with the data that entered so that it can provide training results without labels from the data itself [20].

Fig. 3 is a general structure in Autoencoder which consists of three main parts namely Encoder, Decoder and Bottleneck (Code). Encoder tries to learn how to reduce the input dimensions and compress the input data into the Bottleneck while Decoder functions to learn how to reconstruct the data from Bottleneck to be as similar as possible to the input. Bottleneck is a layer which is the data with the smallest dimension stored from the input.

The workings of Convolutional Autoencoder (CAE) generally have similar structures with ordinary autoencoders. The difference is, the weight is distributed to all locations in the input and maintains spatial locality [21]. CAE can process 2-dimensional structures like images because it uses a convolution layer.

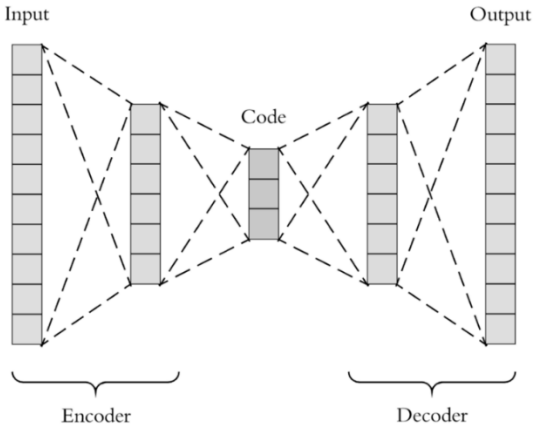


Fig. 3. Common structure of autoencoders.

Various CAEs can be stacked together into Stacked Convolution Autoencoder (SCAE) and form deeper structures as found by [21]. SCAE can be used for unsupervised pre-training and the weights can be fine-tuned with CNN as a feature vector.

E. DeepFont

DeepFont proposed by [4] aims to minimize the problem of domain mismatch [3]. DeepFont gets an accuracy of 80% on its top-5. DeepFont is ideal for font recognition and measuring the similarity of fonts as a suggestion for font selection for its users [4]. DeepFont was trained with 2383 font classes.

Getting real world data is very difficult because large number of accessible text images do not have font information and to provide it requires special expertise that can only be done by certain people [3], [4]. Thus, forming synthetic datasets to get ground truth labels is needed. In addition, to provide domain adaptability, DeepFont was trained using real world unlabeled datasets. These two datasets are used separately on the autoencoder and CNN.

Noise, blur, rotation, shading, character spacing, and aspect ratio augmentations are added on the synthetic data. Using only the 105 x 105 patch for each image sacrifices various features that differentiate between font types. So, the squeezing operation is carried out by changing the width of the image relative to its height. Then cropping the image (crop) to produce image patches from 1 sample measuring 105 x 105. These image patches are the data used in the process of making the DeepFont model.

From Fig. 4, DeepFont consists of 2 CNN fractions combined into one namely **Cu** (red) sub-network and **Cs** sub-network (green). The **Cu** sub-network is formed by training Stacked Convolution Auto Encoder (SCAE) in unsupervised way with synthetic dataset and real dataset using the structure shown in Fig. 5. The first and second layers of SCAE are used as **Cu** layers on the combined CNN and freezed. The **Cs** layer that gets input from **Cu** is trained in supervised way with labeled synthetic data.

The **Cu** architecture are obtained from the SCAE consists of an input layer with a shape of 105 x 105. In the encoder, the first convolution layer produces an output shape of 48 x 48 x 64 then followed by the max pooling layer with an output shape 24 x 24 x 64. The second convolution layer produces an output shape of 24 x 24 x 128 which is the smallest dimension of the autoencoder. Followed by decoders namely deconvolution layer with output shape 24 x 24 x 64 and Up Sampling / Un pooling with output shape 48 x 48 x 64. And finally closed with deconvolution layer with output shape like input shape which is 105 x 105. This architecture was trained using Mean Squared Error loss function and learning rate 0.01. Hyperparameters such as kernel size, optimizer, stride, padding, and epoch are not included in the DeepFont paper.

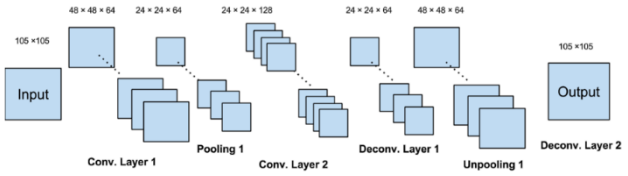


Fig. 5. DeepFont SCAE.

Cu architecture is followed by **Cs** which consists of 3 convolution layers with an output shape of 12 x 12 x 256. Then, 2 fully connected layers with a total of 4096 units using dropout techniques for each layer. The activation layer consists of 2383 according to the number of classes in the AdobeVFR dataset. The combination of **Cu** and **Cs** forms the DeepFont CNN. The classifier used was Softmax and all layers used "relu" activation function. At this stage, the second training was conducted with the parameters batch size 128, momentum 0.9 and weight decay 0.0005. Training is also accompanied by a reduction in learning rate by dividing the learning rate by 10 when the validation error is not reduced at any given time. Hyperparameters not listed in the paper are kernel size of the convolution layer (Conv layers 3, 4 and 5), padding, optimizer, and epoch.

F. Confusion Matrix

Confusion matrix are used to evaluate the performance of a classification system. In accordance with the formulation of the problem, the metrics used are accuracy, precision and recall which can be taken from the Confusion Matrix. The

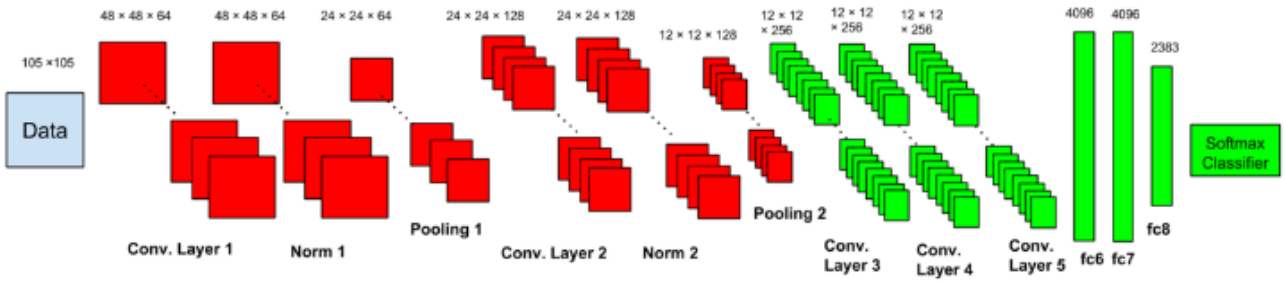


Fig. 4. DeepFont CNN Architecture.

advantage of accuracy testing is easy to compute with low complexity, besides this method is easy to understand [22].

Figure 2. 6 is a table of values in the confusion matrix. TP (True Positive) occurs when the prediction of the model produces a positive value and the facts are also positive. TN (True Negative) occurs when the model predictions produce negative values and facts are also negative. TP and TN occur when the model predicts correctly. FP (False Positive) or Type 1 Error occurs when the model is wrongly predicted by giving a positive value for negative facts. FN (False Negative) or Type 2 Error occurs when the model is wrongly predicted by giving a negative value for facts that are positive. It can be concluded that FP and FN are conditions that the model fails to predict correctly.

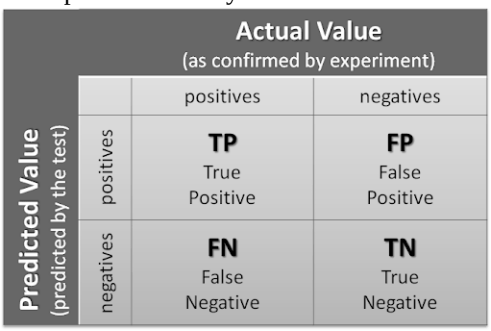


Fig. 6. Confusion Matrix.

