IIETA International Information and Engineering Technology Association
Advancing the World of Information and Engineering

Journal Européen des Systèmes Automatisés
Vol. 57, No. 6, December, 2024, pp. 1603-1611
Journal homepage: http://iieta.org/journals/jesa

Development of a Semantic Text Classification Mobile Application Using TensorFlow Lite and Firebase ML Kit

Dony Novaliendry[1*], Adam Permana[1], Nurindah Dwiyani[2], Noper Ardi[3], Cheng-Hong Yang[4], Fadhillah Majid Saragih[1]

[1] Electronic Department, Universitas Negeri Padang, Padang 25131, Indonesia
[2] Engineering Department, Sekolah Tinggi Ilmu Pelayaran, Jakarta 14150, Indonesia
[3] Informatic Department, Politeknik Negeri Batam, Batam 29461, Indonesia
[4] Electronic Department, National Kaohsiung University Science and Technology, Kaohsiung 807618, Taiwan

Corresponding Author Email: dony.novaliendry@ft.unp.ac.id

https://doi.org/10.18280/jesa.570607

Received: 26 September 2024
Revised: 15 November 2024
Accepted: 21 December 2024
Available online: 31 December 2024

Keywords:
neural networks, semantic text classification, machine learning, TensorFlow Lite, Firebase ML Kit

ABSTRACT

The development of neural networks in the current industrial era 4.0 should help various work fields, one of which is the scientific literature. The problem that often occurs is that scientific papers still use manual sorting of themes/semantics. The purpose of this research is to build a semantic text classification application that can allow users to sort by theme/semantics by using a neural network model, Recurrent Neural Network (RNN) embedded in a smartphone. The development of this application uses the waterfall method in which there are analysis and system design. The application implements the text recognition feature of the Firebase ML Kit. It is developed using a general machine learning cycle method or approach consisting of data identification, data preparation, algorithm selection, model training, model evaluation and model deployment. The model was built using abstract data from scientific papers from the State University of Padang Library. The total data obtained 84 training data and 21 test data using a ratio of 80:20 percent to perform the validation test. The neural network model uses the AverageWordVec specification provided by TensorFlow Lite Model Maker with three classification outputs. The model validation test reached 0.7619 accuracy values with 0.7782 loss values. The model is executed using the TensorFlow Lite interpreter embedded in the application. The application results fulfill the overall system functional requirements analysis.

1. INTRODUCTION

The upcoming industrial revolution is based on a revolutionary technology that substantially impacts industrial production input and output and 3D printing, genetic engineering, and especially artificial intelligence [1]. One of the reasons artificial intelligence plays a role in the 4.0 industrial revolution is the development of adequate infrastructure and a boost to needs due to the increasing data collected by the internet service industry. This large amount of collected data has become difficult for traditional database methods to handle, giving rise to big data. On the way, scientists continue to research handling big data with statistical methods that became the forerunner of machine learning.

Machine learning is a branch of computational algorithms designed to mimic human intelligence by learning from the surrounding environment [2]. There are various algorithms from machine learning ranging from the most straightforward, such as regression, to sophisticated ones such as deep neural networks. Artificial neural network (ANN) or neural network is the most sophisticated algorithm because it is inspired and based on a biological brain structure. This is because ANN allows it to be designed with multiple layers to become a variety of architectural forms. This layered ANN becomes the foundation for sub-branches of the latest machine learning computation algorithms called deep learning.

On November 9, 2015, Google Corporation developed an open-source library called TensorFlow under the Apache 2.0 license, which is only used internally by Google. TensorFlow is a library for symbolic math dataflows popular with machine learning cases, especially neural networks and even deep neural networks. In February 2017, the first version (1.0.0) of TensorFlow was released, then in May 2017, Google released a particular software stack for mobile software development called TensorFlow Lite. TensorFlow Lite is a lightweight version of TensorFlow, capable of being executed faster on mobile devices and even embedded devices such as microcontrollers [3, 4]. The TensorFlow library also provides a unique model maker for TensorFlow Lite called TFLite Model Maker, aiming to simplify the adaptation process and change the TensorFlow neural network model to specific data input.

Obtaining meaning from a text is a sophisticated ability that only the human brain has from the many creatures on earth. Neurons in the human brain can interpret sentences, and even paragraphs in both text and context [5-7]. However, the human

brain has limitations in endurance [8-10]. Besides that, interpreting a text is also not an easy job. It takes analysis and essential knowledge to consume enough energy, mostly if it is done intensively for a long time. This problem is potentially made easier by the existence of TensorFlow as a supporting library for the development of artificial intelligence.

A survey conducted on general students consisting of 27 respondents (16 women and 11 men) found that library visitors often tended to find books and scientific papers that did not fit their category or shelf. Survey details can be seen in Table 1.

Table 1. Library performance questionnaire responses

No.	Question	Response
1	How often did you go to the library (before the pandemic)?	Rarely (22.2%) Quite Rarely (22.2%) Often Enough (33.3%) Always (22.2%)
2	Are you having trouble finding scientific articles that match the theme you want in the library?	Yes (63%) No (29.6%) Maybe (7.4%)
3	Is a search with "keywords" enough to help you find a suitable theme?	Helpful Enough (55.6%) Helpful (37%) Less Helpful (3.7%)
4	How often do you do books and scientific papers that don't fit the category/shelf?	C1/Rarely (11.4%) C2/Quite Rarely (50%) C3/Always (38.4%)

Before implementing the application in the field of library and archiving or other fields, it is necessary to design the initial prototype and test it so that the next researcher will develop this prototype. The application prototype is designed in a modular fashion so that the application is used in one area and can be used in other fields. Further development only needs to change the neural network model with little or no change in application design.