Accuracy measures the ratio of correct predictions to the sum of all data evaluated. The accuracy metrics (acc) of the confusion matrix table can be represented in equation (3).

$$acc = \frac{tp+tn}{tp+fp+tn+fn} \quad (1)$$

Precision is used to measure positive patterns that are correctly classified from the total positive class of money predicted. The precision metric (p) is represented in equation 4.

$$p = \frac{tp}{tp+fp} \quad (2)$$

Recall is used to measure the fraction of a positive pattern that is classified correctly. The recall metric (r) is represented in equation 5.

$$r = \frac{tp}{tp+tn} \quad (3)$$

G. Synthetic Data Generation

As noted in section E, synthetic datasets are needed to produce a ground truth label used to train the DeepFont CNN. however, generate the dataset is not well explained in DeepFont paper. In this study, the author uses an open-source library named TextRecognitionDataGenerator (TRDG) to help generate synthetic data needed.

TRDG[1] is an open-source python library for doing text image generation. TRDG has been accompanied by image augmentation features such as skew, affine trainsform, blur and background. however, TRDG has the disadvantage of not having the noise feature because it is incorporated into the background feature in the form of background noise. TRDG also has no background shading modifications. The background feature in TRDG can be used to change the background of an image by accessing a directory containing randomly used images. To produce images, we can simply use the trdg command line followed by parameters such as *-output_dir [address]* and *-c [image count]*.

H. Google Colab

Collaboratory or abbreviated as "Colab" is a free service by Google that allows the public to write and execute Python code from a browser [23]. This facility is perfect for the needs of Machine Learning, data analysis and education. Basically, Google Colab is hosted by Jupyter Notebook that provides GPU facilities for free.

Limitation of Google Colab itself is there is an idle timeout, maximum VM lifetime, limited GPU supply and limited memory supply. however, Google does not publish these limitations because they can change overtime [23]. GPU types provided by Google include Nvidia K80, T4, P4 and P100 with up to 32 GB of RAM memory allocation. However, users cannot directly choose the GPU when using Google Colab. Google provides services for users who want to use more facilities through Colab Pro.

In this study, Google Colab is used to conduct training, data visualization and model evaluation. The key feature used in Google Colab is that it can run code in groups (cells) and only show the output of that group. Google Colab is also able to run Linux commands to perform system operations (move, copy, paste, unzip, etc.).

I. Tensorflow Keras

Tensorflow is an open-source platform for machine learning. Tensorflow was formed by researchers and developers from the Google Brain team in Google's Machine Intelligence Research to conduct research on machine learning and deep neural networks [24]. Keras is a multibackend high-level API for building and training models that can be used on the Tensorflow, CNTK and Theano backends [25].

Tensorflow Keras is used in this study because it has key advantages. Some of the advantages of Tensorflow over other frameworks are that they are the most popular and developing Deep Learning tools and are supported by strong developers compared to other competitors [26]. In addition, Tensorflow is also efficient in performing mathematical calculations and multi-dimensional arrays. Tensorflow 2.0 officially forms a special submodule to support the API under the name tf.keras. Advantages of Keras is a high-level API that minimalistic, user-friendly, and modular (Nguyen et al., 2019). Merging through these modules can complement Tensorflow's weakness which is still a low-level API in forming a modular deep learning model. The above advantages are the main reason for choosing Tensorflow Keras as the main library in the formation and training of the DeepFont model.

This study uses various functions in Tensorflow Keras. Among them is ImageDataGenerator on Keras.preprocessing to access the image directory. The Model.fit () function is for training models that are made from Keras.layers layer libraries such as (Dense, Conv2D, UpSampling2D, etc.) and Keras.optimizers library to provide optimizers such as (SGD, Adam, etc.). Keras also has the ReduceLROnPlateau feature to modify training methods by reducing the learning rate by considering existing metrics (accuracy, val_loss, val_accuracy, etc).

The precision and recall metrics have been removed by Keras. Thus, to evaluate the model, Sklearn.metric library have been used. *precision_score* and *recall_score* function in the library can be used to calculate precision and the recall value. The confusion matrix can be created using

[1] https://github.com/Belval/TextRecognitionDataGenerator.

confusion_matrix function. And to generate performance reports for each of its classes can use *classification_report* function.

III. METHODOLOGY

A. Specification Analysis

This analysis aims to determine the system specifications based on the needs of users to create a font recognition application. The results of this analysis will be the basis for making the system in this study. The data collection method used is an *offline* quantitative survey. The results of the survey were analyzed using a frequency distribution in determining the specifications of the *font* recognition application. The respondents addressed by this survey are graphic design practitioners in Universitas Ciputra, especially students with Visual Communication Design Program with graphic design specialization. The number of respondents collected was 33 respondents.

To determine the specifications of the font recognition system, aspects of the question need to be adjusted to the user's needs. The following aspects of the main questions based on the needs of making the system are:

1) Type of Windows Operating Systems
2) Number of fonts
3) Input flow for the app
4) Output flow for the app
5) Other features

Aspect 1 is used to determine the operating system used by the application. Aspect 2 is used to determine the number of font classes that are trained in the font recognition model. Aspects 3 and 4 are used to determine the appearance and input and output methods of the application. Aspect 5 is used to identify other additional features as application develop. These aspects can be derived into questions as shown in Table I.

TABLE I: DERIVATION OF SPECIFICATION ANALYSIS SURVEY QUESTIONS

Aspect	Question	
Type of Windows Operating Systems	1	What version of the Windows operating system are you using?
Number of fonts	2	How many fonts do you use often?
	3	How many fonts do you have in your computer?
Input flow for the app	4	If you want to do font recognition on a image, what kind of input flow do you choose?
	5	What kind of input display do you expect?
Output flow for the app	6	How will the result be presented?
	7	What kind of output display do you expect?
Other features	8	What additional features are expected besides font recognition?

From Aspect 1, the operating system version used is Windows 10 with a percentage of 67%, 18% are still using Windows 7 and 15% are not using Windows. There are no respondents who still use Windows Vista (0%) or Windows XP (0%). From the Aspect 2, There are less than 100 fonts with a total of 24 respondents followed by 101-500 fonts as many as 6 respondents and 501-1000 fonts as many as 3 respondents. There are no respondents who answered more than 1000 fonts. 23 respondents stated that there were 101-500 fonts available on their computers, 5 respondents answered 501-1000 fonts , 4 respondents answered less than 100 fonts and 1 respondent answered above 1000 fonts. From Aspect 3, 52% of respondents chose drag and drop method, 21% of respondents chose capture and cropping, 18% of respondents chose to access files individually and 9% of respondents chose to use a laptop camera. From Aspect 4, 34% of respondents choose to display the full *font* type by showing an example of the text using the introduction *font*, 27% of respondents choose to show the full *font* type with its accuracy numbers, 24% of respondents choose to only show the type *typeface* only, and 15% respondents choose to show only the full *font* types . Fig. 7 shows the results of input and output display from the specification analysis survey which 52% of respondents choose the input display (a) and 91% has choose the output display (b).

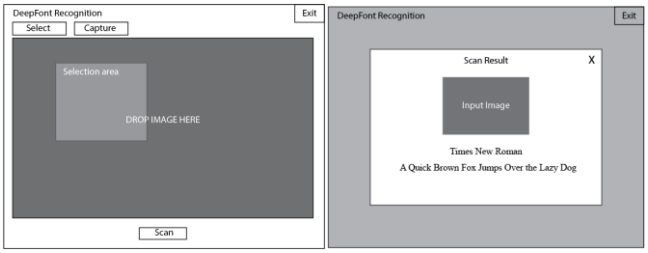


Fig. 7 User Interfaces based on specification analysis survey result. (a) input display; (b) output display.

The survey results state that most respondents have used Windows 10 (67%) thus answering Aspect 1 regarding the operating system used on this system. The number of fonts owned and used by the user has a very contrasting difference. Although the dominant respondent (23 people) has 101-500 fonts used, the survey results prove that the fonts used are below 100 fonts (24 people). Due to the limited computing power and the maximum number of fonts that the respondent has as many as 500 fonts, the number of fonts that can be recognized by the application is 500 fonts thus answering Aspect 2. For Aspects 3, the survey results are using the drag and drop method and use the input view as shown on Fig. 7 (a). As for the output flow, the survey results state that the dominant respondent chose to display the full font type and showed an example of the text using the recognition font (34%) compared to showing the curation number (27%) and typeface type (24%). The output display chosen by the respondent is the output display as shown in Fig. 7 (b). Thus, answering the aspect 4 specification.

B. Building DeepFont Architecture

The overall pipeline for building the DeepFont architecture can be shown in Fig. 8 In the first stage, a gradation background is made which is used as a background to recognize the font. Then, followed by the generation of synthetic datasets using font files that have been downloaded from the internet using the available background shading. The generation process produces a synthetic data directory which is then processed together with a real dataset (AdobeVFRDataset) that has been downloaded from the internet. The results of the preprocess produce pieces of font images that are divided into two main directories namely **syn_train** and **real_train**. The dataset on **syn_train** and **real_train** is used to train the Autoencoder and CNN models which then produce the DeepFont model. The DeepFont model can be interacted using DeepFontAPI.py for application purposes. The following steps are explained in detail in the next section.

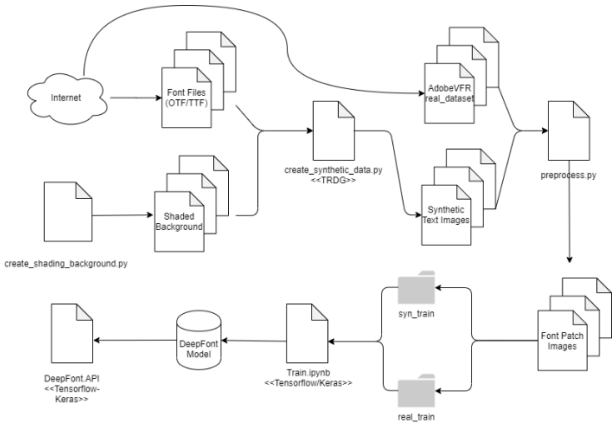


Fig. 8. Pipeline for building DeepFont CNN

1) *Generating Dataset*

There are two datasets used, they are real dataset and synthetic dataset. Real dataset is a text image that is not labeled and obtained from the real world. Synthetic dataset is a dataset generated from the reconstruction of text images from a computer. Fig. 9 is a sample of the real dataset adopted from the Adobe VFR real_u dataset shared by Z. Wang [4]. The dataset contains 201,853 images in various formats. In this study, the formats used are jpeg, jpg, png, tiff, bmp, and gif.

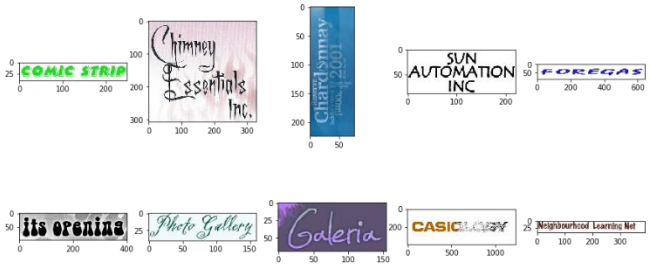


Fig. 9. Adobe VFR Real_u dataset samples

To produce a synthetic dataset, the author uses the TRDG (TextRecognitionDataGenerator) library. TRDG can help produce images with random text and augmentations that have been provided. Standard perturbation for general images was made similar to the DeepFont paper that is blur, rotation, background shading and noise. Blur, rotation, and background shading augmentations are carried out simultaneously. Noise is added at the time of preprocessing is manually due to feature Gaussian noise at TRDG is a feature background that becomes one with the pictures (for the input feature image background shading) so that the process should be separated to apply both features at once.

The process of generation of synthetic datasets can be seen in Fig. 11. The background shading image is generated in create_shading_background.py which generates images with a gradation background. Font files in the form of OTF and TTF are used to produce text along with background shading by TRDG. The synthetic dataset produced was 1,000 images for each 2,507 fonts named synthetic text.

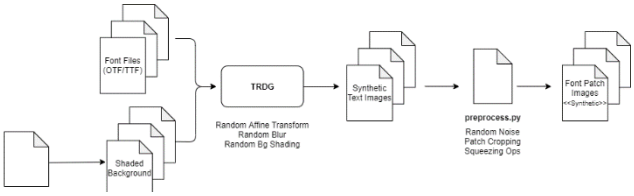


Fig. 11. Creation flow of synthetic dataset

The number of fonts used is as many as 500 fonts. So, from 2,507 fonts selected equally 250 serif fonts and 250 sans-serif fonts. Adjustment of the size of the real dataset is also done using a comparison of the number of font classes with the size of the real dataset. DeepFont has 2383 font classes and uses 201,853 real images while this study only uses 500 font classes. The number of real images in this study can be calculated using equation (6).

$$\frac{NFontClassDeepFont}{NDatasetRealFile} = \frac{500\ Fonts}{NDatasetReal} \quad (6)$$

NFontClassDeepFont is the number of supported font classes that is 2383 font classes and NDatasetRealFile is the number of real files used by DeepFont which is 201,853. With this equation 42,352 images can be obtained to support 500 fonts in this study. Figure 3. 14 is a sample of a synthetic dataset that was successfully generated using TRDG.

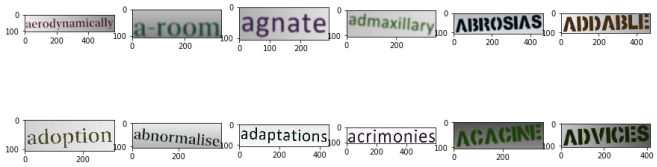


Fig. 10. Synthetic Dataset sample.

2) *Preprocessing Dataset*

The preprocessing has been carried out by the dataset generation stage with the help of TRDG namely blur, rotation, and background shading. At this stage, it focuses on squeezing operations, cropping, adding noise and grayscale. Squeezing operations, grayscale and crop are helped using the PIL library. Noise is generated using the help of the numpy library. The image is converted into a *numpy* array which is then added to a random value and limited by a value between 0 - 255. The array will again be read as an image again. Afterwards, the results of preprocessing are organized into separate folders to form a **real_train** dataset (for unsupervised) consisting of image fragments from real and synthetic datasets while **syn_train** (for supervised) consisting of only synthetic dataset images. Figure 3. 15 are some samples from 1,718,754 pieces of images in real_train and some samples from 1,668,521 pieces of images in syn_train.

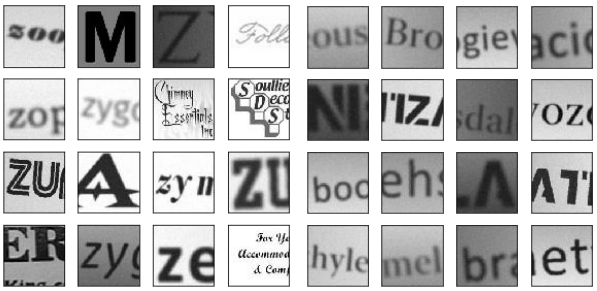


Fig. 12. Preprocessing results real_train (left) and syn_train (right)

3) *Implementing Stacked Convolution Autoencoder*

Unsupervised Training is conducted on Google Colab using the autoencoder architecture from DeepFont. Stacked Convolution Autoencoder (SCAE) is trained using **real_train**, which is a combination of real and synthetic datasets.

Error! Reference source not found., is the structure of the autoencoder that is made to follow the output shapes according to the DeepFont paper using Keras library, the autoencoder formation using the Conv2D, MaxPooling2D,

Conv2DTranspose and UpSampling2D classes. To form an autoencoder model hyperparameters such as kernel size, stride and padding are needed. The parameters in the table were chosen because they can form image shape representations in accordance with the DeepFont paper with minimal parameters.

TABLE II: INITIAL AUTOENCODER PARAMETERS

Name	Kernel size	Stride	Input size	Output size	Filters	Padding
Input				105	1	-
Convolution 1	11	2	105	48	64	valid
Maxpool 1	-	2	48	24	64	-
Convolution 2	1	1	24	24	128	valid
Deconv1	1	1	24	24	64	valid
Upsampling	-	2	24	48	64	-
Deconv2	11	2	48	105	1	valid

Valid padding is chosen because the same padding provides equal output size and input size assuming stride is 1. Changing the kernel size will not affect the output size if we use the same padding on the Keras library. To meet the output size requirements in the first convolution layer (Convolution 1) is to use valid padding which is the basis of the initial parameters of all convolution layers in the initial autoencoder.