2. LITERATURE REVIEW

2.1 Summary of scientific papers

Scientific paper summaries are generally abstracts derived from research articles, theses, reviews, conference proceedings, or any in depth analysis of a specific subject. They are often used to help readers quickly ascertain the purpose of a paper [11, 12]. Scientific paper summaries are suitable for training data on neural network models due to the limited number of words. The summary of scientific papers in question summarizes the scientific paper of the thesis at the Library of the Department of Electronics, State University of Padang.

Summary / abstracts in scientific writing in the Library of the Department of Electronics, State University of Padang follow the rules and regulations of the Thesis / Final Project Preparation Guide of Padang State University with the following conditions:

•In general, abstracts are arranged in the following order: word abstract, author's name, thesis title, abstract content, and keywords.

•Abstract content is written one space in three paragraphs with a maximum length of 200 words.

•The first paragraph contains a brief description of the problem and research objectives. The second paragraph contains the research method and or approach. The third paragraph contains the research results.

2.2 World embedding and text semantic

Lai et al. [13] stated that word embedding is another term for distributed word representation, capturing the semantic and syntactic information of words from a large unlabeled corpus (collection of writings). Meanwhile, according to Turian et al. [14], word representation is a mathematical object associated with each word, often in a vector. The distributed word representation has the characteristics of being denser, has lower dimensions and has real value than the word representation.

Nowadays, developer discussions, especially machine learning engineers, define word embedding as a collective term for a model that learns to map a series of words or phrases in the vocabulary to vectors of numerical values [15, 16]. Because neural networks are designed to learn from numerical data, word embedding aims to increase neural networks' ability to learn from text data by converting them into vectors. This vector is then called embedding [17, 18].

A common way of handling data is to use a one-hot encoding. This method is very inefficient because most vectors will have a value of 0. Then the output for the multiplication of the matrix is also likely to be 0. The illustration of one hot encoding can be seen in Figure 1.

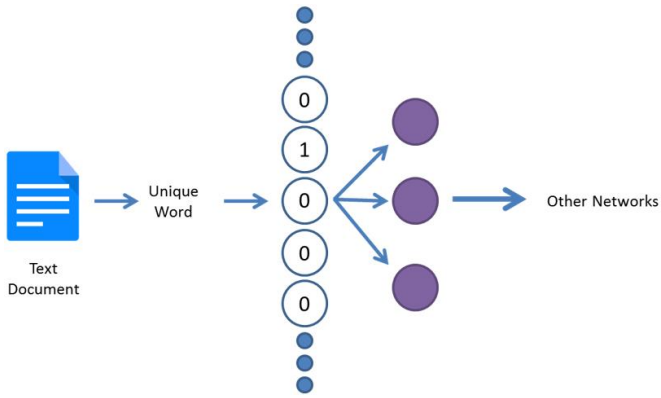

Figure 1. Illustration of one hot encoding

Instead of multiplying the matrix between the input and hidden layers, the value can be taken from the embedding weight matrix, which serves as a lookup table. The following is an illustration of the embedding weight matrix with an example of the word 'eat'.

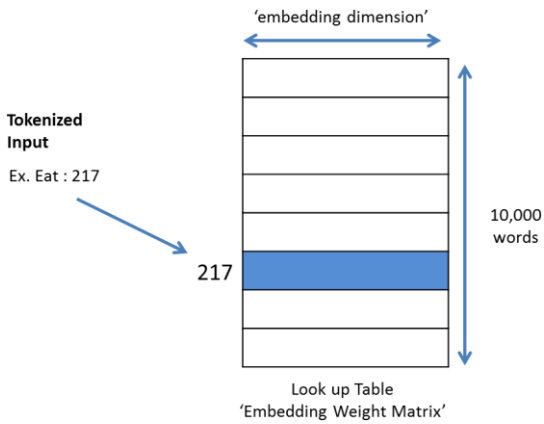

Figure 2. Illustration of embedding weight matrix

Word embedding itself in machine learning can be divided into two categories, namely, unsupervised and supervised. It can be seen in Figure 2.

2.3 Machine learning

Hurwitz, in his book, states that machine learning is a form of artificial intelligence that allows a system to learn from data without the need to be explicitly programmed [19, 20]. Machine learning uses various algorithms that iteratively learn from data to improve, describe data, and predict outcomes. As an algorithm that can digest training data, this algorithm can create the right model based on that data. A machine learning model is an output that is generated when training a machine learning algorithm with data. Several machine learning models are built using neural networks [21, 22].

2.3.1 Neural network

A neural network or artificial neural network by Cross is a machine learning method that consists of a set of processing units (nodes) that simulate neurons and are interconnected through a series of "weights" (similar to synaptic connections in the nervous system) with a way that allows signals to travel through the network both in parallel as well as serial [23].

A neural network is a generalization of a mathematical model based on a biological neural network with the assumption that:

•Information processing resides in many neurons (perceptron).

•Signals are sent between neurons using a connection.

•The connector between these neurons has a weight value that can strengthen or weaken the signal.

•To produce the output, each neuron/perceptron uses various activation functions as needed.

Each neuron unit or perceptron has an activation function that determines the output of the input processing it performs. The output is the value that becomes the neuron impulse that continues to spread/propagate to the last network layer. Modern neural networks do not only forward propagation but also do backpropagation. Backpropagation allows the weight of the connection between perceptron to change automatically with each forward and backward propagation. This forward and backward propagation activity is then called the epoch count. In one epoch, precisely at the last perceptron, there is a calculation of the difference between the actual results and the expected results. There are various functions to determine how much the error (loss) is, so determining the difference between expectations and the actual value of the neural network is called a loss function. Loss function gives high value if model prediction tends to be wrong. Conversely, the higher the model prediction success, the smaller the loss function value. This value helps optimize the model in the next epoch.

The process of model optimization exists in various methods, methods or optimization algorithms, which is then known as the optimizer. Optimizer performs optimization by changing neural network attributes such as weight and learning rate to reduce epoch's loss [24, 25]. There are various optimizers, including Gradient Descent, Stochastic Gradient Descend, LMA, Momentum, RMSprop, Adagrad, AdaDelta, Adam (Adaptive Moment Estimation), and others, each of which has its advantages and advantages in some instances/tasks.