Kernel size (K) can be found using equation (8) which is substituted from the convolution equation (7). Where the value W is input size, O is output size, K is kernel size, P is padding and S is stride. Valid padding or no padding will give a value of 0 on. can be selected for. The larger the kernel size, the greater the resulting parameters, so that the value of the selected kernel size is the one that produces the smallest parameter. For example, if S = 1 and K = 58 will result 215360 parameters, then if S = 2 and K = 11 with 7808 parameters and so. Then the selected is kernel size (11,11) because it has the most minimal parameters.

$$O = \frac{W-K+2P}{S} + 1 \quad (7)$$

$$K = W + 2P - S(O-1)\{K \mid K > 0\} \quad (8)$$

Based on DeepFont paper, the learning rate used is 0.01 and uses the loss function Mean Squared Error (MSE). Because the optimizer is not explained by the paper, the authors chose the Stochastic Gradient Descent (SGD) which has a standard learning rate of 0.01. Other parameters that need to be determined are the number of epochs and the size of the kernel size because they are not explained in the DeepFont paper. Therefore, the author does some testing using **real_train**. The results of epoch testing can be seen on Table III.

TABLE III: EPOCH TESTING RESULTS ON AUTOENCODER

AE Nama	Epoch	Params	Loss, Val_Loss	Time(s)	Desc
Autoencoder_e10_SGD	10	32129	0.0054, 0.0067	10201.44 (2.8 hr)	A little blurry, there is a white border.
Autoencoder_e5_SGD	5	32129	0.0059, 0.0074	6008.07 (1.67 hr)	A little blurrier than e10_SGD, there is a white border
Autoencoder_e20_SGD	20	32129	0.0050, 0.0062	19533.3s (5.43 hr)	The picture clearly looks little blurry, there is a white border

If a comparison is made between the number of epochs on e5 and e10, then 1: 2 difference is generated, the time difference between e5 and e10 is 1.13 hours which can give a difference of loss of -0,0005 and val loss of -0,0007 in 5 epochs. The time difference between e20 and e10 is 2.63 hours can give a difference of loss of -0,0004 and val_loss -0,0005 in 10 epochs, which means -0,0002 and -0,00025 large loss and val loss if for every 5 epochs. With this, the authors use epoch 10 because of the effective time and effective effect compared to e5 and e20.

Fig. 13 is the result of reconstruction of each autoencoder in the epoch test. We can see the appearance of the "border" effect. These border effects can be minimized by adding padding to the convolution layer [27]. The convolution layer parameter only uses "valid" or no padding to fill the shape of the DeepFont paper. Therefore, to find the kernel size value, the addition of "same" padding is done along with the kernel size.

Fig. 13 Reconstruction results of epoch testing. (a) Original; (b) E5; (c) E10; (d) E20

To determine the new kernel size, the autoencoders are trained with (1,1), (3,3) and (5.5) in 10 epochs. Determined kernel size has to consider the number of parameters and model quality (loss and loss val). Table 3. 4 is the result of kernel size testing. In Autoencoder_e10_3_SGD and Autoencoder_e10_5_SGD given 'same' padding and changes in the kernel size are proven to minimize border effects and sharpen the results of loss and val_loss significantly than in (1,1).

TABLE IV: KERNEL SIZE TESTING RESULTS ON AUTOENCODER

Nama AE	Conv1	Conv2	Loss, Val_Loss	Time	Desc
Autoencoder_e10_SGD	(11,11), s=2, (valid)	(1,1), s=1, (valid)	0.0054, 0.0067	2.8 hr	A little blurry, there is a white border.
Autoencoder_e10_3_SGD	(11,11), s=2, (valid)	(3,3), s=1, (same)	0.00248, 0.00339	2.6 hr	Details, some samples do not show borders.
Autoencoder_e10_5_SGD	(11,11), s=2, (valid)	(5,5), s=1, (same)	0.00246, 0.00339	2.6 hr	Details, some samples do not show borders

Figure 3. 18 is the result of reconstruction using Autoencoder on each kernel size tested. There are no border effects on the kernel size (3.3) and (5.5). Testing the kernel size (3,3) and (5,5) is time consuming and gives relatively the same loss and val loss results so that it gives identical results.

Fig. 14. Reconstruction results of kernel size testing. (a) Original; (b) 1,1; (c) 3,3; (d) 5,5

In this study, the authors used a kernel size (3.3) for all convolution layers on CNN with 10 epoch training parameters to optimize the magnitude of the parameters and the time spent in forming the DeepFont model.

4) Implementing DeepFont CNN

Supervised training is also conducted at Google Colab. Continuing the previous training phase, the supervised phase aims to create the DeepFont model itself so that it can classify font types using layers in the Autoencoder that have been trained on real data. Table V is the structure of CNN used in this study. The autoencoder layer that is inserted into the CNN is represented in **blue**. The resulting output shape is similar to Fig. 4.

TABLE V: FORMED DEEPFONT CNN STRUCTURE

Name	Kernel Size	Stride	Input size	Output size	Filters	Padding
Input				105	1	-
Convolution 1	11	2	105	48	64	valid
Batch Normalization	-	-	-	48	-	-
Maxpool 1	-	2	48	24	128	-
Convolution 2	3	1	24	24	128	same
Batch Normalization	-	-	-	24	-	-
Maxpool 2	-	2	24	12	128	-
Convolution 3	3	1	12	12	256	same
Convolution 4	3	1	12	12	256	same
Convolution 5	3	1	12	12	256	same
Flatten						

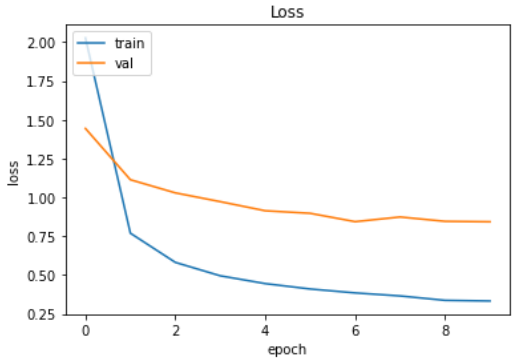
MLP	Units	Fungsi	Add On
Dense 6	4096	ReLU	Dropout
Dense 7	4096	ReLU	Dropout
Dense 8	500	Softmax	

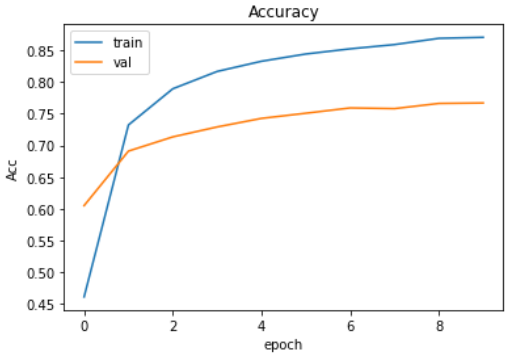
Next, the training continued with 10 epoch parameters and the kernel size for convolution layers 3,4 and 5 is (3,3). The optimizer used is Stochastic Gradient Descent (SGD) because it has learning rate, momentum, and decay parameters. SGD was formed with a learning rate of 0.01, momentum 0.9 and decay 0.0005 in accordance with DeepFont training requirements. The loss function uses the Sparse Categorical Crossentropy due to the nature of the dataset, 1 image inferring 1 class. To reduce the learning rate if validation loss does not improve, **ReduceLROnPlateau** is used with the monitor parameter "val_loss", a factor of 0.1, which means the learning rate is 0.1 times if val_loss does not improve. The data used is syn_train.

Figure 3. 19 is a graph of Loss, Accuracy and Learning Rate results from the DeepFont CNN training. The results of the training were completed in 6196.88 seconds (1.7 hours) with the last epoch loss 0.33, val_loss 0.84, val_acc 0.77, acc 0.87. The last learning rate detected was 0.0001. The learning rate dropped at the 8th epoch when val_loss stagnated. This is in line with the accuracy that has begun to stagnate in epoch 8.

C. Building Font Recognition Application

Windows based applications are used to present CNN models that have been made. The application uses C # with a python backend to support the CNN model. Fig. 16 is an application architecture design chart. In the picture,





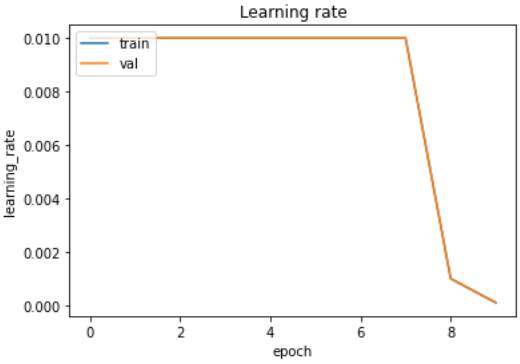


Fig. 15 Results of loss, accuracy, and learning_rate of CNN supervised training

architecture is divided into two major parts, namely the frontend and backend. Frontend is a place where users interact with font recognition applications while Backend where applications get font recognition results. Frontend applications are formed using Windows Form App by the .NET Core Framework. Python scripts are used to run processes using the Python Interpreter that is run by the application. Python Interpreter has the duty to load the model and provide prediction results back to the application to be presented to the user.

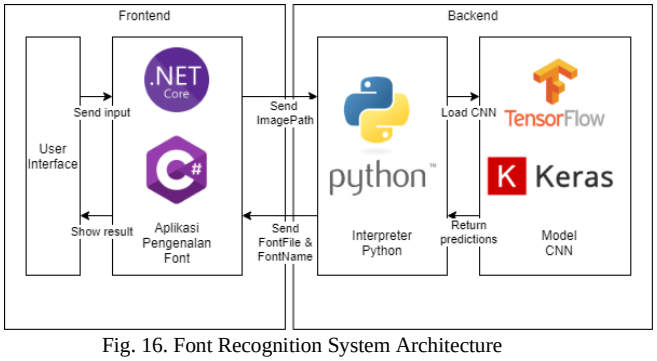


Fig. 16. Font Recognition System Architecture

1) User Interface

The user interface selected is using Drag and Drop as input and raises output using the Pop Up. The pop up shows the full font labels (e.g. light, bold, italic) and displays sample text using recognized font. The Pop Up is also given the ability to display more than one prediction result.

2) Use Case Diagram

Use Case Diagram show user's interactions with the application. Fig. 18 shows the user can perform the font recognition based on input image.

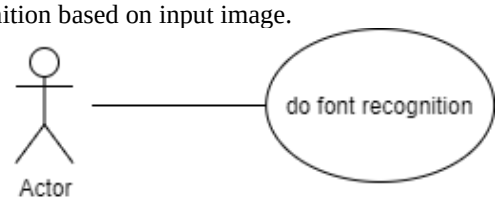


Fig. 18. Use Case Diagram.

3) Activity Diagram

Fig. 20 is the first page of the Activity Font recognition diagram. The user can determine whether to use the drag and drop feature or not. If the user uses drag and drop, user can only drag and drop image files. If users choose to use the "Open" button, the system opens a file picker dialog where the user can choose the file is directly from their directory. Then the application checks whether the file is an image or not. If the file is not an image, then the system raises an error dialog indicating that the file must be an image. If the user uses the file chooser dialog feature, it will be reappeared until the user enters an image file correctly.

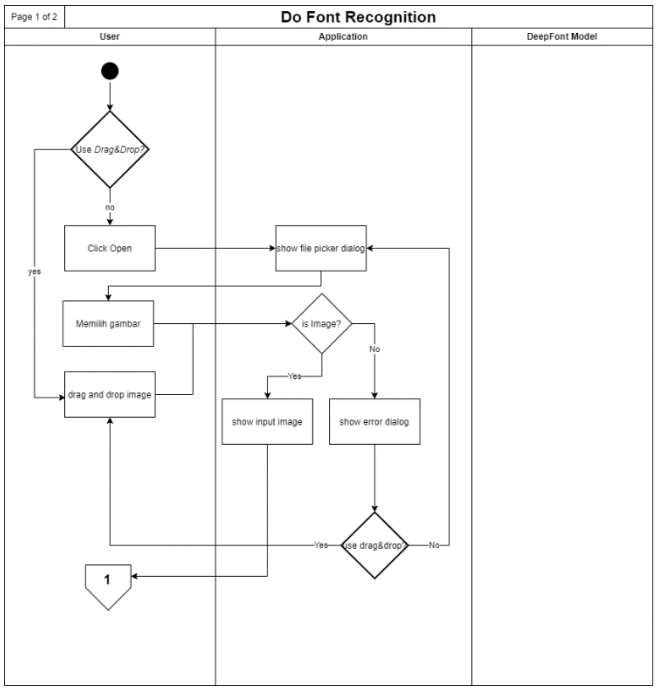


Fig. 20. Font Recognition activity diagram part 1

After the image successfully displayed by the system, the user can perform Crop or directly perform Scan. If the user decides to do Cropping, then the user can directly perform Click and drag on the image and the system shows the area of the crop. If the user clicks Crop, the system cuts the image according to the crop area. After the user decides to do a Scan, the system takes the image address and runs the DeepFont script in Python Interpreter. The script that instructs the system to open the address of the image that and do predictions on the image. The prediction results are then read by the application which then shows the prediction results from the model. The second part is shown on Fig. 17.

4) Class Diagram

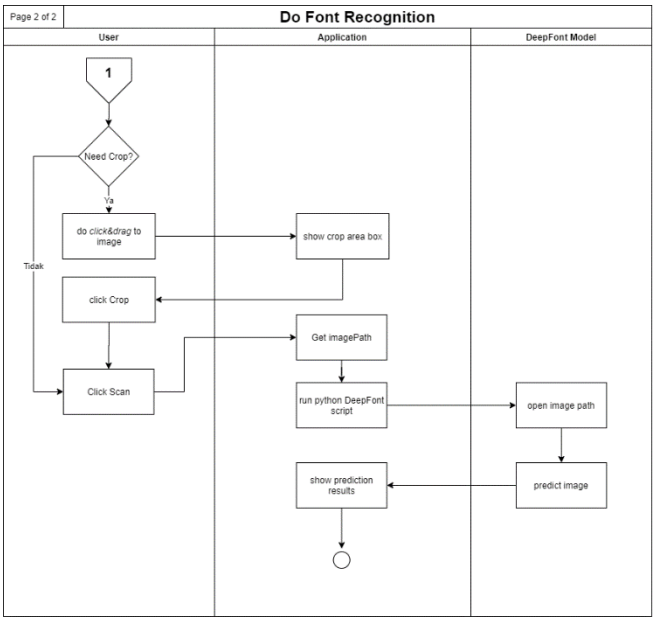


Fig. 17 Font Recognition activity diagram part 2

Fig. 19 is a Class Diagram of the font recognition application. The MainForm class functions as the main controller in the application. Class that handles input from the user either by drag and drop or OpenFile, cropping, run the script python on Scan and call dialog MsgBox to display the prediction results. The MsgBox class is used as a controller to control the display of results that are presented to the user.

MainFormUI is a user interface that holds interaction elements such as Button, Label and ImageBox. CropButton

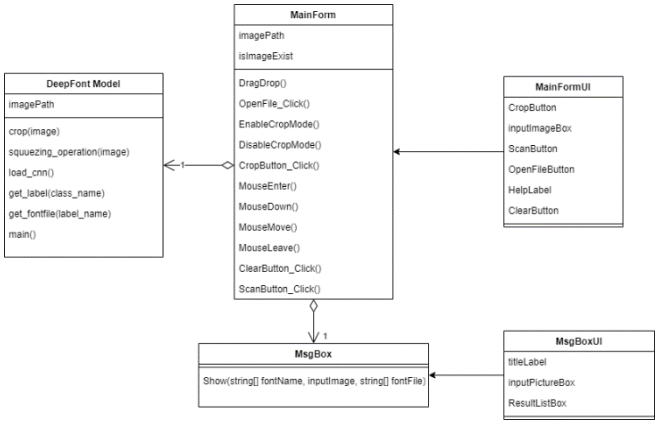


Fig. 19. Class Diagram

is a Crop button, inputImageBox is an input area that can be used to drag and drop and display input images , Scan Button is a button to start font recognition , OpenFileButton is a button to open the Open File dialog, HelpLabel is a label to display instructions for users , Clear Button is a button to delete and reset UI conditions .