2.4 AverageWordVec

AverageWordVec is the name of a class in the TensorFlow Lite Model Maker library (Figure 3). This class applies the word embedding algorithm to be precise by taking the average (average) of the input word vector. As can be seen in Table 2, AverageWordVec itself has a layered architecture that can be categorized as a deep neural network. This class creates an object instance that will execute sequential modelling with multiple TensorFlow + Keras layers. The following is a neural network sequential model specification with three classifications by this class:

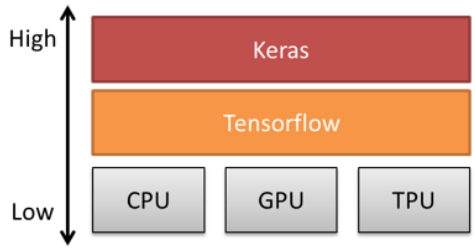

Figure 3. Keras & TensorFlow

Table 2. AverageWordVec sequential model specification by default

Layer Type	Output Shape (batch size, seq_len, wordvec_dim)
Embedding	(None, 256, 16)
Average Pooling 1D	(None, 256, 16)
Dense	(None, 128, 16)
DropOut	(None, 16)
Dense	(None, 3)

The essential parts that need to be considered in using this class include the Keras layers in it, the loss function using the sparse binary cross-entropy and the optimizer, named RMSprop.

3. RESULT AND DISCUSSION

3.1 System analysis

System analysis is the decomposition of a complete system into its parts to identify and evaluate the system's problems, the opportunities and obstacles that occur, and the expected needs so that improvements can be proposed.

Analysis of system requirements is where some material and system requirements will be used to add and assist in doing a project.

Functional requirements in the form of features that exist in applications that are included in Table 3.

Analysis of device requirements is a part that will support the development of the Semantic Text Classifier Application. Table 4 shows the functional Requirement Analysis for designed system.

System Security Analysis is a part that users need to pay attention to when using the Semantic Text Classifier Application. The suggestions for using the system are described in Table 5.

Table 3. Functional requirement analysis

No.	Functionality	Description
1	The apps are capable of capturing images (Camera Capture)	The application uses Android resources to activate the camera and capture the image for processing
2	The apps can read text patterns on images	Using the Text Recognition feature of ML Kit, the application can convert any text in the captured image into a text string
3	The apps can save the existing text as a text file (.txt)	To simplify the data collecting process when building a neural network model, the application provides three buttons to save text strings into category folders in text file format (.txt)
4	The apps can display prediction results	The prediction results display the category along with the percentage value ranging from 0 to 1. Prediction is made through the inference of the TensorFlow Lite model installed in the application
5	The application can save prediction results	Prediction results in the form of recognized text and predicted categories/classes can be saved to nonvolatile storage such as smartphone internal memory or database

Table 4. Functional requirement analysis

No.	Computer System	Description
1	Hardware requirements	Development: Personal Computer with a minimum specification of an Intel Core i5 processor (~ 2.3 GHz) and 4GB RAM (an additional Smartphone is required for debugging) Usage: A mobile phone or tablet with a minimum 8 Megapixel camera
2	Software requirements	Development: IDE Android Studio, Google Collabs (Accessed via web), Java, Kotlin and Android SDK Usage: Android Lollipop operating system (SDK 21)
3	Brainware requirements	Development: Users with an understanding of Android software development and modern neural network concepts (deep neural network) Usage: Does not require specifications. They were designed for the public so that anyone can operate the application

Table 5. Suggestion and consideration

No.	Suggestion	Consideration
1	The entire corpus was caught on camera	Every word that the camera can't read will affect the semantics and even the model training process. It is essential to make sure the entire corpus is caught on camera
2	The captured writings are in the computer printouts form	The model in Firebase Text Recognition is trained through the standard fonts used as well as the handwriting that is significantly distinguished by the computer. Too insignificant handwriting will read differently by Firebase Text Recognition. For example, between "0" and "o", by handwriting that does not pay attention to the space between words, the letter "o" can be read as "0" by the system
3	The position of the camera is straight on the paper/camera catch object	The perspective in processing the captured object image affects the height/length of the character being read

In the design of the semantic text classifier application, the user inputs the camera capture data. The output produced by the application is in the form of classification category data and its confidence value.

Procedure analysis is carried out to determine what processes will be carried out by the system. In the implementation process, it is carried out by established procedures, namely:

-The user opens the apps
-The user captures an image from the camera
-The user informed the scanned text and its prediction

3.2 System design

System design is generally carried out to provide an overview of the system to be made. When the system design is done, the most dominant thing done is modelling the user's needs.

Modelling is done using the unified modeling language. Because some diagrams are straightforward to be modelled with UML, application design is only represented in some UML diagrams, namely use case diagrams, activity diagrams, class diagrams, object diagrams and component diagrams.

3.2.1 Use case diagram

Use case diagrams represent dynamic aspects of the system. Specifically, use case diagrams are used to get system requirements, including internal and external influences.

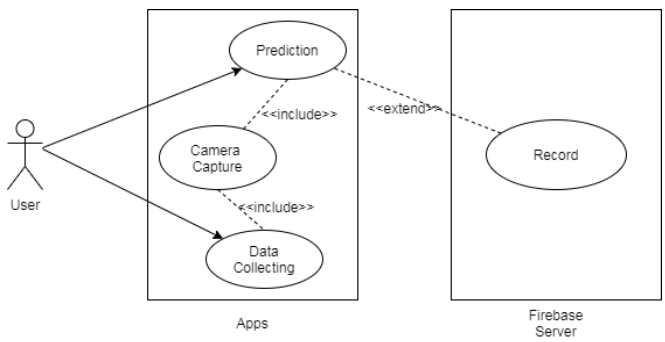

Figure 4. Designed use case diagram

From Figure 4, it can be seen that the requirements that users must have been prediction features and data collection. It only consists of 1 user, in which the prediction feature requires an external system with firebase services. The

prediction feature requires API communication with firebase server to log the prediction results.

3.2.2 Activity diagram

An activity diagram is a flowchart to represent the flow from one activity to another. An activity can be described as an operating system or a process that involves users to interact that must have a user interface. However, in Android software development, an activity is a package of classes and a user interface that has methods to adapt to the Android application lifecycle itself. This application consists of 4 activities, but only two activities need to be designed with an activity diagram. These activities are MainActivity and DataCollectingActivity.

Activity Main or MainActivity is the main feature of the application wherein the menu the user needs to select the "Start!" to begin with. In Figure 5, it can be seen that the processes in this activity apply the multithreading concept. Every instruction in an application is executed with a thread. In an Android application, a thread that can be watched directly by the user is called the main thread or UI thread. Meanwhile, threads other than the main thread are called worker threads. The Classification Thread is a worker thread executed when the constructor of the TextClassificationClient class is executed.

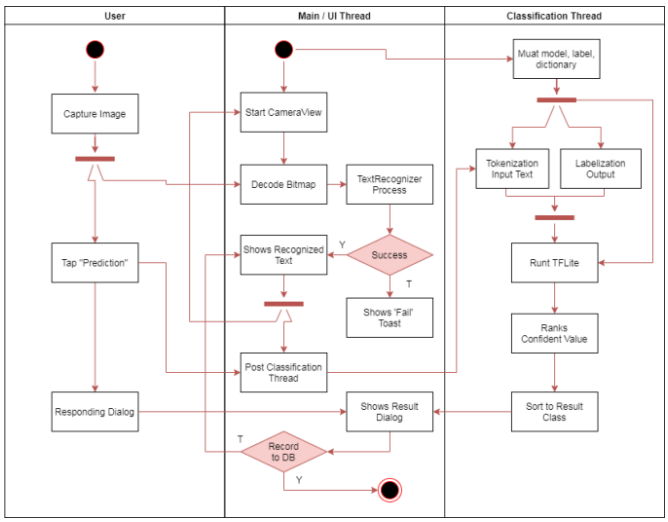

Figure 5. Designed activity diagram for main activity

Activity Data Collecting or Data Collecting Activity is a unique feature created to make it easier for developers to get text data in building a neural network model.

Activity Data Collecting adapts the text recognition feature (Text Recognizer) in the Main Activity. But unlike Activity Main, Activity Data Collecting is simpler. Because this activity aims only to read and then store data in files in several directories/folders. The DataCollecting Activity can be seen in Figure 6.

Class diagrams represent each class in the application. Each class consists of attributes and methods. MainActivity, as a class that forms objects from the main activity, has an aggregation relationship to TextClassificationClient and CounterPrefs, which means that some functions in MainActivity need objects from these two classes. The Result class has a composition relationship to TextClassificationClient, which means that if the TextClassificationClient object is destroyed, the object from the Result class will be excluded. The class diagram can be seen in Figure 7.

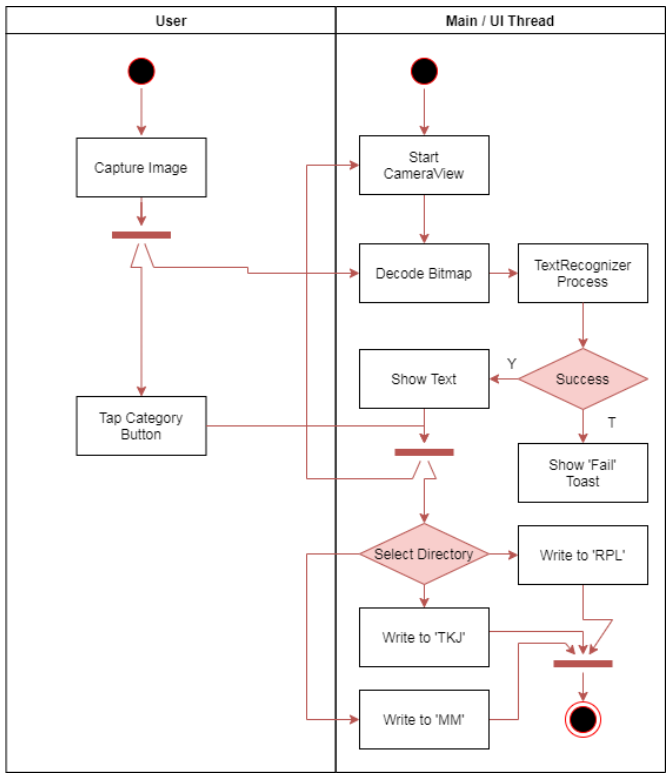

Figure 6. Designed activity diagram for Data Collecting Activity

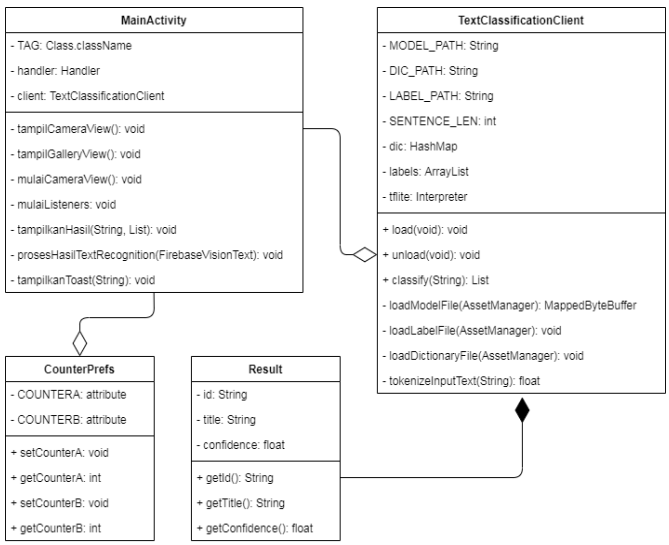

Figure 7. Designed class diagram

Object diagrams represent objects or instances that are formed from existing classes. As shown in Figure 8, each object has a value for each of its attributes. The MainActivity object, as the main thread, will call the handler the TextClassificationClient handler. The TextClassificationClient object forms objects from the Result as many categories as defined by the "label" attribute. Each result object is stacked in the list data structure.