MainForm is a controller from the MainForm UI. ImagePath on MainForm used to accommodate the address image that can be of DragDrop or OpenFile. IsImageExist is used as an indicator whether there are images in the InputImageBox. DragDrop invoked when users perform drag and drop a file into the area InputImageBox. OpenFile_Click invoked when the user clicks the button OpenFileButton and

initiate OpenFileDialog. EnableCropMode is a function to enable the ability to crop in the application and allow users perform cropping and click and drag on InputImageBox. DisableCropMode is the opposite of EnableCropMode. CropButton_Click invoked when the user clicks the button Crop after selecting the crop area. CropButton_Click function also place the cropped image inside InputImageBox. MouseEnter is a function called when the mouse entered the InputImageBox area. MouseDown function is invoked when the mouse is clicked. MouseMove function called when the mouse is moved inside InputImageBox area. MouseLeave function is called when the mouse leaves the InputImageBox area. ClearButton_Click function called at the time the button CleanButton clicked by the user. ClearButton_Click deletes the image from InputImageBox, deletes the contents of ImagePath and call DisableCropMode. ScanButton_Click called when user clicks the Scan button that will start the process of Python DeepFont Model and initiate MsgBox if already completed.

DeepFont Model is a python script which is run by MainForm. This script aims to make prediction process and return the font recognition results. ImagePath is supplied by the application and used to access the images. Crop and Squeezing Operation function perform image preprocessing before the data is inserted into the model. Load_cnn is a function to load the CNN model. Get_label is a function to translate class names into font names and get_fontfile is used to get the font name of the file based on the label obtained. Main is the main function in the script to run other functions.

MsgBoxUI a UI box dialog and holding elements of interaction that consists of ListBox and InputPictureBox . ListBox is used to display prediction results that have been sorted as Top Predictions. InputPictureBox is used to show input images from users.

MsgBoxUI has a controller that is MsgBox. MsgBox's show function is to give the text labels and displays the UI as a form of poppup with the font recognition results, input image and array of font file of the recognition results.

5) Sequence Diagram

Fig. 21 explains the operation in more detail how the application performs operations from the beginning of the user to insert the image to process input to provide predictive results to the user. Users can click Open on the UI MainForm, then MainFormController bring up an Open File Dialog and entering the loop up to the user to enter an image file. If the file is an image, then it is displayed in the MainForm UI whereas if it is wrong then an error dialog box is raised. Users also can directly perform drag and drop where MainFormController also checks file whether is an image or not. If the image has been loaded, the user can perform cropping on the image. MainFormController responds by bringing up the crop area on the UI.

When user presses the Crop button, the Main Form Controller crop the selected area and display it to user interface. When the user clicks Scan, the controller runs DeepFont Model *script* passing the image path. Then, the model loads, predicts the image and returns the results.

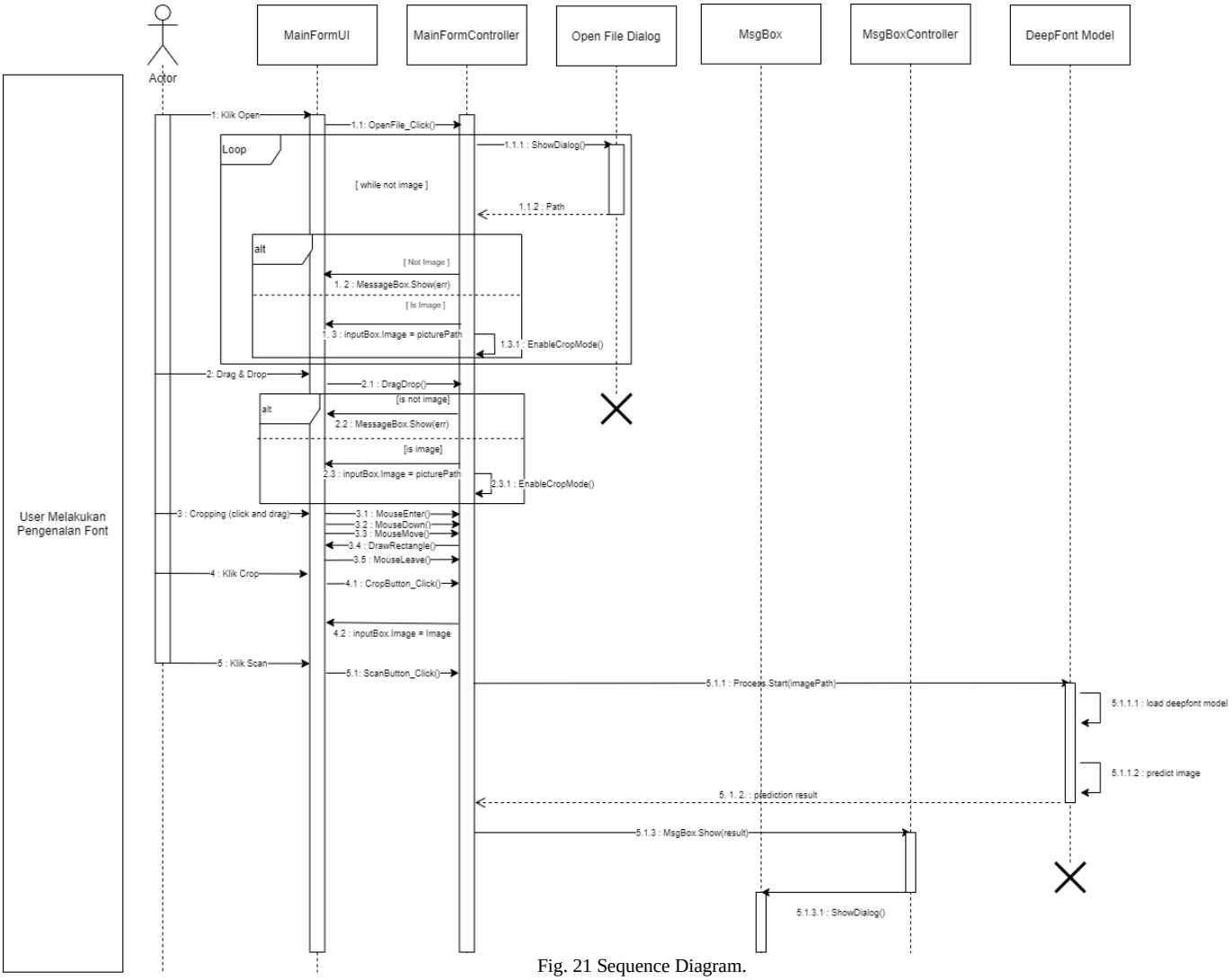


Fig. 21 Sequence Diagram.

IV. Results and Discussion

A. DeepFont Model Performance

To know the performance of the model, we test the model with another similar dataset. Due to the difficulty of finding text images datasets with the same font class and groundtruth labels, the author conducted a generation of synthetic datasets with different augmentation method from the training set. The generated dataset including no augmentation, partial augmentation, and full augmentation similar to the training set. The partial augmentation including random blur, affine transform, and noise. Each dataset is tested once using Google Colab. The test results are precision, recall, confusion matrix and classification report along with the top 5 best and top 5 worst fonts. The samples are shown on Fig. 22.

Fig. 22. Test Dataset Samples

Table V is the result of testing the metrics for each dataset that has been generated. Evaluation using the first dataset produces accuracy rate of 86%, a precision level of 86.2% and a recall rate of 86.1%. The second dataset produced accuracy rate of 77%, a precision level of 80% and a recall rate of 76.9%. The third dataset produces an accuracy rate of 73%, precision level of 76% and recall rate of 72.5%.