The component diagram represented in Figure 9 describe a set of components and their relationships. In this semantic classifier application design, three components are implemented. Cameraiew consists of several classes that manage camera drivers on Android devices. These classes

include Audio, CameraListener and CameraUtils. Text recognition consists of the FirebaseVision, FirebaseVisionImage, and FirebaseVisionText classes.

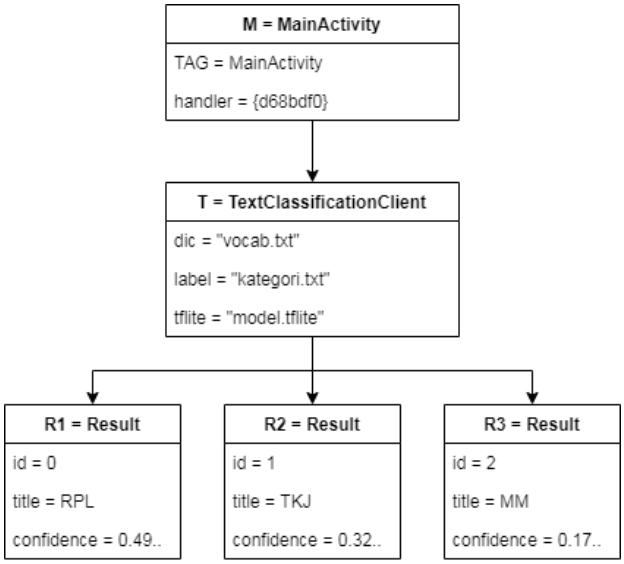

Figure 8. Designed object diagram

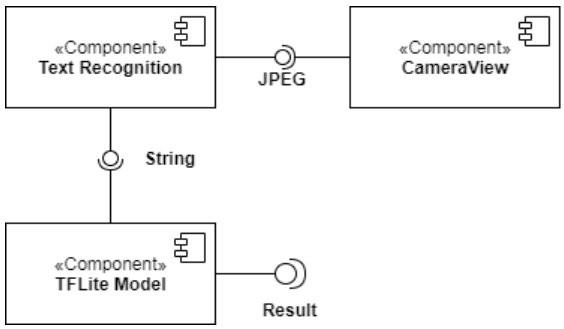

Figure 9. Designed component diagram

Specifically, for developing this application, the researcher chose the AverageWordVec architecture/specification model, a deep neural network model for semantic classification. The loss function used is categorical cross-entropy, with the optimizer is RMSprop.

3.3 Interface implementation

The implementation of the Semantic Text Classifier Application interface design is explained based on each existing activity as follows:

The Main Menu Activity or HomeActivity acts as a launcher and does not require a splash screen as the assets are loaded multithreading (Figure 10). As the name implies, this activity is the main menu in the application.

This activity has 4 Button components that call Intent so that it can move to other activities. There are two types of TextView components: the first to inform the user what model is embedded in the application and the second to identify the application developer's name and institution. The following is the XML code used for this.

In source code 1, the HomeActivity (Main Menu) class is the only class that uses this layout. The layout on the main menu uses ConstraintLayout, a relative layout that binds each component to at least the parent layout. This layout also has a background with colour gradations so that it looks elegant with blue, which symbolizes productivity. There is an icon in the middle of the layout as its identity adjusts colour gradations.

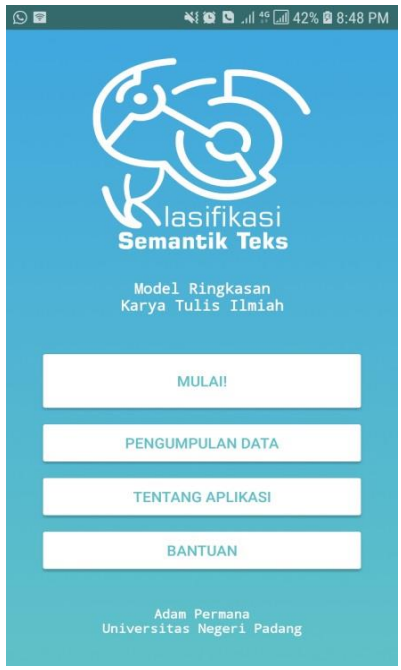

Figure 10. Main Menu interface

Activity of the prediction feature or MainActivity has two layouts that make changes in the middle of the process. The first layout has an id with the name relative_layout_panel_overlay_camera, while the second layout is relative_layout_panel_overlay_result as can be seen in Figure 11.

Layout relative_layout_panel_overlay_camera, or camera overlay for short, is the layout that is called the first time this activity is running. Camera overlay has a UI component, namely CameraView, to capture images. Meanwhile, relative_layout_panel_overlay_result or shortened as result overlay will be loaded after the text recognition process is complete by displaying the text in the Textview component while providing a single button to execute predictions. This single button also displays a dialogue for saving the predicted data to the Firebase Realtime Database. The following is the XML code used for this activity layout.

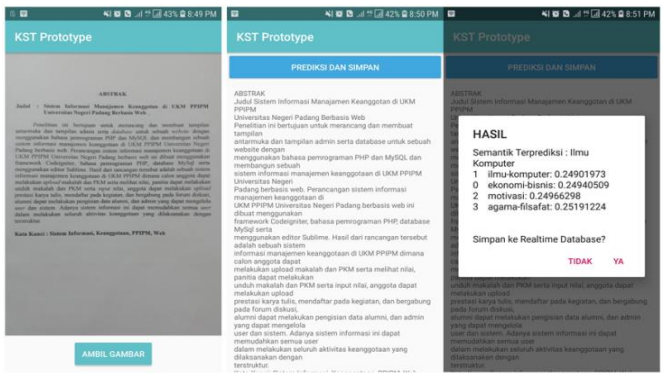

Figure 11. Prediction feature interface

The layout of this activity is not much different from the Prediction feature. Because there is a text recognition process, this activity also has two layouts: the camera overlay and the resulting overlay. It's just that in the resulting overlay, this

layout displays buttons for saving text into a text file (.txt). When pressed, the selected button will flatten, and the system will display the Toast component to tell the user where the file is stored. Figure 12 shows the appearance of this activity.

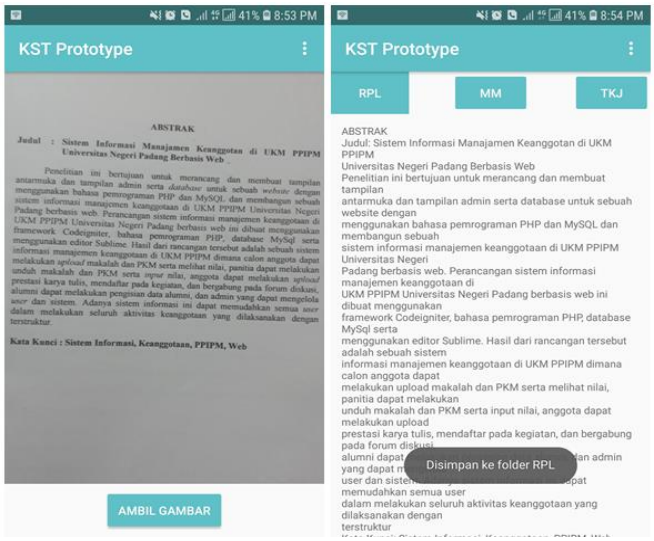

Figure 12. Data collecting interface

The resulting overlay on this activity also provides menu options in the form of a camera button to speed up shooting consecutively. The following is the XML code used for this activity layout.

This activity has the most straightforward layout because it only displays how to use the application. Figure 13 shows the activity interface that was successfully developed.

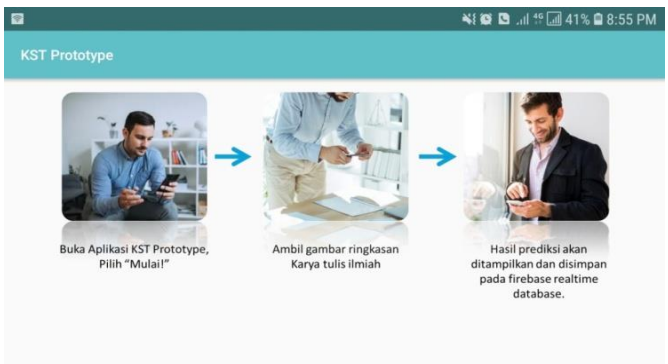

Figure 13. Help page interface

This activity has a landscape orientation when running so that the image looks large and can match its size to all smartphone sizes. The following is the XML code used for this activity layout.

3.4 System implementation

In this subchapter, each source code and program logic is discussed in more detail than in design. Discussion of the system will be discussed as a process specification per each function in one activity. Considering that about and help activities do not have a complicated process, the activities discussed are only predictive and data collecting activities. The process specifications are as follows:

When MainActivity is run from the UI Thread, the active CameraView will serve the user, and the "take a picture" button. In this process, the CameraView class has initiated a listener, which contains an onPictureTaken method in the form of an event.

The onPictureTaken event itself continues the process by using the TextRecognizer to get the text/string on the decoded bitmap image. If the process fails, the activity calls the Toast method to display it to the user. If getting the text is successful, then the activity will change the layout from OverlayCamera to OverlayResult.

In the OverlayResult layout, there are predictive TextView and Button components. This TextView component is used to display text that the TextRecognizer has translated. Meanwhile, Button will trigger to complete the Classification Thread process, namely input tokenization.

Meanwhile, at the same time when MainActivity is first to run, the Classification Thread will contain three components for prediction functions, namely models, labels and dictionaries. Dictionary is a term for a text file representing the Weight Embedding Matrix concept in the Word Embedding algorithm. Labels are text files containing category names where the model only recognizes the labels by order. Meanwhile, the model is a compiled FlatBuffer (.tflite format) file that the TensorFlow Lite Interpreter can only read. The model itself represents a neural network that already has a trained connection pattern from data that has been previously taught (training).

After loading the three components of the prediction function, the Classification Thread cannot proceed to the interpretation process because the model requires tokenized text input. Tokenization is a process that represents a look-up on a dictionary or weight embedding matrix. Therefore, the Classification Thread will wait for the UI Thread to parse the parameter to one of its methods, the classification method.

After the user presses the "prediction" button on the OverlayResult layout, the UI Thread communicates with the Classification Thread using the Handler's post method. The text that TextRecognizer has read can be parsed to the classification method. At this stage, the classification method will run the text input tokenization algorithm. Simply put, this algorithm performs string comparisons from text input against a list of texts in the dictionary, then takes word-for-word numeric values to be the input of the neural network model. The tokenization algorithm is applied to the tokenizeInputText method, which includes data cleaning techniques such as ensuring lowercase text input, deleting punctuation marks such as commas, question marks, minuses, and others. The value of the word for word numbers is stored into an array with the float data type to be directly executed by the Interpreter.

The interpreter's execution (in the form of confidence value) is processed in a ranking algorithm using the PriorityQueue class, which has a FIFO (First In First Out) concept by making a lambda function in it to perform comparisons. By PriorityQueue, the value is then used to call the label according to the predicted confidence value. These labels are then accommodated into an ArrayList ready to be displayed in the Dialog.