TABLE VI: Model Evaluation Results

No	Dataset	Generation	Accuracy	Precision	Recall
1.	Dataset_test_50	Augmentation same as train	86%	86.2%	86.1%
2.	Dataset_test_50_ noAugmen	Without augmentation.	77%	80%	76.9%
3.	Dataset_test_50_ RBKN	Augmentation in the form of random blur, random transform, random noise	73%	76%	72.5%

The reduced accuracy of the dataset without augmentation or partial is caused by the uneven distribution of augmentation in the training dataset, causing the model to overfit the segmented data. Generations using TRDG do not provide the option to deploy augmentation using a function such as Gaussian but instead rely on random .

Table VII is the result of five fonts with the best accuracy performance that has been normalized based on the number of samples. Which belong to the T op 5 Best Font on Dataset_test_50 is Wolf's Bane Outline (100%), Grendel's mother (100%), Gill Sans MT Ext Condensed Bold (100%), Bombing (100%), VTC Belials Blade Tricked (99.7%). The top 5 best fonts on Dataset_test_50_RBKN are Wolf's Bane Outline (100%), Wolf's Bane (100%), Juice ITC (100%), Han Solo Shadow Italic (100%), Grendel's Mother Shadow Italic (100%). The top 5 best fonts on Dataset_test_50_noAugmen are Wolf's Bane Outline (100%), MeninBlue (100%), Juice ITC (100%), Han Solo Shadow Laser (100%), Han Solo Condensed Laser (100%).

TABLE VII: Top 5 Best Performing Font

No	Dataset_test_50	Dataset_test_50_ RBKN	Dataset_test_50_No Augmen
1	Wolf's Bane Outline (100%)	Wolf's Bane Outline (100%)	Wolf's Bane Outline (100%)
2	Grendel's Mother (100%)	Wolf's Bane (100%)	MeninBlue (100%)
3	Gill Sans MT Ext Condensed Bold (100%)	Juice ITC (100%)	Juice ITC (100%)
4	Bombing (100%)	Han Solo Shadow Italic (100%)	Han Solo Shadow Laser (100%)
5	VTCBelialsBlade Tricked (99.7%)	Grendel's Mother Shadow Italic (100%)	Han Solo Condensed Laser (100%)

The Wolf's Bane Outline font is easier to recognize in all three datasets despite using different image modifications. This is because the font has the most different characteristics compared to the entire class in the dataset. The font is the only one that has outline characteristics. This uniqueness makes the model able to differentiate the font above other fonts in the dataset. This can also happen to other fonts that have high accuracy across three datasets. Fig. 23 shows the sample of the largest fonts listed as the top 5 of the three datasets.

Fig. 23 Font samples showed on the top 5

Table VIII is the result of five fonts with the worst accuracy performance that has been normalized based on the number of samples. The top 5 worst fonts in Dataset_test_50 are HK Grotesk Light Legacy (23.3%), HK Grotesk Medium Italic Legacy (23.8%), Lucida Grande (25.4%), Gill Sans Ultra Bold (26.5%) and Century Schoolbook (27.6%). In Dataset_test_50_RBKN are Gill Sans MT (0.77%), Humnst777 BlknCn BT Black (0.8%), Gill Sans Nova Bold (1.5%) and HK Grotesk Regular (2%) . In Dataset_test_50_NoAugmen are Humnst777 Cn BT Bold (0.6%), Granada Antique Bold (1.1%), Century751 BT Roman (1.8%), Gill Sans MT (1.8%) and OFL Sorts Mill Goudy TT (2.6%).

TABLE VIII: TOP 5 WORST PERFORMING FONT

No	Dataset_test_50	Dataset_test_50_RBKN	Dataset_test_50_NoAugmen
1	HK Grotesk Light Legacy (23.4%)	Gill Sans MT (0.77%)	Humnst777 Cn BT Bold (0.6%)
2	HK Grotesk Medium Italic Legacy (23.8%)	Humnst777 BlkCn BT Black (0.8%)	GranadaAntique-Bold (1.1%)
3	Lucida Grande (25.4%)	Adobe Arabic Regular (0.87%)	Century751 BT Roman (1.8%)
4	Gill Sans Ultra Bold (26.5%)	Gill Sans Nova Bold (1.5%)	Gill Sans MT (1.8%)
5	Century Schoolbook (27.6%)	HK Grotesk Regular (2%)	OFL Sorts Mill Goudy TT (2.6%)

The HK Grotesk Light Legacy font (23.4%) has the worst performance on Dataset_test_50. The font is misidentified as HK Grotesk Light (48.9%), Humnst777 Lt BT Light (9.4%) and Calibri Light (5.8%). Fig. 24 shows the font Grotesk Light Legacy is very similar to the font that have a higher degree of accuracy.

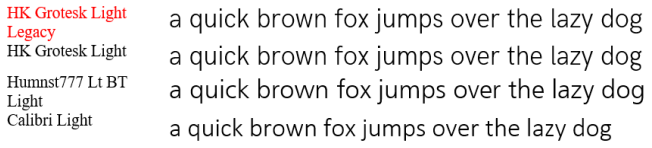


Fig. 24 Comparative Sample for HK Grotesk Light Legacy

The worst accuracy in the Dataset_test_50_RBKN is Gill Sans MT with 0.77% accuracy. From the confusion matrix, Gill Sans MT was misidentified as Humanist 521 East (66%) and Humanst521 East Roman (18.4%). This happens because the Gill Sans MT font is very similar to Humanist 521 BT and Humanist 521 BT Roman so that the model is very difficult to find the features that distinguish the font. The comparative samples are shown on Fig. 25.

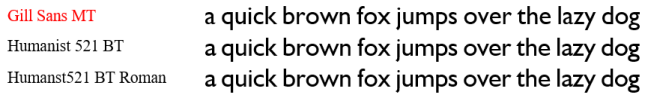


Fig. 25 Comparative Sample for Gill Sans MT

The worst accuracy in the Dataset_test_50_noAugmen is Humnst777 Cn BT Bold (0.6%). The font are misidentified as GlasgowSerial-Medium-Regular (94.5%) and Arial Narrow Bold (41.4%) by the model. Like the two previous datasets, this is also due to the model's difficulty in differentiating the font from other fonts. Figure 5. 5 shows Humnst777 Cn BT Bold is very similar to Glasgow Serial - Medium-Regular and Arial Narrow Bold.

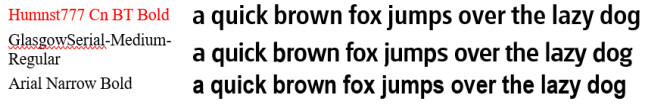


Fig. 26. Comparative Sample for Humnst777 Cn Bt Bold

According to the analysis of the top 5 best and top 5 worst, it can be seen that the more unique a font is, the model can easily recognize the unique features of the font. On the contrary, the more similar a font is to other fonts, the model has difficulty learning the features that distinguish the font.

B. *Application Performance*

To test the application *bug-free*, the author uses various scenarios to test whether a bug is found or not on each feature. Each of these scenarios is reduced to various test cases. Each test case is tested as much as ten times to determine whether the test case is declared a success or not.

The main scenario being tested is to recognize the font. The main scenario is divided into five sub-scenarios which include:

1) Testing the Drag and Drop feature
2) Testing OpenFile features
3) Testing the Crop feature
4) Test Scan feature
5) Testing the Clear feature

The subscenarios is broken into test cases to test the various possibilities that can occur in each subscenario of the main scenario. There are 13 test cases and 10 tests each for each test case. The results show that all test cases were successful, and no bugs occurred in the application. But when the scenario 4 test is carried out, the time to release the results from the prediction requires quite a long time. The author tested scenario 4 as much as five times with an average waiting time of 28 seconds with the longest waiting time of 43 seconds.