In the dialog, a response button is provided to save the predicted results to the Firebase Realtime Database. It is provided for further development, allowing users to manage the semantic prediction data of scientific papers.

3.5 Neural network model result

The text recognition feature (TextRecognizer) in

MainActivity (prediction feature) is the same code used in DataCollectingActivity. After getting a text with ML Kit TextRecognizer, the user will be provided with three buttons to select the folder where the text file is stored. Shared Preference is applied as a counter of the used file names to save files with different names so that no text file is saved with the same name. The total data obtained 84 training data and 21 test data using a ratio of 80:20 percent. The data were split using stratified random sampling.

The neural network model design for the Semantic Text Classifier Application is explained based on three advanced stages of the machine learning cycle. These stages are the compilation process (training data), data evaluation and the deployed model.

In the training stage, the data is trained with the parameters that the AverageWordVec model specification has determined. This model provides loss and accuracy information for each epoch. At this stage, the developer trains 100 epoch models and achieves the following accuracy values. The data can be seen in Figure 14.

```
Epoch 95/100
2/2 [==============================] - 0s 22ms/step - loss: 0.5513 - accuracy: 0.9167
Epoch 96/100
2/2 [==============================] - 0s 25ms/step - loss: 0.5669 - accuracy: 0.8854
Epoch 97/100
2/2 [==============================] - 0s 20ms/step - loss: 0.6152 - accuracy: 0.8229
Epoch 98/100
2/2 [==============================] - 0s 21ms/step - loss: 0.5649 - accuracy: 0.8542
Epoch 99/100
2/2 [==============================] - 0s 23ms/step - loss: 0.5357 - accuracy: 0.9479
Epoch 100/100
2/2 [==============================] - 0s 19ms/step - loss: 0.5381 - accuracy: 0.9167
```

Figure 14. Loss and accuracy value at 100th epoch

At this stage, the developer evaluates by calculating the model's accuracy and loss using data testing. The model's performance was measured using recall and F1-score metrics, yielding promising results.

If you pay attention to the value of accuracy and loss using test_data is lower than the training_data carried out as can be seen in Figure 15. This is because the data is still small, so the pattern data obtained from the training data is not sufficient to represent the testing data's suitability.

```
model.summary()

Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 256, 16)           40768
_________________________________________________________________
global_average_pooling1d (Gl (None, 16)                0
_________________________________________________________________
dense (Dense)                (None, 16)                272
_________________________________________________________________
dropout (Dropout)            (None, 16)                0
_________________________________________________________________
dense_1 (Dense)              (None, 3)                 51
=================================================================
Total params: 41,091
Trainable params: 41,091
Non-trainable params: 0
_________________________________________________________________
```

```
loss, acc = model.evaluate(test_data)

1/1 [==============================] - 0s 131ms/step - loss: 0.7782 - accuracy: 0.7619
```

Figure 15. Model evaluation result

The deploy model process determines the format of the model being built. In developing this model, data is exported in the TensorFlow Lite flat buffer format (.tflite) as can be seen in Figure 16. In version 0.2.2, the AverageWordVec model specification exports the vocab file and labels by default to the model metadata.

```
model.export(export_dir='.')

INFO:tensorflow:Assets written to: /tmp/tmpaeph8kwq/assets
INFO:tensorflow:Assets written to: /tmp/tmpaeph8kwq/assets
INFO:tensorflow:Vocab file and label file are inside the TFLite model with metadata.
INFO:tensorflow:Vocab file and label file are inside the TFLite model with metadata.
INFO:tensorflow:Saved vocabulary in /tmp/tmpazpl4tul/vocab.txt.
INFO:tensorflow:Saved vocabulary in /tmp/tmpazpl4tul/vocab.txt.
INFO:tensorflow:Saving labels in /tmp/tmpazpl4tul/labels.txt.
INFO:tensorflow:Saving labels in /tmp/tmpazpl4tul/labels.txt.
Finished populating metadata and associated file to the model:
./model.tflite
The metadata json file has been saved to:
./model.json
The associated file that has been packed to the model is:
['vocab.txt', 'labels.txt']
/usr/local/lib/python3.6/dist-packages/tensorflow_lite_support/metadata/metadata.py:344:
  "tflite model is still allowed.".format(f))
```

Figure 16. Model deploy

4. CONCLUSION

This analysis examines scientific papers that use the waterfall software development method to design and develop semantic text classifier applications using Firebase ML Kit technology with TensorFlow Lite. The purpose of this analysis is to provide insights into the effectiveness of these technologies and identify potential implementation challenges.

The findings suggest that Firebase ML Kit technology with TensorFlow Lite can be relied on to build robust semantic text classifier applications with efficient performance. However, an important challenge is the development of a well-designed neural network model that can accurately classification and categorize textual data. Inadequate training data can lead to suboptimal performance, highlighting the need for ongoing refinement of the neural network model and training data set to improve accuracy.

This study also emphasizes the importance of systematic planning and execution in the software development process. Using the waterfall software development method, developers can identify and address potential issues early on, leading to a more efficient and effective approach.

The prototype testing of the overall system analysis was successfully realized from the initial design, highlighting the potential of the waterfall software development method and the effectiveness of Firebase ML Kit technology with TensorFlow Lite in creating reliable semantic text classifier applications. User acceptance testing (UAT) of the TensorFlow Lite Summary Text Classification application for scientific work was successfully conducted with five academic experts. The application performed as expected, meeting all user requirements.

In conclusion, the use of Firebase ML Kit technology with TensorFlow Lite in building semantic text classifier applications is promising. While challenges exist in developing these applications, the benefits of these technologies outweigh the drawbacks. This study emphasizes the importance of designing the neural network model carefully and continually refining the training data set to improve performance. The use of the waterfall software development method ensures a systematic approach to the development process, ultimately leading to the successful realization of the prototype testing. This study highlights the potential of Firebase ML Kit technology with TensorFlow Lite and the need for further research and development to expand the potential use cases of these technologies.