TABLE IX: TEST SCENARIOS AND TEST CASES

Test Scenario ID	Descriptive Scenario Test	Test Case ID	Test Case Desc
TS_1	Test the Drag & Drop File	TS_1_1	drag and drop image files
	Test the Drag & Drop File	TS_1_2	drag and drop files not images
	Test the Drag & Drop File	TS_1_3	drag and drop image files when cropping
	Test the Drag & Drop File	TS_1_4	drag and drop files not images during cropping
TS_2	Test Open File	TS_2_1	open image file
	Test Open File	TS_2_2	open file not an image
	Test Open File	TS_2_3	Test open image files when Cropping
	Test Open File	TS_2_4	Test open files instead of images when cropping
TS_3	Test Crop Tools	TS_3_1	Make a *crop* box
	Test Crop Tools	TS_3_2	Cropping
TS_4	Test Scan	TS_4_1	Perform a font recognition from the input image
	Test Scan	TS_4_2	Perform an introduction to the font of the input image after *cropping*
TS_5	Test the Clear button	TS_5_1	Reset the application to its default state.

Based on the results of tests conducted by the author, the application runs smoothly without any bugs that cause the system to crash. Therefore, the application was successfully implemented and integrated with the DeepFont CNN model properly.

The waiting time in scan feature is caused by the process of loading a large CNN model. The CNN model has 171,386,612 total parameters and produce 1.09 GB file. The size causes loading the model into memory takes a long time.

V. Conclusion

Based on the implementation, experiments, and testing, it can be concluded as follows:

1) The font recognition model training with CNN DeepFont was successful with an accuracy of 87% in the training set.
2) The average accuracy of the DeepFont CNN test on the three datasets tested was 7 8.6% with a value of 86% in Dataset_test_50, 77% in Dataset_test_50_noAugmen and 73% in Dataset_test_50_RBKN.
3) The average precision of the DeepFont CNN test on three datasets is 80.7% with a value of 86.2% on Dataset_test_50, 80% on Dataset_test_50_noAugmen and 76% on Dataset_test_50_RBKN.
4) The average recall of CNN DeepFont tests on three datasets was 78.5% with 86.1% in Dataset_test_50, 76.9% in Dataset_test_50_noAugmen and 72.5% in Dataset_test_50_RBKN.
5) The more unique a font is, the easier it is to be recognized by the model while the more similar the font is, the more difficult it is to be recognized by the model.
6) The font recognition application was successfully tested using Alpha Test with 5 test scenarios and 13 test cases.

In addition of the success of DeepFont implementation and integration with application, the model can be improved by adding more font classes and using another form of image augmentation. It is suggested to undertake own generation methods to increase flexibility in managing the distribution of image augmentations and avoid overfitting the model. Another suggestion is to implement better parallel code to avoid loading the CNN more than once to the memory. The application itself can be extended to have *font* searching mechanism to help designers get the recognized fonts.

Conflict of Interest

The authors declare no conflict of interest.

Author Contributions

Bobby Pehtrus did the research; generated both test set and training sets; prepared the experiment; conducted tests; design the font recognition application; analyzed all the results and wrote the paper. Caecilia C. Lestari supervised overall research pipelines and ensures the method applied running well. All authors had approved the final version.

References

[1] Creative Bloq, "Typography rules and terms every designer must know," 2019. [Online]. Available: https://www.creativebloq.com/typography/what-is-typography-123652. [Accessed: 29-Nov-2019].

[2] P. Yadav, D. Chakrabarti, and D. Bisoyi, "Typography as a statement of Design," *Int. Ergon. Conf. HWWE*, no. May, p. 6, 2014.

[3] G. Chen *et al.*, "Large-Scale Visual Font Recognition," in *2014 IEEE Conference on Computer Vision and Pattern Recognition*, 2014, pp. 3598–3605.

[4] Z. Wang *et al.*, "DeepFont: Identify Your Font from An Image," *Proc. 23rd ACM Int. Conf. Multimed. - MM '15*, pp. 451–459, Jul. 2015.

[5] H. Luqman, S. A. Mahmoud, and S. Awaida, "Arabic and farsi font recognition: Survey," *Int. J. Pattern Recognit. Artif. Intell.*, vol. 29, no. 1, 2015.

[6] H. Luqman, S. A. Mahmoud, and S. Awaida, "KAFD Arabic font database," *Pattern Recognit.*, vol. 47, no. 6, pp. 2231–2240, 2014.

[7] A. Borji and M. Hamidi, "Support Vector Machine for Persian Font Recognition," *Eng. Technol.*, vol. 2, no. 3, pp. 10–13, 2007.

[8] H. Wehle, "ML – AI- COGNITIVE," no. August, 2017.

[9] M. Wang and W. Deng, "Deep Visual Domain Adaptation: A Survey," *Neurocomputing*, vol. 312, pp. 135–153, Feb. 2018.

[10] M. Z. Alom *et al.*, "A state-of-the-art survey on deep learning theory and architectures," *Electron.*, vol. 8, no. 3, pp. 1–67, 2019.

[11] K. Simon, "Digital 2020: Indonesia," *DataReportal*, 2020. [Online]. Available: datareportal.com. [Accessed: 25-May-2020].

[12] D. Doermann and K. Tombre, *Handbook of Document Image Processing and Recognition*, no. May. London: Springer London, 2014.

[13] W. Garrick, "Font vs typeface : the ultimate guide," *Creative Bloq*, 2019. [Online]. Available: https://www.creativebloq.com/features/font-vs-typeface. [Accessed: 21-Apr-2020].

[14] Y. Wang, Z. Lian, Y. Tang, and J. Xiao, "Font Recognition in Natural Images via Transfer Learning," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 10704 LNCS, no. September, 2018, pp. 229–240.

[15] C. Tensmeyer, D. Saunders, and T. Martinez, "Convolutional Neural Networks for Font Classification," in *2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR)*, 2017, vol. 1, pp. 985–990.

[16] A. Khan, A. Sohail, U. Zahoora, and A. S. Qureshi, "A Survey of the Recent Architectures of Deep Convolutional Neural Networks," pp. 1–67, 2019.

[17] S. Albawi, T. A. Mohammed, and S. Al-Zawi, "Understanding of a convolutional neural network," *Proc. 2017 Int. Conf. Eng. Technol. ICET 2017*, vol. 2018-Janua, no. August, pp. 1–6, 2018.

[18] A. F. Agarap, "An Architecture Combining Convolutional Neural Network (CNN) and Support Vector Machine (SVM) for Image Classification," pp. 5–8, 2017.

[19] K. Jarrett, K. Kavukcuoglu, M. A. Ranzato, and Y. LeCun, "What is the best multi-stage architecture for object recognition?," in *2009 IEEE 12th International Conference on Computer Vision*, 2009, pp. 2146–2153.

[20] U. Schmid, J. Günther, and K. Diepold, "Stacked Denoising and Stacked Convolutional Autoencoders," 2017.

[21] J. Masci, U. Meier, D. Cireşan, and J. Schmidhuber, "Stacked convolutional auto-encoders for hierarchical feature extraction," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 6791 LNCS, no. PART 1, pp. 52–59, 2011.

[22] H. M and S. M.N, "A Review on Evaluation Metrics for Data Classification Evaluations," *Int. J. Data Min. Knowl. Manag. Process*, vol. 5, no. 2, pp. 01–11, Mar. 2015.

[23] Google, "Colaboratory : Frequently Asked Questions," 2020. [Online]. Available: https://research.google.com/colaboratory/faq.html#resource-limits. [Accessed: 15-Apr-2020].

[24] Tensorflow, "About Tensorflow," 2020. .

[25] Keras, "About Keras," 2020. .

[26] G. Nguyen *et al.*, "Machine Learning and Deep Learning frameworks and libraries for large-scale data mining: a survey," *Artif. Intell. Rev.*, vol. 52, no. 1, pp. 77–124, 2019.

[27] J. Brownlee, *Deep Learning for Computer Vision: Image Classification, Object Detection, and Face Recognition in Python*. Machine Learning Mastery, 2019.

Bobby Pehtrus was born in Medan, Indonesia 1998. He is currently pursuing bachelor degree in informatics at the Faculty of Information Technology, Universitas Ciputra, Surabaya, Indonesia.

His research interest are machine learning and deep learning.