ACKNOWLEDGMENT

The authors appreciate the support from the National Kaohsiung University of Science and Technology, Taiwan, and Universitas Negeri Padang, Indonesia.

REFERENCES

[1] Garrett, B. (2013). An Emerging Third Industrial Revolution. Atlantic Council.

[2] El Naqa, I., Murphy, M.J. (2015). What is Machine Learning? Springer International Publishing.

[3] Alsing, O. (2018). Mobile object detection using tensorflow lite and transfer learning. Master thesis, KTH Royal Institute of Technology.

[4] Ardi, N., Supardianto, S., Lubis, A.I. (2023). Predicting missing value data on IEC TC10 datasets for dissolved gas analysis using tertius algorithm. Journal of Applied Informatics and Computing, 7(1): 44-50.

[5] Novaliendry, D., Darmi, R., Hendriyani, Y., Nor, M., Azman, A. (2020). Smart learning media based on android technology. International Journal of Innovation, Creativity and Change, 12(11): 715-735.

[6] Krismadinata, U.V., Jalinus, N., Rizal, F., Sukardi, P.S., et al. (2020). Blended learning as instructional model in vocational education: Literature review. Universal Journal of Educational Research, 8(11B): 5801-5815. https://doi.org/10.13189/ujer.2020.082214

[7] Novaliendry, D., Adri, M., Sriwahyuni, T., Huda, A., Huda, Y., Irfan, D., Jaya, P., Ramadhani, D., Anori, S. (2020). Development of smart learning media model based on Android. International Journal of Engineering Research and Technology, 13(12): 5354-5364.

[8] Novaliendry, D., Wattimenac, F.Y., Renyaan, A.S., Lubis, A.L., Ramadhani, D., Lizar, Y., Guci, A. (2020). Development of an expert system application to detect vitamin deficiencies in the human body. International Journal of Early Childhood Special Education, 29(5): 956. https://doi.org/10.24205/03276716.2020.1092

[9] Renyaand, Y.L., Gucif, A., Ariyong, M., Ramadhanih, D., et al. (2020). Prediction of mortalityinthe hemodialysis patient with diabetes using support vector machine. Revista Argentina de Clínica Psicológica, 29(4): 219-232.

[10] Novaliendry, D., Hendriyani, Y., Yang, C.H., Hamimi, H. (2015). The optimized K-means clustering algorithms to analyzed the budget revenue expenditure in Padang. Proceeding of the Electrical Engineering Computer Science and Informatics, 2(1): 61-66.

[11] Blake, G., Bly, R.W. (1993). The Elements of Technical Writing. New York, NY: Macmillan.

[12] Ardi, N., Setiawan, N.A., Adji, T.B. (2019). Analytical incremental learning for power transformer incipient fault diagnosis based on dissolved gas analysis. In 2019 5th International Conference on Science and Technology (ICST), Yogyakarta, Indonesia, pp. 1-4. https://doi.org/10.1109/ICST47872.2019.9166441

[13] Lai, S., Liu, K., He, S., Zhao, J. (2016). How to generate a good word embedding. IEEE Intelligent Systems, 31(6): 5-14. https://doi.org/10.1109/MIS.2016.45

[14] Turian, J., Ratinov, L., Bengio, Y. (2010). Word representations: A simple and general method for semi-supervised learning. In Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics, Uppsala, Sweden, pp. 384-394.

[15] Cao, H., Dong, C. (2022). An English pronunciation error detection system based on improved random forest. Mobile Information Systems, 2022(1): 6457286. https://doi.org/10.1155/2022/6457286

[16] Wang, C. (2021). Efficient English translation method and analysis based on the hybrid neural network. Mobile Information Systems, 2021(1): 9985251. https://doi.org/10.1155/2021/9985251

[17] Ji, G., Zheng, Y. (2022). Automatic lane line detection system based on artificial intelligence. Journal of Electrical and Computer Engineering, 2022(1): 5284185. https://doi.org/10.1155/2022/5284185

[18] Ma, J., Ye, X., Huang, K. (2022). Development of integrated choice and latent variable (ICLV) models using matrix-based analytic approximation and automatic differentiation methods on tensorflow platform. Journal of Advanced Transportation, 2022(1): 6556282. https://doi.org/10.1155/2022/6556282

[19] Hurwitz, J., Kirsch, D. (2018). Machine Learning for Dummies. IBM Limited Edition.

[20] Huda, A., Ardi, N. (2021). Predictive analytic on human resource department data based on uncertain numeric features classification. International Journal of Interactive Mobile Technologies, 15(8): 172-181. https://doi.org/10.3991/ijim.v15i08.20907

[21] Jiang, D., He, Z., Lin, Y., Chen, Y., Xu, L. (2021). An improved unsupervised single-channel speech separation algorithm for processing speech sensor signals. Wireless Communications and Mobile Computing, 2021(1): 6655125. https://doi.org/10.1155/2021/6655125

[22] Broll, B., Timalsina, U., Völgyesi, P., Budavári, T., Lédeczi, Á. (2020). A machine learning gateway for scientific workflow design. Scientific Programming, 2020(1): 8867380. https://doi.org/10.1155/2020/8867380

[23] Cross, S.S., Harrison, R.F., Kennedy, R.L. (1995). Introduction to neural networks. The Lancet, 346(8982): 1075-1079.

[24] Saputra, D., Handani, S., Indartono, K., Wijanarko, A. (2020). SMART-in English: Learn English using speech recognition. Journal of Robotics and Control, 1(4): 109-113. https://doi.org/10.18196/jrc.1423

[25] Ketkar, N. (2017). Introduction to Keras. In Deep Learning with Python, pp. 97-111. https://doi.org/10.1007/978-1-4842-2766-4_7